The Dialectica Translation of Type Theory

Andrej Bauer **Pierre-Marie Pédrot**

University of Ljubljana INRIA

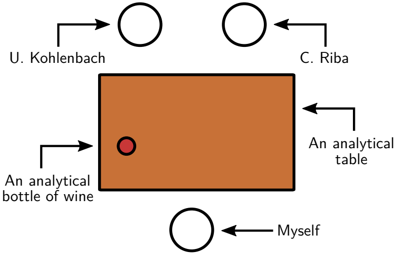
TYPES
24th May 2016

Previously at TYPES...

Analytical description of the TYPES 2013 social event (Toulouse)

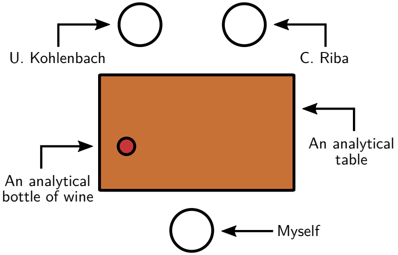
Previously at TYPES...

Analytical description of the TYPES 2013 social event (Toulouse)



Previously at TYPES...

Analytical description of the TYPES 2013 social event (Toulouse)



DRAMATIS PERSONAE:

- ULRICH KOHLENBACH, King of Dialectica
- COLIN RIBA, a Proof-Theory Gentleman
- PIERRE-MARIE PÉDROT, a Novice PhD Student
- THE BOTTLE OF WINE

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN O, Dialectica, the mysterious functional interpretation!

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN	O, Dialectica, the mysterious functional interpretation!
ULRICH	*nods*.

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN	O, Dialectica, the mysterious functional interpretation!
ULRICH	*nods*.
COLIN	For thou canst not be understood through Curry-Howard!

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN	O, Dialectica, the mysterious functional interpretation!
ULRICH	*nods*.
COLIN	For thou canst not be understood through Curry-Howard!
ULRICH	*nods*.

Previously at TYPES...

The Bottle of Wine *is almost empty.* Colin, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* Gödel.

Colin	O, Dialectica, the mysterious functional interpretation!
Ulrich	*nods.*
Colin	For thou canst not be understood through Curry-Howard!
Ulrich	*nods.*
P.-M.	That can't be true!

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN	O, Dialectica, the mysterious functional interpretation!
ULRICH	*nods.*
COLIN	For thou canst not be understood through Curry-Howard!
ULRICH	*nods.*
P.-M.	That can't be true!
ULRICH	*nods.*

Previously at TYPES...

The BOTTLE OF WINE *is almost empty.* COLIN, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* GÖDEL.

COLIN	O, Dialectica, the mysterious functional interpretation!
ULRICH	*nods.*
COLIN	For thou canst not be understood through Curry-Howard!
ULRICH	*nods.*
P.-M.	That can't be true!
ULRICH	*nods.*

The BOTTLE OF WINE *is empty. The characters disappear in a blurred mist. Noone can really recollect this dialogue.*

Previously at TYPES...

The Bottle of Wine *is almost empty.* Colin, *carried away by the enthusiasm of proof theory, begins to claim his love for the works of* Gödel.

Colin	O, Dialectica, the mysterious functional interpretation!
Ulrich	*nods.*
Colin	For thou canst not be understood through Curry-Howard!
Ulrich	*nods.*
P.-M.	That can't be true!
Ulrich	*nods.*

The Bottle of Wine *is empty. The characters disappear in a blurred mist. Noone can really recollect this dialogue.*

But I had found the matter for my PhD!

(Morale: you definitely should attend social events.)

