An Effectful Way to Eliminate Addiction to Dependence

Pierre-Marie Pédrot[1] Nicolas Tabareau[2]

[1]University of Ljubljana, [2]INRIA

Gallium Seminar
4th May 2017

The Most Important Issue of Them All

Let's start this talk by a **fundamental** flaw of type theory.

The Most Important Issue of Them All

Let's start this talk by a **fundamental** flaw of type theory.

- Assume 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

Let's start this talk by a **fundamental** flaw of type theory.

- Assume 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 proof-as-program correspondence,

Intuitionistic Logic $\Leftrightarrow$ Functional Programming

A Well-known Limitation

This is pretty much standard. By proof-as-program correspondence,

Intuitionistic Logic $\Leftrightarrow$ Functional Programming

which means **no effects** in TT, amongst which:

- no exceptions
- no state
- no non-termination
- no printing

A Well-known Limitation

This is pretty much standard. By proof-as-program correspondence,

Intuitionistic Logic $\Leftrightarrow$ Functional Programming

which means **no effects** in TT, amongst which:

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

On Burritos

In less expressive settings, a few workarounds are known.

Typically, on the programming side, use the **monadic** style.

- A type $T : \square \to \square$
- A combinator $\mathtt{return} : \alpha \to T\,\alpha$
- A combinator $\mathtt{bind} : T\,\alpha \to (\alpha \to T\,\beta) \to T\,\beta$
- A few equations

On Burritos

In less expressive settings, a few workarounds are known.

Typically, on the programming side, use the **monadic** style.

- A type $T : \square \to \square$
- A combinator $\texttt{return} : \alpha \to T\ \alpha$
- A combinator $\texttt{bind} : T\ \alpha \to (\alpha \to T\ \beta) \to T\ \beta$
- A few equations

Interpret mechanically effectful programs using this (see Moggi).

This is pervasive in e.g. Haskell.

Less is More

On the logic side, take the issue the other way around.

Less is More

On the logic side, take the issue the other way around.

Effects are known to implement non-intuitionistic axioms!

- `callcc` $\sim$ classical logic (Griffin '90)
- exceptions $\sim$ Markov's rule (Friedman's trick)
- global monotonous cell $\sim$ $\neg\mathrm{CH}$ (forcing)
- delimited continuations $\sim$ double negation shift
- ...

Achieve this using logical translations, e.g. double-negation.

Alternative Facts

We want a type theory with effects!

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

Alternative Facts

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.

The Expressivity Wall

Problem is:

Programming and logical techniques do not scale to type theory.

The Expressivity Wall

Problem is:

Programming and logical techniques do not scale to type theory.

- Monads do not aknowledge dependence

$$\texttt{bind} : T\,\alpha \to (\alpha \to T\,\beta) \to T\,\beta$$

$$\texttt{dbind} : \Pi\hat{x} : T\,\alpha.\,(\Pi x : \alpha.\,T\,(\beta\ x)) \to T\,(\beta\ ?)$$

- They don't aknowledge types-as-terms either
- And they don't preserve the computational rules of TT

The Expressivity Wall

Problem is:

Programming and logical techniques do not scale to type theory.

- Monads do not aknowledge dependence

$$\mathtt{bind} : T\,\alpha \to (\alpha \to T\,\beta) \to T\,\beta$$

$$\mathtt{dbind} : \Pi \hat{x} : T\,\alpha.\,(\Pi x : \alpha.\,T\,(\beta\ x)) \to T\,(\beta\ ?)$$

- They don't aknowledge types-as-terms either
- And they don't preserve the computational rules of TT

On the other hand:

- Herbelin showed that CIC + `callcc` is unsound!

In This Talk

1. Adding a vast range of effects to (almost) full TT
    - reader (already done previously with the **forcing translation**)
    - writer, exceptions, non-termination, non-determinism...
    - All with the new **weaning translation**!

In This Talk

1. Adding a vast range of effects to (almost) full TT
   - reader (already done previously with the **forcing translation**)
   - writer, exceptions, non-termination, non-determinism...
   - All with the new **weaning translation**!
2. Implementing them thanks to program translations
   - No crazy category theory models!
   - So-called **syntactic models**.
   - Compile them on-the-fly into vanilla type theory!

