A Parametric CPS to Sprinkle CIC with Classical Reasoning

Pierre-Marie Pédrot

University of Ljubljana

LOLA 2017
19th June 2017

CIC, I'm loving it

Dependent Type Theory is awesome!

CIC, I'm loving it

Dependent Type Theory is awesome!

The pinacle of the Curry-Howard correspondence:

- You can program with it

  "A pure functional programming with crazily precise types."

- You can prove with it

  "A incredibly rich constructive logic with built-in computation."

CIC, I'm loving it

Dependent Type Theory is awesome!

The pinacle of the Curry-Howard correspondence:

- You can program with it

  "A pure functional programming with crazily precise types."
- You can prove with it

  "A incredibly rich constructive logic with built-in computation."
- Everything at the same time!

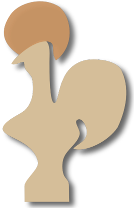
  "Prove your programs! Program your proofs!"

An effective object

That's just not theoretical ramble.

An effective object

That's just not theoretical ramble.



Lots of actual, serious, big developments.

- CompCert, VST, RustBelt...
- Four Colour Theorem, Feit-Thompson...

A Classical Problem

In practice, many people reason in the dreaded classical logic.

$$\texttt{em} : \Pi(A : \square).\, A \vee \neg A$$

A Classical Problem

In practice, many people reason in the dreaded classical logic.

$$\texttt{em} : \Pi(A : \square).\, A \vee \neg A$$

Both a theoretical and practical limitation!

- CIC is deadcore intuitionistic
- Requires that you write your statements in the right way
- Most non-logicians don't care about this fuss (both CS and math...)

A Classical Problem

In practice, many people reason in the dreaded classical logic.

$$\texttt{em} : \Pi(A : \Box).\, A \vee \neg A$$

Both a theoretical and practical limitation!

- CIC is deadcore intuitionistic
- Requires that you write your statements in the right way
- Most non-logicians don't care about this fuss (both CS and math...)

It would be nice to have a $\mathfrak{classical}$ type theory...

Attempt 1: The Truth is Out There

There is a very simple straightforward solution.

Attempt 1: The Truth is Out There

There is a very simple straightforward solution.

$$\texttt{Axiom classical : forall (A : Type), A} \lor \neg\texttt{A.}$$

Pro: Simple, local, works in Coq, be my guest.

Attempt 1: The Truth is Out There

There is a very simple straightforward solution.

```
Axiom classical : forall (A : Type), A ∨ ¬A.
```

Pro: Simple, local, works in Coq, be my guest.

Cons:

- Axioms are dangerous, you have to show consistency externally

  Classical logic holds in the well-known **Set** *model, blah-blah...*
- Non-trivial interactions: e.g. classical CIC implies proof-irrelevance.

  Classical logic is incompatible with univalence! (Your mileage may vary.)
- The logic does not compute anymore, axioms block reduction...

Attempt 2: CIC and `call/cc` are in a boat

Since Griffin, it's folklore that control operators implement classical logic.

$$\texttt{callcc} : ((A \rightarrow B) \rightarrow A) \rightarrow A$$

Attempt 2: CIC and `call/cc` are in a boat

Since Griffin, it's folklore that control operators implement classical logic.

$$\texttt{callcc} : ((A \to B) \to A) \to A$$

Essentially allows to reify context evaluation.

$$E[\texttt{callcc}\ M] \equiv_\beta \texttt{callcc}\ (\lambda k.\, E[M\ (E \circ k)])$$

The type of `callcc` is Peirce's law, the minimal logic equivalement of EM.

Attempt 2: CIC and `call/cc` are in a boat

"Just" throw `call/cc` into CIC!

Attempt 2: CIC and `call/cc` are in a boat

“Just” throw `call/cc` into CIC!

Pro: Computational by construction.

Cons:

- Needs a whole new proof assistant implementation.