A Quick Recap

- Dialectica is a logical translation due to Gödel
- Nowadays would be called a *realizability* intepretation

$$\vdash_{\mathbf{HA}} \pi : A \quad \rightsquigarrow \quad \begin{cases} \text{A } \lambda\text{-term } \pi^{\bullet} : [\![A]\!] \\ \text{A logical property } \pi^{\bullet} \Vdash A \text{ in the meta} \end{cases}$$

A Quick Recap

- Dialectica is a logical translation due to Gödel
- Nowadays would be called a *realizability* intepretation

$$\vdash_{\mathbf{HA}} \pi : A \quad \rightsquigarrow \quad \begin{cases} \text{A } \lambda\text{-term } \pi^\bullet : [\![A]\!] \\ \text{A logical property } \pi^\bullet \Vdash A \text{ in the meta} \end{cases}$$

- It preserves consistency, i.e. there is no $\pi : [\![\bot]\!]$ s.t. $\pi \Vdash \bot$
- It interprets strictly more than $\mathbf{HA}$, namely:

$$\begin{array}{ll} \mathrm{MP} : \neg(\forall x : \mathbb{N}. \neg P) \to \exists x : \mathbb{N}. P & (P \text{ decidable}) \\ \mathrm{IP} : (I \to \exists x : \mathbb{N}.P) \to \exists x : \mathbb{N}. I \to P & (I \text{ irrelevant}) \end{array}$$

Curry-Howard & Realizability

"Realizability interpretations tend to hide a programming translation."

Logic	Programming
Kreisel modified realizability	Identity translation
Krivine classical realizability	Lafont-Reus-Streicher CPS
Gödel Dialectica realizability	**?**

Curry-Howard & Realizability

"Realizability interpretations tend to hide a programming translation."

Logic	Programming
Kreisel modified realizability	Identity translation
Krivine classical realizability	Lafont-Reus-Streicher CPS
Gödel Dialectica realizability	**A fancy one!**

"Realizability interpretations tend to hide a programming translation."

Logic	Programming
Kreisel modified realizability	Identity translation
Krivine classical realizability	Lafont-Reus-Streicher CPS
Gödel Dialectica realizability	**A fancy one!**

- Gives first-class status to stacks
- Features a computationally relevant substitution
- Mix of LRS with delimited continuations
- Requires computational (finite) multisets $\mathfrak{M}$

Program translation, did you say?

- It operates on raw syntax (no need for the typing derivation)

$$t \in \Lambda \qquad \rightsquigarrow \qquad t^\bullet \in \Lambda + \ldots$$

- It preserves typing:

$$t : A \qquad \rightsquigarrow \qquad t^\bullet : [\![A]\!]$$

- **It preserves syntactic program equality (conversion)**:

$$t \equiv_\beta u \qquad \rightsquigarrow \qquad t^\bullet \equiv_\beta u^\bullet$$

Program translation, did you say?

- It operates on raw syntax (no need for the typing derivation)

$$t \in \Lambda \qquad \rightsquigarrow \qquad t^\bullet \in \Lambda + \ldots$$

- It preserves typing:

$$t : A \qquad \rightsquigarrow \qquad t^\bullet : [\![A]\!]$$

- **It preserves syntactic program equality (conversion)**:

$$t \equiv_\beta u \qquad \rightsquigarrow \qquad t^\bullet \equiv_\beta u^\bullet$$

There is an itch: this requires multisets that compute definitionally

$$\emptyset \uplus t \equiv_\beta t \quad t \uplus u \equiv_\beta u \uplus t \quad (t \uplus u) \uplus r \equiv_\beta t \uplus (u \uplus r) \quad \ldots$$

Effectively

In CBN, the effect provided by Dialectica can be explained as follows:

$$\begin{array}{llcl} \text{From} & \lambda x.\, t & : & A \to B \\ & u & : & A \\ & \pi & : & B^{\perp} \\ \hline \text{Recover} & \mu & : & \mathfrak{M}\ A^{\perp} \end{array}$$

where:

- $X^{\perp}$ is the type of stacks accepting X (first-class contexts)
- μ is obtained by the following process:
  1. evaluate t on stack π
  2. each time t dereferences x, store the current stack ρ_i and continue with u
  3. when finished, return the multiset of all $[\rho_1; \ldots; \rho_n]$

Effectively

In CBN, the effect provided by Dialectica can be explained as follows:

$$
\begin{array}{llcl}
\text{From} & \lambda x.\, t & : & A \to B \\
 & u & : & A \\
 & \pi & : & B^{\perp} \\
\hline
\text{Recover} & \mu & : & \mathfrak{M}\ A^{\perp}
\end{array}
$$

where:

- $X^{\perp}$ is the type of stacks accepting X (first-class contexts)
- μ is obtained by the following process:
  1. evaluate t on stack π
  2. each time t dereferences x, store the current stack ρ_i and continue with u
  3. when finished, return the multiset of all $[\rho_1; \ldots; \rho_n]$

Thus Dialectica instruments stack manipulation and substitution.