In This Talk

1. Adding a vast range of effects to (almost) full TT
    - reader (already done previously with the **forcing translation**)
    - writer, exceptions, non-termination, non-determinism...
    - All with the new **weaning translation**!
2. Implementing them thanks to program translations
    - No crazy category theory models!
    - So-called **syntactic models**.
    - Compile them on-the-fly into vanilla type theory!
3. Introducing a generic notion of effectful dependent type theory
    - A simple, sensible restriction of dependent elimination
    - Seemingly compatible with all known effects

Syntactic Models

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Syntactic Models

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Obviously, that's subtle.

- The correctness of $[\cdot]$ lies in the meta (Darn, Gödel!)
- The translation must preserve typing (Not easy)
- In particular, it must preserve conversion (Argh!)

Syntactic Models

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Obviously, that's subtle.

- The correctness of $[\cdot]$ lies in the meta (Darn, Gödel!)
- The translation must preserve typing (Not easy)
- In particular, it must preserve conversion (Argh!)

Yet, a lot of nice consequences.

- Does not require non-type-theoretical foundations (*monism*)
- Can be implemented in your favourite proof assistant
- Easy to show (relative) consistency, look at $[\![\texttt{False}]\!]$
- Easier to understand computationally

(Mis)understanding Dependent Type Theory

There are two essential properties of TT that need to be explicited.

(Mis)understanding Dependent Type Theory

There are two essential properties of TT that need to be explicited.

#1. Type theory is call-by-name by construction.

- This is because of the unrestricted conversion rule.
- But the usual monadic interpretation is call-by-value!
- We need to rely on an alternative decomposition (based on CBPV).

(Mis)understanding Dependent Type Theory

There are two essential properties of TT that need to be explicited.

#1. Type theory is call-by-name by construction.

- This is because of the unrestricted conversion rule.
- But the usual monadic interpretation is call-by-value!
- We need to rely on an alternative decomposition (based on CBPV).

#2. Dependent elimination is hardcore intuitionistic.

- It rules out non-standard inductive terms that exist in CBN + effects
- Reminiscent of Brouwer vs. Bishop mathematics
- Needs to be weakened in presence of effects (« Bishop-style TT »)

My Name is Call, Call-by-Name

TT is intrisically call-by-name because of the conversion rule:

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

where $\equiv_\beta$ is generated by:

$$(\lambda x : A.\, M)\ N \equiv_\beta M\{x := N\}$$

My Name is Call, Call-by-Name

TT is intrisically call-by-name because of the conversion rule:

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

where $\equiv_\beta$ is generated by:

$$(\lambda x : A.\, M)\ N \equiv_\beta M\{x := N\}$$

To be call-by-value, it would require instead $\equiv_{\beta v}$ generated by:

$$(\lambda x : A.\, M)\ V \equiv_{\beta v} M\{x := V\}$$

where V is a value. But that's not TT...

Tell Me Eleinberg-Moore

Turns out it is easy to give a call-by-name monadic decomposition.

Use the Eleinberg-Moore category, i.e. the category of algebras.

Tell Me Eleinberg-Moore

Turns out it is easy to give a call-by-name monadic decomposition.

Use the Eleinberg-Moore category, i.e. the category of algebras.

For us, a T-algebra will be an inhabitant of:

$$\boxed{\square} := \Sigma A : \square .\; T\ A \rightarrow A$$

A few remarks:

- It is hard to formulate the notion of algebra without higher-order types
- We don't require any equations in $\boxed{\square}$ (they're quite not algebras)
- It turns out it is not necessary...

Required structure

We assume a monad given by universe-polymorphic terms:

$$
\begin{array}{lcl}
T & : & \square_i \to \square_i \\
\texttt{ret} & : & \Pi(A : \square).\, A \to T\ A \\
\texttt{bind} & : & \Pi(A\ B : \square).\, T\ A \to (A \to T\ B) \to T\ B
\end{array}
$$

and we require **no equations**!!

