RETT, a Reasonably Exceptional Type Theory

Pierre-Marie Pédrot[1],
Nicolas Tabareau[1], Hans Fehrmann[2], Éric Tanter[1, 2]

[1]INRIA, [2]University of Chile

ICFP 2019

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

CIC, the Calculus of Inductive Constructions.

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'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

The Cake is Not Not a Lie

¿ CIC, a very powerful **functional** **programming language** ?

The Cake is Not Not a Lie

¿ CIC, a very powerful functional **programming language** ?

... as long as you can live in purely functional fantasyland

- No native effects (not even non-termination!)
- Haskell monadic style awkward with dependent types
- What is even the point of using Coq then?

The Cake is Not Not a Lie

¿ CIC, a very powerful functional **programming language** ?

… as long as you can live in purely functional fantasyland

- No native effects (not even non-termination!)
- Haskell monadic style awkward with dependent types
- What is even the point of using Coq then?

CIC, a not so great **effectful** **programming language** ☹

Tainting CIC with Impurities

We have been working on extending CIC with **side-effects**.

Tainting CIC with Impurities

We have been working on extending CIC with **side-effects**.

- To `program` more, but also to *prove* more
- Justification via compilation (more on that soon)
- A lot of interesting stuff to say but time is pressing

Tainting CIC with Impurities

We have been working on extending CIC with **side-effects**.

- To `program` more, but also to **prove** more
- Justification via compilation (more on that soon)
- A lot of interesting stuff to say but time is pressing

Effect *du jour*

We will concentrate today on only one particular, simple effect.

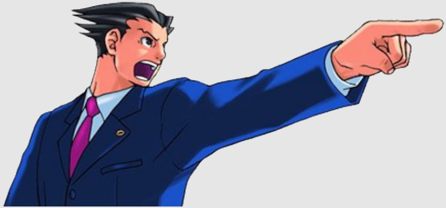
Tainting CIC with Impurities

We have been working on extending CIC with **side-effects**.

- To `program` more, but also to *prove* more
- Justification via compilation (more on that soon)
- A lot of interesting stuff to say but time is pressing

Effect *du jour*

We will concentrate today on only one particular, simple effect.



EXCEPTIONS!

Summary of the Previous Episodes

Pédrot & Tabareau, ESOP 2018

ExTT, an extension of CIC with **exceptions**.

- ▷ Add a failure mechanism to CIC
- ▷ Fully computational **call-by-name** exceptions
- ▷ Contains the whole of CIC (including krazy dependent stuff)
- ▷ Compiled away to vanilla CIC (so-called syntactic model)

Summary of the Previous Episodes

Pédrot & Tabareau, ESOP 2018

ExTT, an extension of CIC with **exceptions**.

- Add a failure mechanism to CIC
- Fully computational **call-by-name** exceptions
- Contains the whole of CIC (including krazy dependent stuff)
- Compiled away to vanilla CIC (so-called syntactic model)

Let's have a look at ExTT!

The Exceptional Type Theory: Overview

ExTT extends CIC with an **exception-raising** primitive (ITT: no payload).

$$\texttt{raise} \quad : \quad \Pi A : \square.\, A$$

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

The Exceptional Type Theory: Overview

ExTT extends CIC with an **exception-raising** primitive (ITT: no payload).

$$\texttt{raise} \;:\; \Pi A : \square.\, A$$

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

They can be **caught** on inductive types via a generalization of eliminators.

$$\begin{array}{ll} \mathbb{B}_{\texttt{rec}} : & \Pi P : \mathbb{B} \to \square. \\ & \quad P\ \texttt{true} \to \\ & \quad P\ \texttt{false} \to \\ & \\ & \Pi b : \mathbb{B}.\, P\ b \end{array} \quad \rightsquigarrow \quad \begin{array}{ll} \texttt{catch}_{\mathbb{B}} : & \Pi P : \mathbb{B} \to \square. \\ & \quad P\ \texttt{true} \to \\ & \quad P\ \texttt{false} \to \\ & \quad \color{red}{P\ (\texttt{raise}\ \mathbb{B}) \to} \\ & \Pi b : \mathbb{B}.\, P\ b \end{array}$$

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

Exception is the Rule

While a fairly simple effect, exceptions are already super useful!