Effectively

In CBN, the effect provided by Dialectica can be explained as follows:

$$\begin{array}{llcl} \text{From} & \lambda x.\, t & : & A \to B \\ & u & : & A \\ & \pi & : & B^{\perp} \\ \hline \text{Recover} & \mu & : & \mathfrak{M}\ A^{\perp} \end{array}$$

where:

- $X^{\perp}$ is the type of stacks accepting X (first-class contexts)
- μ is obtained by the following process:
  1. evaluate t on stack π
  2. each time t dereferences x, store the current stack ρ_i and continue with u
  3. when finished, return the multiset of all $[\rho_1; \ldots; \rho_n]$

Thus Dialectica instruments stack manipulation and substitution.

In practice

The translation goes roughly as follows:

$$A \text{ type} \quad \rightsquigarrow \quad \begin{cases} \mathbb{W}(A) \text{ witness type: type of objects} \\ \mathbb{C}(A) \text{ counter type: type of stacks} \end{cases}$$

In practice

The translation goes roughly as follows:

$$A \text{ type} \quad \rightsquigarrow \quad \begin{cases} \mathbb{W}(A) \text{ witness type: type of objects} \\ \mathbb{C}(A) \text{ counter type: type of stacks} \end{cases}$$

In particular,

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

There is a special translation handling open terms:

$$x_1 : \Gamma_1, \ldots, x_n : \Gamma_n \vdash t : A \quad \rightsquigarrow \quad \begin{cases} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathfrak{M}\ \mathbb{C}(\Gamma_1) \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathfrak{M}\ \mathbb{C}(\Gamma_n) \end{cases}$$

Moar!

This translation is actually easily adapted to the dependent case.

There is a Dialectica translation for $\mathbf{CC}_\omega$ (by making stuff dependent).

Moar!

This translation is actually easily adapted to the dependent case.

There is a Dialectica translation for $\mathbf{CC}_\omega$ (by making stuff dependent).

And you can also account for algebraic datatypes.

There is a Dialectica translation for $+, \times, \ldots$ (by a $\mathbf{LL}$ decomposition).

Moar!

This translation is actually easily adapted to the dependent case.

There is a Dialectica translation for $\mathbf{CC}_\omega$ (by making stuff dependent).

And you can also account for algebraic datatypes.

There is a Dialectica translation for $+, \times, \ldots$ (by a $\mathbf{LL}$ decomposition).

But it seems you can't have the full power of dependent elimination.

Interpreting dependent elimination through Dialectica looks complicated.

Moar!

This translation is actually easily adapted to the dependent case.

There is a Dialectica translation for $\mathbf{CC}_\omega$ (by making stuff dependent).

And you can also account for algebraic datatypes.

There is a Dialectica translation for $+, \times, \ldots$ (by a $\mathbf{LL}$ decomposition).

But it seems you can't have the full power of dependent elimination.

Interpreting dependent elimination through Dialectica looks complicated.

`</End of the recap of my PhD>`

Decomposing Dialectica

- In her PhD, De Paiva gave a $\mathbf{LL}$ decomposition of Dialectica
- Root of double-glueing constructions
- Although it works, it inherits from the quirks of $\mathbf{LL}$

Decomposing Dialectica

- In her PhD, De Paiva gave a $\mathbf{LL}$ decomposition of Dialectica
- Root of double-glueing constructions
- Although it works, it inherits from the quirks of $\mathbf{LL}$

- We give a new decomposition in $\mathbf{CBPV}$
- It is inherently highly dependent
- And it naturally provides an interpretation for the whole $\mathbf{CIC}$

CBPV

CBPV is a syntax for a pervasive class of models

$$
\begin{array}{llll}
\text{value types} & A, B & := & \mathbf{U}\,X \mid A + B \mid A \times B \mid \ldots \\
\text{computation types} & X, Y & := & \mathbf{F}\,A \mid A \to X \mid \ldots \\
\text{values} & v, w & := & \ldots \\
\text{computations} & t, u & := & \ldots
\end{array}
$$

Essentially, it decomposes Moggi's monadic language in an adjunction

$$T\,A := \mathbf{U}\,(\mathbf{F}\,A)$$

Thus, finer-grained.