  Reminder: Coq is a 33-year old project.

- Changes the global meaning of logical connectives.

  What does $\Sigma x : A.\, B$ means?

Attempt 2: CIC and `call/cc` are in a boat

“Just” throw `call/cc` into CIC!

Pro: Computational by construction.

Cons:

- Needs a whole new proof assistant implementation.

  Reminder: Coq is a 33-year old project.

- Changes the global meaning of logical connectives.

  What does $\Sigma x : A.\, B$ means?

- … and it changes it so much that it also proves `False`!!!

  Pro: At least my proofs are going to be easier.

Attempt 2: CIC fell into the water!

Herbelin showed a paradox in CIC + `callcc`, boiling down to:

Dependent elimination + Proof-relevance + `callcc` = TROUBLE.

Attempt 2: CIC fell into the water!

Herbelin showed a paradox in CIC + `callcc`, boiling down to:

Dependent elimination + Proof-relevance + `callcc` = TROUBLE.

Essentially:

- `callcc` allows to build booleans that are neither `true` nor `false`

$$b := \texttt{if em CIC\_consistency then true else false}$$

- Dependent elimination is oblivious of this fact

$$\Pi P : \mathbb{B} \to \square.\, P\ \texttt{true} \to P\ \texttt{false} \to \Pi b : \mathbb{B}.\, P\ b$$

- Modern avatar of "Axiom of choice in classical logic is fishy".

BLATANT ADVERTISMENT

Come to see my LICS talk for a potential generic solution to CIC + effects!

BLATANT ADVERTISMENT

Come to see my LICS talk for a potential generic solution to CIC + effects!

Restrict dependent eliminations to semantically call-by-value predicates.

Buzzword: *linearity*. (Little to do with syntactic linearity BTW.)

$$\frac{\Gamma \vdash M : \mathbb{B} \qquad \Gamma \vdash N_1 : P\,\mathtt{true} \qquad \Gamma \vdash N_2 : P\,\mathtt{false} \qquad P \text{ linear in } b}{\Gamma \vdash \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 : P\{b := M\}}$$

BLATANT ADVERTISMENT

Come to see my LICS talk for a potential generic solution to CIC + effects!

Restrict dependent eliminations to semantically call-by-value predicates.

Buzzword: *linearity*. (Little to do with syntactic linearity BTW.)

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

- Works for CBN forcing
- Works for our new weaning translation
- Inspired by classical realizability
- Prevents Herbelin's particular paradox
- **Unluckily, a consistent model of callcc is still missing!**

In This Talk: Program Translations

Observations:

- Morale of Attempt 1: Axioms are both unwieldy and fishy.
- Morale of Attempt 2: Arbitrary computational primitives are fishier.

In This Talk: Program Translations

Observations:

- Morale of Attempt 1: Axioms are both unwieldy and fishy.
- Morale of Attempt 2: Arbitrary computational primitives are fishier.

OTOH, a well-known program translation implementing `callcc`.

Continuation-passing style!

In This Talk: Program Translations

Observations:

- Morale of Attempt 1: Axioms are both unwieldy and fishy.
- Morale of Attempt 2: Arbitrary computational primitives are fishier.

OTOH, a well-known program translation implementing `callcc`.

Continuation-passing style!

We propose in this talk a much less grand solution than linearity.

The first *cheating* CPS translation of CIC.

Syntactic Models, a.k.a. Program Translations of CIC

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

$$\vdash_{\mathsf{CIC}^+} M : A \qquad \text{implies} \qquad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

Syntactic Models, a.k.a. Program Translations of CIC

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

$$\vdash_{\mathsf{CIC}^+} M : A \qquad \text{implies} \qquad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

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, a.k.a. Program Translations of CIC

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

$$\vdash_{\mathsf{CIC}^+} M : A \qquad \text{implies} \qquad \vdash_{\mathsf{CIC}} [M] : [\![A]\!]$$

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

Baby steps

CIC is call-by-name by construction.