While a fairly simple effect, exceptions are already super useful!

Dead code stays so.

« Come on Coq, I know this branch cannot occur! »

Use `raise`.

Exception is the Rule

While a fairly simple effect, exceptions are already super useful!

Dead code stays so.

« Come on Coq, I know this branch cannot occur! »

Use `raise`.

Post-hoc reasoning.

« Why do I need to thread decidable hypotheses everywhere? »

Use `raise`.

Exception is the Rule

While a fairly simple effect, exceptions are already super useful!

Dead code stays so.

« Come on Coq, I know this branch cannot occur! »

Use `raise`.

Post-hoc reasoning.

« Why do I need to thread decidable hypotheses everywhere? »

Use `raise`.

Failure as a default.

« Why on earth do I have to return an option? »

Use `raise`.

Typical problems from the wild: mathcomp, hs-to-coq...

Bottom Model

How do we prove that ExTT makes any sense?

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

Bottom Model

How do we prove that ExTT makes any sense?

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

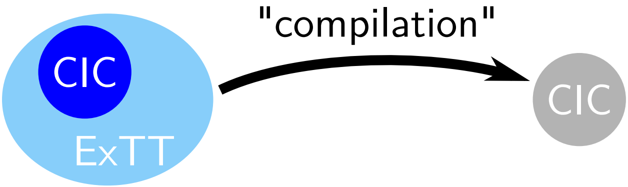
We want a **model of** the exceptional type theory!

Bottom Model

How do we prove that ExTT makes any sense?

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

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



« CIC, *the LLVM of Type Theory* »

The Exceptional Implementation (sketch)

A Truly Simple Model!

$$\vdash_{\mathsf{ExTT}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [\![A]\!] : \square \quad + \quad \vdash_{\mathsf{CIC}} [A]_{\varnothing} : [\![A]\!]$$

$$\vdash_{\mathsf{ExTT}} M : A \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

Every exceptional type comes with its own implementation of failure!

The Exceptional Implementation (sketch)

A Truly Simple Model!

$$\vdash_{\mathsf{ExTT}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [\![A]\!] : \square \quad + \quad \vdash_{\mathsf{CIC}} [A]_{\varnothing} : [\![A]\!]$$

$$\vdash_{\mathsf{ExTT}} M : A \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

Every exceptional type comes with its own implementation of failure!

$$\begin{array}{lcl} [\![\square]\!] & := & \Sigma A : \square.\, A \\ [\![\Pi x : A.\, B]\!] & := & \Pi x : [\![A]\!].\, [\![B]\!] \end{array}$$

The Exceptional Implementation (sketch)

A Truly Simple Model!

$$\vdash_{\mathsf{ExTT}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [\![A]\!] : \square \quad + \quad \vdash_{\mathsf{CIC}} [A]_{\varnothing} : [\![A]\!]$$

$$\vdash_{\mathsf{ExTT}} M : A \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

Every exceptional type comes with its own implementation of failure!

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

The Exceptional Implementation (sketch)

A Truly Simple Model!

$$\vdash_{\mathsf{ExTT}} A : \square \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [\![A]\!] : \square \quad + \quad \vdash_{\mathsf{CIC}} [A]_{\varnothing} : [\![A]\!]$$

$$\vdash_{\mathsf{ExTT}} M : A \quad \rightsquigarrow \quad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

Every exceptional type comes with its own implementation of failure!

$$\begin{array}{lcl} [\![\square]\!] & := & \Sigma A : \square.\, A \\ [\![\Pi x : A.\, B]\!] & := & \Pi x : [\![A]\!].\, [\![B]\!] \\ & & \\ [\square]_{\varnothing} & := & \ldots \\ [\Pi x : A.\, B]_{\varnothing} & := & \lambda x : [\![A]\!].\, [B]_{\varnothing} \\ & & \\ [M] & := & \ldots \\ [\mathtt{raise}\ A] & := & [A]_{\varnothing} \end{array}$$

The Exceptional Implementation, Positive case

Add an error case to every inductive type!

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

The Exceptional Implementation, Positive case

Add an error case to every inductive type!

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

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 $\mathbb{B}_\varnothing$, **reraise** using $[P\ b]_\varnothing$

Where is the fish?