Required structure

We assume a monad given by universe-polymorphic terms:

$$\begin{array}{lcl} T & : & \square_i \to \square_i \\ \texttt{ret} & : & \Pi(A : \square).\, A \to T\ A \\ \texttt{bind} & : & \Pi(A\ B : \square).\, T\ A \to (A \to T\ B) \to T\ B \end{array}$$

and we require **no equations**!!

Furthermore, in Type Theory, types are terms. We want the monad to be **self-algebraic**. This is given by:

$$\begin{array}{lcl} \texttt{El} & : & T\ \square_i \to \square_i \\ \texttt{El}\ (\texttt{ret}\ \square\ M) & \equiv_\beta & M \end{array}$$

A lot of monads appear to be self-algebraic.

The Weaning Translation of the Negative Fragment

$$
\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & \lambda x : [\![A]\!].\, [M] \\
[M\ N] & := & [M]\ [N] \\
[\Box_i] & := & \texttt{ret}\ \Box_{i+1}\ (T\ \Box_i, \mu_\Box) \\
[\Pi x : A.\, B] & := & \texttt{ret}\ \Box\ (\Pi x : [\![A]\!].\, [\![B]\!], \mu_\Pi) \\
[\![A]\!] & := & (\texttt{El}\ [A]).\pi_1 \\
\mu_\Box & : & T\,(T\,\Box) \to \Box \\
\mu_\Pi & : & T\,(\Pi x : [\![A]\!].\, [\![B]\!]) \to \Pi x : [\![A]\!].\, [\![B]\!]
\end{array}
$$

The Weaning Translation of the Negative Fragment

$$
\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & \lambda x : [\![A]\!].\, [M] \\
[M\ N] & := & [M]\ [N] \\
[\square_i] & := & \mathtt{ret}\ \square_{i+1}\ (T\ \square_i, \mu_\square) \\
[\Pi x : A.\, B] & := & \mathtt{ret}\ \square\ (\Pi x : [\![A]\!].\, [\![B]\!], \mu_\Pi) \\
[\![A]\!] & := & (\mathtt{El}\ [A]).\pi_1 \\
\mu_\square & : & T\ (T\ \square) \to \square \\
\mu_\Pi & : & T\ (\Pi x : [\![A]\!].\, [\![B]\!]) \to \Pi x : [\![A]\!].\, [\![B]\!]
\end{array}
$$

- Functional fragment untouched, types mangled into algebras
- $[\![\square]\!] \equiv_\beta T\ \square$ and $[\![\Pi x : A.\, B]\!] \equiv_\beta \Pi x : [\![A]\!].\, [\![B]\!]$

The Weaning Translation of the Negative Fragment

$$
\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & \lambda x : [\![A]\!].\, [M] \\
[M\ N] & := & [M]\ [N] \\
[\square_i] & := & \texttt{ret}\ \square_{i+1}\ (T\ \square_i, \mu_\square) \\
[\Pi x : A.\, B] & := & \texttt{ret}\ \square\ (\Pi x : [\![A]\!].\, [\![B]\!], \mu_\Pi) \\
[\![A]\!] & := & (\texttt{El}\ [A]).\pi_1 \\
\mu_\square & : & T\ (T\ \square) \to \square \\
\mu_\Pi & : & T\ (\Pi x : [\![A]\!].\, [\![B]\!]) \to \Pi x : [\![A]\!].\, [\![B]\!]
\end{array}
$$

- Functional fragment untouched, types mangled into algebras
- $[\![\square]\!] \equiv_\beta T\ \square$ and $[\![\Pi x : A.\, B]\!] \equiv_\beta \Pi x : [\![A]\!].\, [\![B]\!]$

Soundness

If $\Gamma \vdash M : A$ then $[\![\Gamma]\!] \vdash [M] : [\![A]\!]$. (In particular, conversion is preserved.)

Reduction vs. Effects

Nothing fancy in the negative fragment, by the well-known duality.

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

Reduction vs. Effects

Nothing fancy in the negative fragment, by the well-known duality.