That's because of the β-equivalence used in conversion.

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

Baby steps

CIC is call-by-name by construction.

That's because of the β-equivalence used in conversion.

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

We have to use a CBN CPS translation.

Let's stick to a variant close to the hardware: Lafont-Streicher-Reus CPS.

(This is LOLA after all.)

Quick recap

In the simply-typed case, the LSR CPS is given as follows.

Quick recap

In the simply-typed case, the LSR CPS is given as follows.

1. Fix some return type $\perp\!\!\!\perp$.

Quick recap

In the simply-typed case, the LSR CPS is given as follows.

1. Fix some return type $\perp\!\!\!\perp$.
2. Inductively define the type of stacks $\mathbb{C}(A)$ and witnesses $\mathbb{W}(A)$.

$$\begin{array}{lcl} \mathbb{W}(A) & := & \mathbb{C}(A) \rightarrow \perp\!\!\!\perp \\ \mathbb{C}(\alpha) & := & \alpha \rightarrow \perp\!\!\!\perp \\ \mathbb{C}(A \rightarrow B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \end{array}$$

Quick recap

In the simply-typed case, the LSR CPS is given as follows.

1. Fix some return type $\perp\!\!\!\perp$.
2. Inductively define the type of stacks $\mathbb{C}(A)$ and witnesses $\mathbb{W}(A)$.

$$\begin{array}{lcl} \mathbb{W}(A) & := & \mathbb{C}(A) \to \perp\!\!\!\perp \\ \mathbb{C}(\alpha) & := & \alpha \to \perp\!\!\!\perp \\ \mathbb{C}(A \to B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \end{array}$$

3. Define the term translation $[\cdot]$ on the syntax s.t.

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad \mathbb{W}(\Gamma) \vdash [M] : \mathbb{W}(A)$$

This Is LOLA After All

Here is the implementation:

$$\begin{array}{lcl} [x] & := & x \\ [\lambda x.\, M] & := & \lambda(x, \omega).\, [M]\ \omega \\ [M\ N] & := & \lambda \omega.\, [M]\ (N, \omega) \end{array}$$

This Is LOLA After All

Here is the implementation:

$$\begin{array}{lcl} [x] & := & x \\ [\lambda x.\, M] & := & \lambda(x, \omega).\, [M]\ \omega \\ [M\ N] & := & \lambda \omega.\, [M]\ (N, \omega) \end{array}$$

Holy celestial teapot! It implements the Krivine machine!

$$\begin{array}{lcl} \langle \lambda x.\, M \mid N \cdot \pi \rangle & \rightarrow & \langle M\{x := N\} \mid \pi \rangle \\ \langle M\ N \mid \pi \rangle & \rightarrow & \langle M \mid N \cdot \pi \rangle \end{array}$$

This Is LOLA After All

Here is the implementation:

$$\begin{array}{lcl} [x] & := & x \\ [\lambda x.\, M] & := & \lambda(x, \omega).\, [M]\ \omega \\ [M\ N] & := & \lambda \omega.\, [M]\ (N, \omega) \end{array}$$

Holy celestial teapot! It implements the Krivine machine!

$$\begin{array}{lcl} \langle \lambda x.\, M \mid N \cdot \pi \rangle & \rightarrow & \langle M\{x := N\} \mid \pi \rangle \\ \langle M\ N \mid \pi \rangle & \rightarrow & \langle M \mid N \cdot \pi \rangle \end{array}$$

Plus there is a proof of:

$$\mathbb{W}(((A \rightarrow B) \rightarrow A) \rightarrow A)$$

mimicking what the classical KAM does.

CICking it out

So far so good, we have a syntactic model for simply-typed λ-calculus.

Sketchy roadmap of what we have to do to scale LSR to CIC:

1. Acknowledging dependent functions
2. Implementing types-as-terms
3. Implementing dependent elimination

CICking it out

So far so good, we have a syntactic model for simply-typed λ-calculus.

