Effects, Substitution and Induction
An Explosive Ménage à Trois

Pierre-Marie Pédrot, Nicolas Tabareau

INRIA

TYPES 2019
13th June 2019
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
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
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
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.
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?
Intuitionistic Logic $\Leftrightarrow$ Functional Programming
The Most Important Issue of Them All, Bis

Intuitionistic Logic ⇔ Functional Programming

Thus, the same problem for mathematically inclined users.
The Most Important Issue of Them All, Bis

Intuitionistic Logic ⇔ Functional Programming

Thus, the same problem for mathematically inclined users.

HOW DO I REASON CLASSICALLY?
Intuitionistic Logic ⇔ Functional Programming

Thus, the same problem for mathematically inclined users.

How do I reason classically?
We want a type theory with effects!
We want a type theory with effects!

To program more!

Non-termination
Exceptions
State...

To prove more!

Classical logic
Univalence
Choice...
Classical logic does not play well with type theory.

Barthe and Uustalu: CPS cannot interpret dependent elimination

Herbelin’s paradox: CIC + callcc is unsound!
Classical logic does not play well with type theory.

Barthe and Uustalu: CPS cannot interpret dependent elimination

Herbelin’s paradox: CIC + callcc is unsound!

We have been working on effectful type theories

We are not the only ones, but our specialty:
Classical logic does not play well with type theory.

Barthe and Uustalu: CPS cannot interpret dependent elimination

Herbelin’s paradox: CIC + callcc is unsound!

We have been working on effectful type theories

We are not the only ones, but our specialty:

We justify them through program translations into CIC itself.

Forcing, reader monad, exceptions, free algebraic...
Classical logic does not play well with type theory.

Barthe and Uustalu: CPS cannot interpret dependent elimination

Herbelin’s paradox: CIC + callcc is unsound!

We have been working on effectful type theories

We are not the only ones, but our specialty:

We justify them through program translations into CIC itself.

Forcing, reader monad, exceptions, free algebraic...

Effectful theories are always half-broken

dependent elimination has to be restricted (BTT)
or consistency forsaken, or worse
This is not a coincidence!

Herbelin / Barthe-Uustalu results are instances of a generic phenomenon!
This is not a coincidence!

Herbelin / Barthe-Uustalu results are instances of a generic phenomenon!

Also, this is kind of folklore.

... but I don’t recall reading it formally anywhere.
Definition

A type theory has **observable effects** if there is a closed term \( t : \mathbb{B} \) that is not observationally equivalent to a value, i.e. there is a context \( C[\cdot] \) s.t.

\[
C[\text{true}] \equiv \text{true} \quad \text{and} \quad C[\text{false}] \equiv \text{true} \quad \text{but} \quad C[t] \equiv \text{false}
\]

This happens for many kind of effects, including continuations.
A type theory has **observable effects** if there is a closed term $t : \mathbb{B}$ that is **not observationally equivalent to a value**, i.e. there is a context $C[\cdot]$ s.t.

\[
C[\text{true}] \equiv \text{true} \quad \text{and} \quad C[\text{false}] \equiv \text{true} \quad \text{but} \quad C[t] \equiv \text{false}
\]

This happens for many kind of effects, including continuations.

Such terms are typically called **non-standard** booleans.

e.g. a function $\text{is\_empty} : \Pi A. A \to \mathbb{B}$ deciding inhabitation of a type.
A Tension Build-up

Definition

A type theory enjoys *substitution* if the following rule is derivable.

\[
\frac{
\Gamma, x : X \vdash \bullet : A \quad \Gamma \vdash t : X
}{
\Gamma \vdash \bullet : A\{x := t\}
}\]
Definition

A type theory enjoys substitution if the following rule is derivable.

\[
\frac{\Gamma, x : X \vdash \bullet : A \quad \Gamma \vdash t : X}{\Gamma \vdash \bullet : A\{x := t\}}
\]

Substitution is usually taken for granted

... hint: this is a bias
Definition

A type theory enjoys \textit{dependent elimination} on booleans if we have:

\[
\Gamma, b : \mathbb{B} \vdash P : \Box \\
\Gamma \vdash \bullet : P\{b := \text{true}\} \\
\Gamma \vdash \bullet : P\{b := \text{false}\} \\
\Gamma, b : \mathbb{B} \vdash \bullet : P
\]
Definition

A type theory enjoys dependent elimination on booleans if we have:

\[
\Gamma, b : \mathbb{B} \vdash P : \square \\
\Gamma \vdash \bullet : P\{b := \text{true}\} \\
\Gamma \vdash \bullet : P\{b := \text{false}\} \\
\Gamma, b : \mathbb{B} \vdash \bullet : P
\]

The landmark of dependent type theory, used to encode induction!

