The Fire Triangle
How to Mix Substitution, Dependent Elimination and Effects

Pierre-Marie Pédrot, Nicolas Tabareau

Gallinette (INRIA)

POPL’20
January, 23th 2020
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
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
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
Yet CIC suffers from a **fundamental** flaw.
A CIC Joke

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

Intuitionistic Logic ⇔ 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!
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**!
Intuitionistic Logic ⇔ 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**!
Intuitionistic Logic ⇔ Functional Programming
Intuitionistic Logic $\iff$ Functional Programming

Thus, the same problem for mathematically inclined users.
Intuitionistic Logic $\Leftrightarrow$ 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?
Non-Intuitionistic Logic ⇔ Impure Programming
Non-Intuitionistic Logic ↔ Impure Programming

We want a type theory with effects!
Non-Intuitionistic Logic ⇔ Impure Programming

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!
Something is Rotten in the State of Type Theory

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

Our specialty:
Something is Rotten in the State of Type Theory

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

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

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
Why do we have trouble mixing effects and dependent types?
Why do we have trouble mixing effects and dependent types?

Coincidence? I Think Not!
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 **dependent elimination** on booleans if we have:

$$\Gamma, b : B \vdash P : □ \Gamma \vdash • : P\{b := true\} \quad \Gamma \vdash • : P\{b := false\}$$

Definition

A type theory has **observable effects** if there is a closed term $t : B$ that is not observationally equivalent to a value, i.e. there is a context $C\[·\]$ s.t. $C\[true\] ≡ true$ and $C\[false\] ≡ true$ but $C\[t\] ≡ false$.
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 **dependent elimination** on booleans if we have:

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

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

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

Definition

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

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

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.

\[
C[\text{true}] \equiv \text{true} \quad \text{and} \quad C[\text{false}] \equiv \text{true} \quad \text{but} \quad C[t] \equiv \text{false}
\]
Sounds like desirable properties, right?
Type Theory on Fire

Sounds like desirable properties, right?

Theorem (Fire Triangle)

\[ \text{substitution} \ + \ \text{dep. elimination} \ + \ \text{effects} \vdash \text{logically inconsistent}. \]
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 $\text{true} = \text{false}$. 

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.
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.
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
Dependent types entail one major difference with simpler type systems.
Dependent types entail one major difference with simpler type systems.

\[
\begin{align*}
  A \equiv_{\beta} B & \quad \Gamma \vdash M : B \\
  \hline
  \Gamma \vdash M : A
\end{align*}
\]
Dependent types entail one major difference with simpler type systems.

\[
\begin{align*}
A & \equiv_{\beta} B & \Gamma \vdash M : B \\
\Gamma \vdash M : A
\end{align*}
\]

Bad news 1

Typing rules embed the dynamics of programs!
Dependent types entail one major difference with simpler type systems.

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

Bad news 1

Typing rules embed the dynamics of programs!

Bad news 2

Effects make reduction strategies relevant.
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

- Substitution is a feature of call-by-name
- Dependent elimination is a feature of call-by-value
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
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

Substitution is a feature of call-by-name

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

Three knobs $\Rightarrow$ Four solutions
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

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

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

- **CBV rules**, respect values, and dump substitution

  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

  The least conservative approach

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

A paradigm shift: from type theory to dependent languages, e.g. ExTT
Assuming you want consistent dependent effects...

Call-by-name vs. Call-by-value
Pick Your Side, Comrade

Assuming you want consistent dependent effects...

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

CBPV
Assuming you want consistent dependent effects...

Call-by-name and Call-by-value

(We had to pick a fancy name, everything else already taken.)
\( \partial \text{CBPV} \)

Justified by all of our syntactic models so far

And we have quite a few!

- Impure Forcing — Unnatural Presheaves
- Reader
- Exceptions — Free algebraic effects
- Self-algebraic monads
- ...
\( \partial \text{CBPV} \)

Justified by all of our syntactic models so far

And we have quite a few!
- Impure Forcing — Unnatural Presheaves
- Reader
- Exceptions — Free algebraic effects
- Self-algebraic monads
- ... ← notice the lack of CPS here
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!

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

\[
\begin{align*}
\Gamma \vdash t : F A & \quad \Gamma, x : A \vdash X : \Box_c & \quad \Gamma, x : A \vdash u : X \\
\Gamma \vdash \text{dlet } x := t \text{ in } u : \text{let } x := t \text{ in } X
\end{align*}
\]
The main novelties: two for the price of one

- Not one, but **two** parallel hierarchies of universes: $\square_v$ vs. $\square_c$!
- Not one, but **two** let-bindings!

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

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

See the paper for more details
Much 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
- Explicit model constructions
- A new look on presheaves
Conclusion

What we did

- Effects and dependent types: you can't have your cake and eat it.
 正面 Purity, CBN, CBV, Michael Bay?
- Even inconsistent theories have practical interest.
- $\partial$CBPV a unifying framework for dependent effects
Conclusion

What we did

- Effects and dependent types: you can't have your cake and eat it.
  - Purity, CBN, CBV, Michael Bay?
- Even inconsistent theories have practical interest.
- $\partial$CBPV a unifying framework for dependent effects

What we should probably do do

- Study more in details CBV type theories
- Try to give a model for classical logic, choice, what else?
- Implement $\partial$CBPV?