Sketchy roadmap of what we have to do to scale LSR to CIC:

1. Acknowledging dependent functions
2. Implementing types-as-terms
3. Implementing dependent elimination

Spoiler: Turns out 1. is trivial, 2. and 3. impossible as-is.

LSR and dependency

Owing to the low-level nature of LSR, dependency is trivial.

$$
\begin{array}{lcl}
\mathbb{W}(A) & := & \mathbb{C}(A) \to \perp\!\!\!\perp \\
\mathbb{C}(A \to B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \\
\mathbb{C}(\Pi x : A.\, B) & := & \Sigma x : \mathbb{W}(A).\, \mathbb{C}(B)
\end{array}
$$

Remark in particular that the arrow case is a degenerate variant.

It means it is easy to give a LSR of $\lambda\Pi$ s.t.

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad \mathbb{W}(\Gamma) \vdash [M]\mathbb{W}(A)$$

LSR and dependency

Owing to the low-level nature of LSR, dependency is trivial.

$$
\begin{array}{lcl}
\mathbb{W}(A) & := & \mathbb{C}(A) \to \perp\!\!\!\perp \\
\mathbb{C}(A \to B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \\
\mathbb{C}(\Pi x : A.\, B) & := & \Sigma x : \mathbb{W}(A).\, \mathbb{C}(B)
\end{array}
$$

Remark in particular that the arrow case is a degenerate variant.

It means it is easy to give a LSR of $\lambda\Pi$ s.t.

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad \mathbb{W}(\Gamma) \vdash [M]\mathbb{W}(A)$$

Note: not as easy for other CBN CPS! So LSR is good for dependency.

LSR and inductive types

In LSR, inductive types are translated free algebras, e.g.

$$\begin{array}{lcl} \mathbb{C}(\mathbb{B}) & := & \mathbb{B} \to \perp\!\!\!\perp \\ \mathbb{W}(\mathbb{B}) & := & (\mathbb{B} \to \perp\!\!\!\perp) \to \perp\!\!\!\perp \end{array}$$

Constructors are returns, elimination is continuation-passing.

$$\begin{array}{lcl} [\mathtt{true}] & := & \lambda\omega.\,\omega\ \mathtt{true} \\ [\mathtt{false}] & := & \lambda\omega.\,\omega\ \mathtt{false} \\ [\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2] & := & \lambda\omega.\,[M]\ (\lambda b.\,\mathtt{if}\ b\ \mathtt{then}\ [N_1]\ \omega\ \mathtt{else}\ [N_2]\ \omega) \end{array}$$

LSR and inductive types: a failure

Alas, no hope to implement dependent elimination!

$$\Pi P : \mathbb{B} \to \square.\, P\ \mathtt{true} \to P\ \mathtt{false} \to \Pi b : \mathbb{B}.\, P\ b$$

$\rightsquigarrow$ For a meta-theoretical reason:

$\mathbb{W}(\mathbb{B}) := (\mathbb{B} \to \perp\!\!\!\perp) \to \perp\!\!\!\perp$, so depending on the choice of $\perp\!\!\!\perp$ there are non-standard booleans.

$\rightsquigarrow$ For a technical reason:

In the typing of `if`, the type of a dependent ω would be wrong.

$$[\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2] \quad := \quad \lambda\omega.\, [M]\ (\lambda b.\, \mathtt{if}\ b\ \mathtt{then}\ [N_1]\ \omega\ \mathtt{else}\ [N_2]\ \omega)$$

LSR and inductive types: a failure

Alas, no hope to implement dependent elimination!

$$\Pi P : \mathbb{B} \rightarrow \square.\, P\ \mathtt{true} \rightarrow P\ \mathtt{false} \rightarrow \Pi b : \mathbb{B}.\, P\ b$$

$\rightsquigarrow$ For a meta-theoretical reason:

$\mathbb{W}(\mathbb{B}) := (\mathbb{B} \rightarrow \perp\!\!\!\perp) \rightarrow \perp\!\!\!\perp$, so depending on the choice of $\perp\!\!\!\perp$ there are non-standard booleans.