CBPV

CBPV is a syntax for a pervasive class of models

$$
\begin{array}{llcl}
\text{value types} & A, B & := & \mathbf{U}\, X \mid A + B \mid A \times B \mid \ldots \\
\text{computation types} & X, Y & := & \mathbf{F}\, A \mid A \to X \mid \ldots \\
\text{values} & v, w & := & \ldots \\
\text{computations} & t, u & := & \ldots
\end{array}
$$

Essentially, it decomposes Moggi's monadic language in an adjunction

$$T\, A := \mathbf{U}\, (\mathbf{F}\, A)$$

Thus, finer-grained.

(We actually studied a dependently-typed variant, although not really thought about.)

Key idea of the decomposition

- We translate value and computation types alike
- The $\mathbb{C}(\cdot)$ type now crucially depends on a corresponding $\mathbb{W}(\cdot)$, i.e.

$$\mathbb{W}(A) : \square \qquad\qquad \mathbb{C}(A)[\cdot] : \mathbb{W}(A) \to \square$$

Key idea of the decomposition

- We translate value and computation types alike
- The $\mathbb{C}(\cdot)$ type now crucially depends on a corresponding $\mathbb{W}(\cdot)$, i.e.

$$\mathbb{W}(A) : \square \qquad\qquad \mathbb{C}(A)[\cdot] : \mathbb{W}(A) \to \square$$

$$\begin{array}{rcl}
\mathbb{W}(A \to X) & := & \Pi x : \mathbb{W}(A).\, \Sigma y : \mathbb{W}(X).\, (\mathbb{C}(X)[y] \to \mathfrak{M}\ \mathbb{C}(A)[x]) \\
\mathbb{C}(A \to X)[f] & := & \Sigma x : \mathbb{W}(A).\, \mathbb{C}(X)[\mathtt{snd}\ (f\ x)]
\end{array}$$

$$\begin{array}{rcl}
\mathbb{W}(\mathbf{F}\ A) & := & \mathbb{W}(A) \\
\mathbb{C}(\mathbf{F}\ A)[x] & := & \mathfrak{M}\ \mathbb{C}(A)[x]
\end{array}$$

$$\begin{array}{rcl}
\mathbb{W}(\mathbf{U}\ X) & := & \mathbb{W}(X) \\
\mathbb{C}(\mathbf{U}\ X)[x] & := & \mathbb{C}(X)[x]
\end{array}$$

Sequents

This naturally gives rise to the interpretation:

$$x_1 : \Gamma_1, \ldots, x_n : \Gamma_n \vdash t : X \quad \rightsquigarrow \quad \begin{cases} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(X) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(X)[t^\bullet] \to \mathfrak{M}\ \mathbb{C}(\Gamma_1)[x_1] \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(X)[t^\bullet] \to \mathfrak{M}\ \mathbb{C}(\Gamma_n)[x_n] \end{cases}$$

This naturally gives rise to the interpretation:

$$x_1 : \Gamma_1, \ldots, x_n : \Gamma_n \vdash t : X \quad \rightsquigarrow \quad \begin{cases} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(X) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(X)[t^\bullet] \to \mathfrak{M}\ \mathbb{C}(\Gamma_1)[x_1] \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(X)[t^\bullet] \to \mathfrak{M}\ \mathbb{C}(\Gamma_n)[x_n] \end{cases}$$

We never use the counter argument and merely pass it around!