- 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, consider:

$$(\lambda b : \texttt{bool}.\, M)\ \mathbf{fail} \quad \rightsquigarrow \quad \text{non-standard inductive terms}$$

In call-by-value + effects, consider:

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

Weaning Inductive Types

For the sake of explanation, let's focus on a very simple type:

$$\texttt{Inductive bool} := \texttt{true} \mid \texttt{false}.$$

We pose:

$$\begin{array}{lcl}
[\texttt{bool}] & := & \texttt{ret}\ \Box\ (T\,\texttt{bool}, \mu_{\texttt{bool}}) \\
[\texttt{true}] & := & \texttt{ret bool true} \\
[\texttt{false}] & := & \texttt{ret bool false} \\
\mu_{\texttt{bool}} & : & T\,(T\,\texttt{bool}) \to T\,\texttt{bool}
\end{array}$$

Weaning Inductive Types

For the sake of explanation, let's focus on a very simple type:

$$\texttt{Inductive bool} := \texttt{true} \mid \texttt{false}.$$

We pose:

$$\begin{array}{lcl} [\texttt{bool}] & := & \texttt{ret}\ \square\ (T\ \texttt{bool}, \mu_{\texttt{bool}}) \\ [\texttt{true}] & := & \texttt{ret bool true} \\ [\texttt{false}] & := & \texttt{ret bool false} \\ \mu_{\texttt{bool}} & : & T\ (T\ \texttt{bool}) \to T\ \texttt{bool} \end{array}$$

Remark that $[\![\texttt{bool}]\!] \equiv_\beta T\ \texttt{bool}$.

Soundness

If $\Gamma \vdash M : A$ then $[\![\Gamma]\!] \vdash [M] : [\![A]\!]$.

E-LI-MI-NATE!

We need a bit more structure on T to implement elimination:

$$\begin{array}{lcl}
\texttt{hbind} & : & \Pi(A:\square)(B:T\,\boxed{\square}).\,T\,A \to (A \to [\![B]\!]) \to [\![B]\!] \\
\texttt{dbind} & : & \Pi(A:\square)(B:A \to T\,\boxed{\square}).\,\Pi(\hat{x}:T\,A). \\
 & & \quad (\Pi(x:A).\,[\![B\,x]\!]) \to (\texttt{El}\ (\texttt{hbind}\ A\ [\square]\ \hat{x}\ B)).\pi_1
\end{array}$$

subject to:

$$\begin{array}{lcl}
\texttt{hbind}\ A\ B\,(\texttt{ret}\ A\ M)\ F & \equiv_\beta & F\ M \\
\texttt{dbind}\ A\ B\,(\texttt{ret}\ A\ M)\ F & \equiv_\beta & F\ M
\end{array}$$

Essentially, `hbind` and `dbind` are variants of `bind`.

E-LI-MI-NATE!

We need a bit more structure on T to implement elimination:

$$
\begin{array}{lcl}
\texttt{hbind} & : & \Pi(A:\Box)(B:T\ \boxed{\Box}).\ T\ A \to (A \to [\![B]\!]) \to [\![B]\!] \\
\texttt{dbind} & : & \Pi(A:\Box)(B:A \to T\ \boxed{\Box}).\ \Pi(\hat{x}:T\ A). \\
 & & \quad (\Pi(x:A).\ [\![B\ x]\!]) \to (\texttt{El}\ (\texttt{hbind}\ A\ [\Box]\ \hat{x}\ B)).\pi_1
\end{array}
$$

subject to:

$$
\begin{array}{lcl}
\texttt{hbind}\ A\ B\ (\texttt{ret}\ A\ M)\ F & \equiv_\beta & F\ M \\
\texttt{dbind}\ A\ B\ (\texttt{ret}\ A\ M)\ F & \equiv_\beta & F\ M
\end{array}
$$

Essentially, `hbind` and `dbind` are variants of `bind`.

Remark that the second equation is well-typed iff the first holds.