Absence of dependent elimination smells of trivial theories.
Sounds like desirable features, right?
Sounds like desirable features, right?

Theorem (Explosive Ménage à Trois a.k.a. Fire Triangle)

\[ \text{Effects} + \text{substitution} + \text{dep. elimination} \vdash \text{logically inconsistent}. \]
The proof is actually straightforward.

Proof.

If $C$ distinguishes boolean values from an effectful term $M$, prove by dependent elimination $\Pi(b : \mathbb{B})$. $C[b] = \text{false}$, apply to $M$ and derive $\text{true} = \text{false}$.
The proof is actually straightforward.

Proof.
If $C$ distinguishes boolean values from an effectful term $M$, prove by dependent elimination $\Pi(b : \mathbb{B}). C[b] = \text{false}$, apply to $M$ and derive $\text{true} = \text{false}$.

We essentially retrofitted the definition of effects to make it work.
There Is No Free Lunch

The proof is actually straightforward.

Proof.

If $C$ distinguishes boolean values from an effectful term $M$, prove by dependent elimination $\Pi(b : \mathbb{B}). C[b] = \text{false}$, apply to $M$ and derive $\text{true} = \text{false}$.

We essentially retrofitted the definition of effects to make it work.

‘But most effects are also observables effects!

So it’s not cheating either.
There Is No Free Lunch

The proof is actually straightforward.

Proof.
If $C$ distinguishes boolean values from an effectful term $M$, prove by dependent elimination $\Pi(b : \mathbb{B}). C[b] = \text{false}$, apply to $M$ and derive $\text{true} = \text{false}$.

We essentially retrofitted the definition of effects to make it work.

‘But most effects are also observables effects!

So it’s not cheating either.

And now for a high-level overview of the problem and solutions
Dependency entails one major difference with usual type systems.
Dependency entails one major difference with usual type systems.

Meet conversion:

\[ A \equiv_{\beta} B \quad \Gamma \vdash M : B \]

\[ \Gamma \vdash M : A \]
Dependency entails one major difference with usual type systems.

Meet conversion:

\[
A \equiv_{\beta} B \quad \Gamma \vdash M : B \\
\frac{}{\Gamma \vdash M : A}
\]

Bad news 1

Typing rules embed the dynamics of programs!
Dependency entails one major difference with usual type systems.

Meet conversion:

\[
A \equiv_{\beta} B \quad \Gamma \vdash M : B
\]

\[
\Gamma \vdash M : A
\]

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.**
Call-by-name vs. Call-by-value

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

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

In call-by-name + effects:
\[(\lambda x. M) N \equiv M \{ x := N \} \Rightarrow \text{arbitrary substitution} \]

\[(\lambda b : \text{bool}. M) \text{fail} \Rightarrow \text{non-standard booleans} \]

Substitution is a feature of call-by-name

In call-by-value + effects:
\[(\lambda x. M) V \equiv M \{ x := V \} \Rightarrow \text{substitute only values} \]

\[(\lambda b : \text{B}. M) N \equiv (\lambda b : \text{B}. M) V \Rightarrow \text{boolean values are booleans} \]

Dependent elimination is a feature of call-by-value

Pédrot & Tabareau (INRIA)
Call-by-name vs. Call-by-value

Call-by-name: \textbf{functions} well-behaved vs. \textbf{inductives} ill-behaved

Call-by-value: \textbf{inductives} well-behaved vs. \textbf{functions} ill-behaved
Call-by-name vs. Call-by-value

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:

\[(\lambda x. M) \; N \equiv M\{x := N\} \leadsto \text{arbitrary substitution}\]

\[(\lambda b : \text{bool}. \; M) \; \text{fail} \leadsto \text{non-standard booleans}\]

Substitution is a feature of call-by-name
Reduction in an Effectful World

**Call-by-name vs. Call-by-value**

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:

\[(\lambda x. M) \; N \equiv M\{x := N\} \quad \leadsto \quad \text{arbitrary substitution}\]

\[(\lambda b : \text{bool}. \; M) \; \text{fail} \quad \leadsto \quad \text{non-standard booleans}\]

**Substitution is a feature of call-by-name**

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

\[(\lambda x. M) \; V \equiv M\{x := V\} \quad \leadsto \quad \text{substitute only values}\]

\[(\lambda b : \mathbb{B}. \; M) \; N \equiv (\lambda b : \mathbb{B}. \; M) \; V \quad \leadsto \quad \text{boolean values are booleans}\]

**Dependent elimination is a feature of call-by-value**
Impossible is not French

**Three knobs ⇒ Four solutions**

This is good ol' CIC, Keep Calm and Carry on.