$\rightsquigarrow$ For a technical reason:

In the typing of `if`, the type of a dependent ω would be wrong.

$$[\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2] \quad := \quad \lambda\omega.\, [M]\ (\lambda b.\, \mathtt{if}\ b\ \mathtt{then}\ [N_1]\ \omega\ \mathtt{else}\ [N_2]\ \omega)$$

No way to recover an actual boolean from a classical boolean.

LSR and universes: failure again

Because $\vdash_{\mathsf{CIC}} \square_i : \square_{i+1}$, we must define $\mathbb{C}(\square_i)$.

Universes are somehow free algebras, so take $\mathbb{C}(\square_i) := \square_i \to \perp\!\!\!\perp$.
In particular, $\mathbb{W}(\square_i) := (\square_i \to \perp\!\!\!\perp) \to \perp\!\!\!\perp$.

LSR and universes: failure again

Because $\vdash_{\mathsf{CIC}} \square_i : \square_{i+1}$, we must define $\mathbb{C}(\square_i)$.

Universes are somehow free algebras, so take $\mathbb{C}(\square_i) := \square_i \to \perp\!\!\!\perp$.
In particular, $\mathbb{W}(\square_i) := (\square_i \to \perp\!\!\!\perp) \to \perp\!\!\!\perp$.

Now, how to implement the meta-function $\mathrm{El} : \mathbb{W}(\square) \rightsquigarrow \square$, needed for

$$\frac{\Gamma \vdash A : \square_i}{\vdash \Gamma, A : \square_i}$$

Actually, you can't. Just as for booleans, double-negation lost information.

No way to recover an actual type from a classical type either.

A Dire Situation

TL; DR: LSR handles negative connectives but not positive ones.

Not totally unexpected from a CPS translation...

How to solve this? It looks inherent to the CPS.

A Dire Situation

TL; DR: LSR handles negative connectives but not positive ones.

Not totally unexpected from a CPS translation...

How to solve this? It looks inherent to the CPS.

Let's cheat!

.

TL; DR: LSR handles negative connectives but not positive ones.

Not totally unexpected from a CPS translation...

How to solve this? It looks inherent to the CPS.

Let's cheat!

.

Let's make the CPS intuitionistic again by using..

Parametricity.

Or equivalently, let's do a bit of...

Intuitionistic realizability.

The Grand Scheme

We lost information in the CPS, let's add it back as a side-condition.

The Grand Scheme

We lost information in the CPS, let's add it back as a side-condition.

Idea: instead of translating

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad \mathbb{W}(\Gamma) \vdash [M] : \mathbb{W}(A)$$

let's rather do

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad [\![\Gamma]\!] \vdash [M]^{!} : [\![A]\!]$$

where

$$[\![A]\!] := \Sigma x : \mathbb{W}(A).\, x \in A \quad \text{and} \quad [M]^{!} := ([M], [M]_{\varepsilon})$$

The Grand Scheme

We lost information in the CPS, let's add it back as a side-condition.

Idea: instead of translating

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad \mathbb{W}(\Gamma) \vdash [M] : \mathbb{W}(A)$$

let's rather do

$$\Gamma \vdash M : A \quad \rightsquigarrow \quad [\![\Gamma]\!] \vdash [M]^{!} : [\![A]\!]$$

where

$$[\![A]\!] := \Sigma x : \mathbb{W}(A).\, x \in A \quad \text{and} \quad [M]^{!} := ([M], [M]_{\varepsilon})$$

We will retrieve the information in $\cdot \in A$ rather than in $\mathbb{W}(A)$!