Interpreting Non-Dependent Elimination

It is easy to provide a non-dependent eliminator using hbind:

$$
\begin{array}{rcl}
[\texttt{bool\_case}] & : & [\![\Pi P : \square.\, P \to P \to \texttt{bool} \to P]\!] \\
 & := & \lambda(P : T\,\boxed{\square})\,(p_t\,p_f : [\![P]\!])\,(\hat{b} : T\,\texttt{bool}). \\
 & & \quad \texttt{hbind bool}\ P\ \hat{b}\ (\lambda b.\, \texttt{if}\ b\ \texttt{then}\ p_t\ \texttt{else}\ p_f)
\end{array}
$$

which has the right reduction rules:

$$
\begin{array}{rcl}
[\texttt{bool\_case}\ P\ p_t\ p_f\ \texttt{true}] & \equiv_\beta & p_t \\
[\texttt{bool\_case}\ P\ p_t\ p_f\ \texttt{false}] & \equiv_\beta & p_f
\end{array}
$$

Remember:

$$
\begin{array}{l}
\texttt{hbind} : \Pi(A : \square)(B : T\,\boxed{\square}).\, T\,A \to (A \to [\![B]\!]) \to [\![B]\!] \\
\texttt{hbind}\ A\ B\ (\texttt{ret}\ A\ M)\ F \equiv_\beta F\ M
\end{array}
$$

Eliminating Addiction to Dependence

We would like to recover dependent elimination...

Eliminating Addiction to Dependence

We would like to recover dependent elimination...

... but it's not valid anymore in presence of effects!

As $[\![\texttt{bool}]\!] \equiv_\beta T\,\texttt{bool}$, if T is not the identity then there are closed booleans in the translation which are neither $[\texttt{true}]$ nor $[\texttt{false}]$.

Eliminating Addiction to Dependence

We would like to recover dependent elimination...

... but it's not valid anymore in presence of effects!

As $[\![\texttt{bool}]\!] \equiv_\beta T\,\texttt{bool}$, if T is not the identity then there are closed booleans in the translation which are neither $[\texttt{true}]$ nor $[\texttt{false}]$.

- Typical of CBN + effects: recall Herbelin's paradox
- Already arose in our forcing translation
- We need to restrict dependent elimination the same way!

Eliminating Addiction to Dependence II

The trick consists in sprinkling a few storage operators. For `bool`:

$$\begin{array}{lcl} [\theta_{\texttt{bool}}] & : & [\![\texttt{bool} \rightarrow (\texttt{bool} \rightarrow \square) \rightarrow \square]\!] \\ & := & [\lambda b.\, \texttt{bool\_case}\ (\texttt{bool} \rightarrow \square)\ (\lambda k.\, k\, \texttt{true})\ (\lambda k.\, k\, \texttt{false})\ b] \end{array}$$

- Only defined in the source via non-dependent eliminator
- In particular, agnostic to the actual translation
- CPS-like to enforce CBV in a CBN world
- Trivial in CIC: $\vdash \Pi b : \texttt{bool}.\, \theta_{\texttt{bool}}\ b\ P = P\ b$

Eliminating Addiction to Dependence II

The trick consists in sprinkling a few storage operators. For `bool`:

$$\begin{array}{lcl} [\theta_{\texttt{bool}}] & : & [\![\texttt{bool} \to (\texttt{bool} \to \square) \to \square]\!] \\ & := & [\lambda b.\, \texttt{bool\_case}\ (\texttt{bool} \to \square)\ (\lambda k.\, k\, \texttt{true})\ (\lambda k.\, k\, \texttt{false})\ b] \end{array}$$

- Only defined in the source via non-dependent eliminator
- In particular, agnostic to the actual translation
- CPS-like to enforce CBV in a CBN world
- Trivial in CIC: $\vdash \Pi b : \texttt{bool}.\, \theta_{\texttt{bool}}\ b\ P = P\ b$

Using `dbind`, this allows to implement:

$$[\texttt{bool\_rect}] : [\![\Pi P : \texttt{bool} \to \square.\, P\, \texttt{true} \to P\, \texttt{false} \to \Pi b : \texttt{bool}.\, \theta_{\texttt{bool}}\ b\ P]\!]$$

with the expected reduction rules.