Theorem

ExTT *contains* CIC...

Where is the fish?

Theorem

ExTT *contains* CIC...*but it also proves everything.* ☹ *(Use* `raise`*!)*

Where is the fish?

Theorem

ExTT *contains* CIC...*but it also proves everything.* ☹ *(Use* `raise`*!)*

An Impure Dependently-typed Programming Language

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

Where is the fish?

Theorem

ExTT *contains* CIC...*but it also proves everything.* ☹ *(Use* `raise`*!)*

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 $\vdash_{\text{ExTT}} M : \bot$, *then* $M \equiv \texttt{raise}\ \bot$.

With Great Effects Come Great Responsibility

In ESOP 2018 we described pExTT, a **consistent** restriction of ExTT.

- Variant of Bernardy-Lasson style parametricity (syntactic model)
- Toplevel exceptions forbidden, but can still be raised locally (meh)

$$\text{CIC} \subsetneq \text{pExTT} \subsetneq \text{ExTT}$$

With Great Effects Come Great Responsibility

In ESOP 2018 we described pExTT, a **consistent** restriction of ExTT.

- Variant of Bernardy-Lasson style parametricity (syntactic model)
- Toplevel exceptions forbidden, but can still be raised locally (meh)

$$\text{CIC} \subsetneq \text{pExTT} \subsetneq \text{ExTT}$$

Now we have a dilemma!

ExTT	pExTT
☹ Inconsistent	☺ Consistent
☺ Unrestricted use of exceptions	☹ Exceptions virtually unusable
☺ Good for programming	☹ Strange logical properties

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

RETT 3in1

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

RETT 3in1

with not one, not two, but **three** universe hierarchies

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

RETT 3in1

with not one, not two, but **three** universe hierarchies

Pure Layer

$\square_i^{\mathrm{p}} \sim$ CIC

- ▷ Consistent
- ▷ No exceptions
- ▷ For proving

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

RETT **3in1**

with not one, not two, but **three** universe hierarchies

Pure Layer	Exceptional Layer
$\square_i^{\mathsf{p}} \sim$ CIC	$\square_i^{\mathsf{e}} \sim$ ExTT

- ▷ Consistent
- ▷ No exceptions
- ▷ For proving

- ▷ Inconsistent
- ▷ Full exceptions
- ▷ For programming

CIC $\subsetneq$ pExTT $\subsetneq$ ExTT

Tired to have to make a choice? We have the answer!

RETT **3in1**

with not one, not two, but **three** universe hierarchies

Pure Layer	Exceptional Layer	Mediating Layer
$\square_i^{\mathrm{p}} \sim$ CIC	$\square_i^{\mathrm{e}} \sim$ ExTT	$\square_i^{\mathrm{m}} \sim$ pExTT

- Consistent
- No exceptions
- For proving

- Inconsistent
- Full exceptions
- For programming

- Consistent
- Local exceptions
- For communication

At the Crossroads

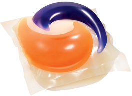
Every hierarchy in isolation behaves as a variant of CIC

$\square_i^{\mathtt{p}} \sim \mathsf{CIC}$ $\square_i^{\mathtt{e}} \sim \mathsf{ExTT}$ $\square_i^{\mathtt{m}} \sim \mathsf{pExTT}$

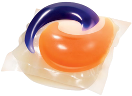
At the Crossroads

Every hierarchy in isolation behaves as a variant of CIC

$\square_i^{\mathtt{p}} \sim$ CIC $\square_i^{\mathtt{e}} \sim$ ExTT $\square_i^{\mathtt{m}} \sim$ pExTT



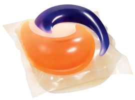
“`Write programs in` $\square^{\mathtt{e}}$,
Prove them in $\square^{\mathtt{m}}$ or $\square^{\mathtt{p}}$.”



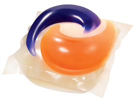
At the Crossroads

Every hierarchy in isolation behaves as a variant of CIC

$\square_i^{\mathtt{p}} \sim$ CIC $\square_i^{\mathtt{e}} \sim$ ExTT $\square_i^{\mathtt{m}} \sim$ pExTT



"Write programs in $\square^{\mathtt{e}}$,
Prove them in $\square^{\mathtt{m}}$ or $\square^{\mathtt{p}}$."