$M \in A$ is the parametricity (resp. realizability) relation of A.

The Grand Scheme II

Morally, our translation is

- Intuitionistic Realizability (Kleene-style?)
- ... where realizers are Lafont-Streicher-Reus CPS-ified terms
- ... and where the realizability relation is internal to CIC

A fancy mix... Is that a known technique?

The Grand Scheme II

Morally, our translation is

- Intuitionistic Realizability (Kleene-style?)
- ... where realizers are Lafont-Streicher-Reus CPS-ified terms
- ... and where the realizability relation is internal to CIC

A fancy mix... Is that a known technique?

Has it a use per se? Can it be used for type-preserving compilation?

A Bit of Detail

Compared from the simply-typed case, $[\cdot]$ is unchanged.

I will not give $[\cdot]_\varepsilon$ here, but it is straightforward. More or less a projection.

A Bit of Detail

Compared from the simply-typed case, $[\cdot]$ is unchanged.

I will not give $[\cdot]_\varepsilon$ here, but it is straightforward. More or less a projection.

We define the realizability condition as follows:

A	$\mathbb{C}(A)$	$(M : \mathbb{C}(A) \to \perp\!\!\!\perp) \in A$
$\Pi x : A.\, B$	$\Sigma x : [\![A]\!].\, \mathbb{C}(B)$	$\Pi x : [\![A]\!].\, (\lambda \omega.\, M\ (x, \omega)) \in B$

A Bit of Detail

Compared from the simply-typed case, $[\cdot]$ is unchanged.

I will not give $[\cdot]_\varepsilon$ here, but it is straightforward. More or less a projection.

We define the realizability condition as follows:

A	$\mathbb{C}(A)$	$(M : \mathbb{C}(A) \to \perp\!\!\!\perp) \in A$
$\Pi x : A.\, B$	$\Sigma x : [\![A]\!].\, \mathbb{C}(B)$	$\Pi x : [\![A]\!].\, (\lambda \omega.\, M\ (x, \omega)) \in B$
$\mathbb{B}$	$\mathbb{B} \to \perp\!\!\!\perp$	$\Sigma b : \mathbb{B}. M = \texttt{ret}\ b$

A Bit of Detail

Compared from the simply-typed case, $[\cdot]$ is unchanged.

I will not give $[\cdot]_\varepsilon$ here, but it is straightforward. More or less a projection.

We define the realizability condition as follows:

A	$\mathbb{C}(A)$	$(M : \mathbb{C}(A) \rightarrow \perp\!\!\!\perp) \in A$
$\Pi x : A.\, B$	$\Sigma x : [\![A]\!].\, \mathbb{C}(B)$	$\Pi x : [\![A]\!].\, (\lambda \omega.\, M\ (x, \omega)) \in B$
$\mathbb{B}$	$\mathbb{B} \rightarrow \perp\!\!\!\perp$	$\Sigma b : \mathbb{B}. M = \texttt{ret}\ b$
$\square$	$\square \rightarrow \perp\!\!\!\perp$	$\Sigma X : \square.\, (M = \texttt{ret}\ X) \times ((M \rightarrow \perp\!\!\!\perp) \rightarrow \square)$

A Bit of Detail

Compared from the simply-typed case, $[\cdot]$ is unchanged.

I will not give $[\cdot]_\varepsilon$ here, but it is straightforward. More or less a projection.

We define the realizability condition as follows:

A	$\mathbb{C}(A)$	$(M : \mathbb{C}(A) \to \bot\!\!\!\bot) \in A$
$\Pi x : A.\, B$	$\Sigma x : [\![A]\!].\, \mathbb{C}(B)$	$\Pi x : [\![A]\!].\, (\lambda\omega.\, M\ (x, \omega)) \in B$
$\mathbb{B}$	$\mathbb{B} \to \bot\!\!\!\bot$	$\Sigma b : \mathbb{B}. M = \texttt{ret}\ b$
$\square$	$\square \to \bot\!\!\!\bot$	$\Sigma X : \square.\, (M = \texttt{ret}\ X) \times ((M \to \bot\!\!\!\bot) \to \square)$

Technically, $[\![A]\!]$, $\mathbb{C}(A)$ and $M \in A$ are macros derived from $[A]_\varepsilon$.