▷ Go CBN and restrict dependent elimination: Baclofen

if $M$ then $P_1$ else $P_2$:

▷ CBV rules, respect values, and dump substitution: one weird trick

The least conservative approach

▷ Who cares about consistency? I want all!

A paradigm shift: from type theory to dependent languages, e.g. ExTT

Pédrot & Tabareau (INRIA)
Impossible is not French

Three knobs ⇒ Four solutions

▷ **Down with effects**: CBN and CBV reconcile

This is good ol’ CIC, **Keep Calm and Carry on**.
Impossible is not French

Three knobs ⇒ Four solutions

▷ Down with effects: CBN and CBV reconcile

This is good ol’ CIC, Keep Calm and Carry on.

▷ Go CBN and restrict dependent elimination: Baclofen Type Theory

\[
\text{if } M \text{ then } N_1 \text{ else } N_2 : \text{if } M \text{ then } P_1 \text{ else } P_2
\]
Impossible is not French

Three knobs ⇒ Four solutions

- **Down with effects**: CBN and CBV reconcile

  This is good ol’ CIC, *Keep Calm and Carry on.*

- **Go CBN** and restrict dependent elimination: Baclofen Type Theory

  if $M$ then $N_1$ else $N_2$: if $M$ then $P_1$ else $P_2$

- **CBV rules**, respect values, and dump substitution: one weird trick

  The least conservative approach
Impossible is not French

Three knobs ⇒ Four solutions

- **Down with effects**: CBN and CBV reconcile
  
  This is good ol’ CIC, Keep Calm and Carry on.

- **Go CBN** and restrict dependent elimination: Baclofen Type Theory

  \[
  \text{if } M \text{ then } N_1 \text{ else } N_2 : \text{if } M \text{ then } P_1 \text{ else } P_2
  \]

- **CBV rules**, respect values, and dump substitution: one weird trick

  The least conservative approach

- Who cares about consistency? **I want all!**

A paradigm shift: from type theory to dependent languages, e.g. ExTT
A Generic Workaround

We have a proposal for a generalization of CBPV to factor both.

\[ \partial \text{CBPV} \] (We had to pick a fancy name.)

The main novelties: two for the price of one

• Not one, but two parallel hierarchies of universes: \( \Box v \) vs. \( \Box c \)

• Not one, but two let-bindings!

\[ \Gamma \vdash t : F A \]

\[ \Gamma \vdash X : \Box c \]

\[ \Gamma, x : A \vdash u : X \]

\[ \Gamma \vdash \text{let } x := t \text{ in } u : X \]

\[ \Gamma \vdash d \text{let } x := t \text{ in } u : \text{let } x := t \text{ in } X \]

• Justified by all of our syntactic models so far (and we have quite a few)
A Generic Workaround

We have a proposal for a generalization of CBPV to factor both.

\[ \partial \text{CBPV} \]

(We had to pick a fancy name.)
A Generic Workaround

We have a proposal for a generalization of CBPV to factor both.

\[\partial\text{CBPV}\]

(We had to pick a fancy name.)

The main novelties: two for the price of one

- Not one, but **two** parallel hierarchies of universes: \(\Box_v\) vs. \(\Box_c\)!
- Not one, but **two** let-bindings!

\[
\begin{align*}
\Gamma \vdash t : F A & \quad \Gamma \vdash X : \Box_c & \quad \Gamma, x : A \vdash u : X \\
\Gamma \vdash \text{let } x := t \text{ in } u : X \\
\Gamma \vdash t : F A & \quad \Gamma, x : A \vdash X : \Box_c & \quad \Gamma, x : A \vdash u : X \\
\Gamma \vdash \text{dlet } x := t \text{ in } u : \text{let } x := t \text{ in } X
\end{align*}
\]

- Justified by all of our syntactic models so far (and we have quite a few)
Many More

This was a very high-level talk

Many things I did not discuss here!

- A good notion of purity: thunkability vs. linearity
- Complex $\partial$CBPV encodings
- Presheaves as observationally pure terms of an impure CBV theory

http://pédrot.fr/articles/dcbpv.pdf
Conclusion

- Effects and dependent types: choose your side.
  - Purity, CBN, CBV, Michael Bay?
- Even inconsistent theories have practical interest.
Thanks for your attention.