Weaning Everywhere

There are a lot of monads that satisfy the weaning conditions.

- Exception monad $T\ A := A + E$
- Non-determinism $T\ A := A \times \texttt{list}\ A$
- Non-termination $T\ A := \nu X. A + X$
- Writer $T\ A := A \times \texttt{list}\ \Omega$ (the one we need for HELLO WORLD)

Note that some lead to a logically inconsistent model.

Weaning Everywhere

There are a lot of monads that satisfy the weaning conditions.

- Exception monad $T\ A := A + E$
- Non-determinism $T\ A := A \times \texttt{list}\ A$
- Non-termination $T\ A := \nu X. A + X$
- Writer $T\ A := A \times \texttt{list}\ \Omega$ (the one we need for HELLO WORLD)

Note that some lead to a logically inconsistent model.

A few monads aren't self-algebraic, e.g. state, reader and continuation.

Logic, at Last

In some inconsistent cases, full dependent elimination is valid.
Most notably, this is the case for the exception monad.

Let's use that to do a Friedman A-translation on steroids!

Logic, at Last

In some inconsistent cases, full dependent elimination is valid.
Most notably, this is the case for the exception monad.

Let's use that to do a Friedman A-translation on steroids!

Lemmatas

With the exception monad $T\ A := A + E$:

- Full dependent elimination is valid (at the expense of consistency)
- We have $[\![\neg\neg A]\!] \cong ([\![A]\!] \rightarrow E) \rightarrow E$
- If A is a first-order type, then $[\![A]\!] \rightarrow A + E$.

In some inconsistent cases, full dependent elimination is valid.
Most notably, this is the case for the exception monad.

Let's use that to do a Friedman A-translation on steroids!

Lemmatas

With the exception monad $T\ A := A + E$:

- Full dependent elimination is valid (at the expense of consistency)
- We have $[\![\neg\neg A]\!] \cong ([\![A]\!] \to E) \to E$
- If A is a first-order type, then $[\![A]\!] \to A + E$.

Admissibility of Markov's rule in CIC

If A is first-order and $\vdash_{\mathrm{CIC}} \neg\neg A$ then $\vdash_{\mathrm{CIC}} A$.

Back to restricted elimination. It turns out we have a semantic criterion for valid dependent predicates.

LINEARITY.

Back to restricted elimination. It turns out we have a semantic criterion for valid dependent predicates.

LINEARITY.

- A concept invented by G. Munch, rephrased recently by P. Levy.
- Little to do with « linear use of variables »
- Essentially, $f : A \to B$ linear in CBN if semantically CBV in A.
- Categorically, f linear iff it is an algebra morphism.
- Storage operators turn freely any morphism into a linear one.
- Can be approximated by a syntactic guard condition.

$$\frac{\Gamma \vdash M : \texttt{bool} \qquad \ldots \qquad P \text{ linear in } b}{\Gamma \vdash \texttt{if } M \texttt{ return } \lambda b.\, P \texttt{ then } N_1 \texttt{ else } N_2 : P\{b := M\}}$$

A Bishop-style Type Theory

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

- Subset of CIC
- Independent from the actual translation.
- Works with forcing
- Works with weaning
- Prevents Herbelin's paradox

A Bishop-style Type Theory

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

- Subset of CIC
- Independent from the actual translation.
- Works with forcing
- Works with weaning
- Prevents Herbelin's paradox

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

(Take that, Brouwerian HoTT!)

Implementation

A nice paper summarizing this talk.

https://www.pédrot.fr/articles/weaning.pdf

Just as for the forcing translation we have a Coq plugin for weaning.

https://github.com/CoqHott/coq-effects

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

(If time permits, small demo here.)

Conclusion

- A new effectful translation of TT, the weaning translation
  - Cosmic version of Eilenberg-Moore categories
  - Gives both programming and logical features
- An experimentally confirmed notion of effectful type theories, BTT
  - Works for forcing, weaning and CPS
  - Restriction of dependent elimination on linearity guard condition
  - Conjecture: the correct way to add effects to TT
- Implementation of a plugin in Coq
  - Try it out today!

Thanks for your attention.