A Few Isomorphims

This translation is very intuitionistic, as it is somehow the identity.

Assuming $\perp\!\!\!\perp$ is hProp:

$$\begin{array}{ccc} [\![\Pi(x : A).\, B]\!] & \cong & \Pi(x : [\![A]\!]).\, [\![B]\!] \\ [\![\mathbb{B}]\!] & \cong & \mathbb{B} \\ [\![\mathtt{empty}]\!] & \cong & \mathtt{empty} \end{array}$$

In particular, it preserves consistency!

A Few Isomorphims

This translation is very intuitionistic, as it is somehow the identity.

Assuming $\perp\!\!\!\perp$ is hProp:

$$\begin{array}{ccc} [\![\Pi(x : A).\, B]\!] & \cong & \Pi(x : [\![A]\!]).\, [\![B]\!] \\ [\![\mathbb{B}]\!] & \cong & \mathbb{B} \\ [\![\texttt{empty}]\!] & \cong & \texttt{empty} \end{array}$$

In particular, it preserves consistency!

The only difference (due to parametricity):

$$[\![\square]\!] \ncong \square$$

Soundness

Interestingly, this translation can be carried in CIC.

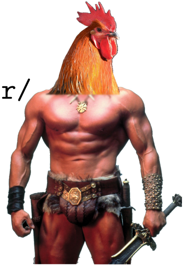
If $\Gamma \vdash_{\mathsf{CIC}} M : A$ then $[\![\Gamma]\!] \vdash_{\mathsf{CIC}} [M]^{!} : [\![A]\!]$

Soundness

Interestingly, this translation can be carried in CIC.

If $\Gamma \vdash_{\mathsf{CIC}} M : A$ then $[\![\Gamma]\!] \vdash_{\mathsf{CIC}} [M]^! : [\![A]\!]$

So it is possible to provide this translation as Coq plugin!
For now only a hand-written shallow embedding.
`https://github.com/CoqHott/coq-effects/blob/master/theories/misc/CPS.v`



Conservativity?

What did we gain? Not a lot of things...

- The resulting theory is **almost** a conservative extension of CIC
- For instance you can't implement `callcc` in general

Conservativity?

What did we gain? Not a lot of things...

- The resulting theory is **almost** a conservative extension of CIC
- For instance you can't implement `callcc` in general
- It is not for sordid reasons related to types (namely $[\![\square]\!] \not\cong \square$)
- ... in particular it **negates** univalence!

Conservativity?

What did we gain? Not a lot of things...

- The resulting theory is **almost** a conservative extension of CIC
- For instance you can't implement `callcc` in general
- It is not for sordid reasons related to types (namely $[\![\square]\!] \not\cong \square$)
- ... in particular it **negates** univalence!

That said, we have new statements in our theory.

Sprinkling Classical Logic

Because we carry classical realizers, we can actually fall back to LSR!

Sprinkling Classical Logic

Because we carry classical realizers, we can actually fall back to LSR!

Behold the classical modality $\langle\cdot\rangle$!

$$\begin{array}{lcl} \mathbb{C}(\langle A \rangle) & := & \mathbb{C}(A) \\ M \in \langle A \rangle & := & \texttt{unit} \end{array}$$

Sprinkling Classical Logic

Because we carry classical realizers, we can actually fall back to LSR!