The expressivity of RETT lies in the interaction between hierarchies

$$\frac{\Gamma \vdash A : \square_i^{\alpha} \qquad \Gamma, x : A \vdash B : \square_j^{\beta} \qquad \alpha, \beta \in \{\mathtt{p}, \mathtt{e}, \mathtt{m}\}}{\Gamma \vdash \Pi(x : A).\, B : \square_{i \vee j}^{\beta}}$$

Eliminating inductive types is even more interesting

CIC	$\mathbb{B}_{\texttt{rec}} : \quad \Pi P : \mathbb{B} \to \square.\, P\ \texttt{true} \to P\ \texttt{false} \to \Pi b : \mathbb{B}.\, P\ b$
ExTT	$\mathbb{B}_{\texttt{catch}} : \Pi P : \mathbb{B} \to \square.\, P\ \texttt{true} \to P\ \texttt{false} \to P\ (\texttt{raise}\ \mathbb{B}) \to \Pi b : \mathbb{B}.\, P\ b$

Eliminating inductive types is even more interesting

CIC	$\mathbb{B}_{\texttt{rec}} : \quad \Pi P : \mathbb{B} \to \square.\ P\ \texttt{true} \to P\ \texttt{false} \to \Pi b : \mathbb{B}.\ P\ b$
ExTT	$\mathbb{B}_{\texttt{catch}} : \Pi P : \mathbb{B} \to \square.\ P\ \texttt{true} \to P\ \texttt{false} \to P\ (\texttt{raise}\ \mathbb{B}) \to \Pi b : \mathbb{B}.\ P\ b$

Depending on the hierarchy of $\mathbb{B}$ and P we get different eliminators!

		$P : \mathbb{B} \to \square^{\beta}$		
		$\square^{\mathtt{e}}$	$\square^{\mathtt{m}}$	$\square^{\mathtt{p}}$
$\mathbb{B} : \square^{\alpha}$	$\square^{\mathtt{e}}$	rec/catch	catch	catch
	$\square^{\mathtt{m}}$	rec	rec	—
	$\square^{\mathtt{p}}$	rec	rec	rec

- `catch` does not make sense when source is consistent
- `catch` is mandatory when eliminating from inconsistent to consistent
- reminiscent of the singleton elimination restriction in CIC.

And Much More

We also have modalities for better interoperability

$$\{-\}^{\alpha}_{\beta} \quad : \quad \square^{\alpha} \to \square^{\beta}$$
$$\iota^{\alpha}_{\beta} \quad : \quad \Pi(A : \square^{\alpha}).\, A \to \{A\}^{\alpha}_{\beta}$$

... as well as an internal purity predicate

$$\mathcal{P} : \Pi(A : \square^{\mathtt{m}}).\, \{A\}^{\mathtt{m}}_{\mathtt{e}} \to \square^{\mathtt{m}}$$

$$\Sigma(x : \{A\}^{\mathtt{m}}_{\mathtt{e}}).\, \mathcal{P}\ A\ x \quad \cong \quad A$$

Main interest of $\square^{\mathtt{m}}$ over $\square^{\mathtt{p}}$.

(A lot to say, but I don't have time.)

CoqRETT

We implemented RETT as a POC Coq plugin.

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

- Allows to add exceptions to Coq just today.
- Piggybacks on the `Prop`/`Type` segregation (hack hack hack)
- Compile RETT on the fly.
- Not really practical though, should this go into the kernel?

TODO

- Actually provide RETT first class in Coq?
- Use it for programming for realz?
- Potential applications to Gradual Typing?
- One hierarchy = one effect?

TODO

- Actually provide RETT first class in Coq?
- Use it for programming for realz?
- Potential applications to Gradual Typing?
- One hierarchy = one effect?

If You Were Sleeping During The Talk

RETT, a 3-in-1 type theory!

1. An **inconsistent** dependently-typed effectful programming language
2. A **consistent** dependently-typed proof language
3. A **consistent** dependently-typed mediating language

Smoothly interacting together!

All of this justified by purely syntactical means!

Implemented in your favourite proof assistant!