Behold the classical modality $\langle\cdot\rangle$!

$$\begin{array}{lcl} \mathbb{C}(\langle A\rangle) & := & \mathbb{C}(A) \\ M \in \langle A\rangle & := & \texttt{unit} \end{array}$$

The modality just drops the parametric proof of the underlying type.

$$[\![\langle A\rangle]\!] := \Sigma x : \mathbb{W}(A).\,\texttt{unit} \cong \mathbb{W}(A)$$

As such, it allows to work with the raw LSR translation.

Moar Principles

This type constructor admits a lot of reasoning principles.

- It has a return:

$$\eta : \Pi(A : \Box).\, A \to \langle A \rangle$$

- It has (a weak form of) choice:

$$\Pi(x : A).\, \langle B \rangle \cong \langle \Pi(x : A).\, B \rangle$$

- It has a form of classical reasoning:

$$\mathsf{cc} : \Pi(A\ B : \Box).\, ((A \to \langle B \rangle) \to \langle A \rangle) \to \langle A \rangle$$

- It is **not** functorial.

$$A \to B \not\vdash \langle A \rangle \to \langle B \rangle$$

- In particular, it is **not** the double negation modality.

Give Me My Propositional Logic Back

Piggy-backing on LSR, we get an embedding of propositional logic.

If $\vdash_{\mathsf{LJ}} A$ then $\vdash_{\mathsf{CIC}^+} \langle A \rangle$.

Give Me My Propositional Logic Back

Piggy-backing on LSR, we get an embedding of propositional logic.

If $\vdash_{\mathsf{LJ}} A$ then $\vdash_{\mathsf{CIC}^+} \langle A \rangle$.

Furthermore, the propositional logic combinators compute. E.g.

$$\mathtt{if}^{\langle\cdot\rangle} : \langle \mathbb{B} \rangle \to \langle A \rangle \to \langle A \rangle \to \langle A \rangle$$

$$\mathtt{if}^{\langle\cdot\rangle}\ (\eta\ \mathbb{B}\ \mathtt{true})\ N_1\ N_2 \equiv_\beta N_1$$

This is all because the LSR CPS is well-behaved w.r.t. β-reduction.

Obviously no dependent elimination in sight. (Because LSR.)

Even More

For particular values of $\perp\!\!\!\perp$, we get more. Typically, for $\perp\!\!\!\perp := \texttt{empty}$.

- The modality is consistent.

$$\langle \texttt{empty} \rangle \rightarrow \texttt{empty}$$

- The modality has excluded middle.

$$\texttt{em} : \Pi(A : \square).\, \langle A + \neg A \rangle$$

Use cases?

What can we do with this modality? Not clear.

Use cases?

What can we do with this modality? Not clear.

When $\perp\!\!\!\perp := \texttt{empty}$, we can escape from it into falsity.

Allows to fake the existence of classical logic in a systematic way.

The Coq user should be happy!

Use cases?

What can we do with this modality? Not clear.

When $\perp\!\!\!\perp :=$ `empty`, we can escape from it into falsity.

Allows to fake the existence of classical logic in a systematic way.

The Coq user should be happy!

When $\perp\!\!\!\perp$ is some other type, one can use it as delimited continuations.

What can we do with that?

Use cases?

What can we do with this modality? Not clear.

When $\perp\!\!\!\perp := \texttt{empty}$, we can escape from it into falsity.

Allows to fake the existence of classical logic in a systematic way.

The Coq user should be happy!

When $\perp\!\!\!\perp$ is some other type, one can use it as delimited continuations.

What can we do with that?

Conclusion

- The first typed CPS of CIC!
- Although we cheat badly.

Conclusion

- The first typed CPS of CIC!
- Although we cheat badly.
- An intricate mix of techniques.
- Implementable in Coq.

Conclusion

- The first typed CPS of CIC!
- Although we cheat badly.
- An intricate mix of techniques.
- Implementable in Coq.
- A modality $\langle \cdot \rangle$ introducing classical logic.
- Preserving the propositional fragment, not dependent elimination.

Again, what can we do with that?

Thanks for your attention.