In absence of datatypes, this is the **same** as the previous translation

Datatypes, at least

This counter dependency is only required for dependent elimination!

$$
\begin{array}{rcl}
\mathbb{W}(A \times B) & := & \mathbb{W}(A) \times \mathbb{W}(B) \\
\mathbb{C}(A \times B)[(x, y)] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \times \mathfrak{M}\ \mathbb{C}(B)[y]
\end{array}
$$

$$
\begin{array}{rcl}
\mathbb{W}(A + B) & := & \mathbb{W}(A) + \mathbb{W}(B) \\
\mathbb{C}(A + B)[\texttt{inl}\ x] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \\
\mathbb{C}(A + B)[\texttt{inr}\ y] & := & \mathfrak{M}\ \mathbb{C}(B)[y]
\end{array}
$$

Datatypes, at least

This counter dependency is only required for dependent elimination!

$$\begin{array}{rcl} \mathbb{W}(A \times B) & := & \mathbb{W}(A) \times \mathbb{W}(B) \\ \mathbb{C}(A \times B)[(x, y)] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \times \mathfrak{M}\ \mathbb{C}(B)[y] \end{array}$$

$$\begin{array}{rcl} \mathbb{W}(A + B) & := & \mathbb{W}(A) + \mathbb{W}(B) \\ \mathbb{C}(A + B)[\mathtt{inl}\ x] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \\ \mathbb{C}(A + B)[\mathtt{inr}\ y] & := & \mathfrak{M}\ \mathbb{C}(B)[y] \end{array}$$

The argument is crucially observed through dependent elimination.
There is an implicit pattern-matching at the head of the definition.

An interesting remark

In absence of dependency, those types are emulated by being less precise. Typically, compare the dependent:

$$\begin{array}{rcl} \mathbb{C}(A \times B)[(x, y)] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \times \mathfrak{M}\ \mathbb{C}(B)[y] \\ \mathbb{C}(A + B)[\texttt{inl}\ x] & := & \mathfrak{M}\ \mathbb{C}(A)[x] \\ \mathbb{C}(A + B)[\texttt{inr}\ y] & := & \mathfrak{M}\ \mathbb{C}(B)[y] \end{array}$$

with the $\mathbf{LL}$-induced:

$$\begin{array}{rcl} \mathbb{C}(A \times B) & := & \mathbb{W}(A) \times \mathbb{W}(B) \to \mathfrak{M}\ \mathbb{C}(A) \times \mathfrak{M}\ \mathbb{C}(B) \\ \mathbb{C}(A + B) & := & (\mathbb{W}(A) \to \mathfrak{M}\ \mathbb{C}(A)) \times (\mathbb{W}(B) \to \mathfrak{M}\ \mathbb{C}(B)) \end{array}$$

Exploiting the translation

There exists a Dialectica translation from $\mathbf{CIC}$ into $\mathbf{CIC} + \mathfrak{M}$.

Exploiting the translation

There exists a Dialectica translation from $\mathbf{CIC}$ into $\mathbf{CIC} + \mathfrak{M}$.

- Not entirely satisfying though.
- There is no such thing as computational multisets.
- Looks like their theory is decidable (?)
- Maybe we can implement a type-checker (?)

Exploiting the translation

There exists a Dialectica translation from $\mathbf{CIC}$ into $\mathbf{CIC} + \mathfrak{M}$.

- Not entirely satisfying though.
- There is no such thing as computational multisets.
- Looks like their theory is decidable (?)
- Maybe we can implement a type-checker (?)

In any case, we can't reuse an off-the-shelf implementation of type theory.

A funny, more intensional $\mathbf{CIC}$ (in CBN)

Through the translation, we get strictly more than $\mathbf{CIC}$.

$\mathbf{CIC}^D$ negates functional extensionality (and thus univalence):

$$\mathbf{CIC}^D \vdash \neg(\Pi f\ g.\,(\Pi x.\, f\ x = g\ x) \to f = g)$$

It is fairly trivial, because of the second component of arrows. E.g.:

$$\lambda\_.\,() \cong 0 \quad \text{vs.} \quad \lambda().\,() \cong 1 \quad \text{in} \quad 1 \to 1 \cong \mathbb{N}$$

A funny, more intensional $\mathbf{CIC}$ (in CBN)

Through the translation, we get strictly more than $\mathbf{CIC}$.

$\mathbf{CIC}^D$ negates functional extensionality (and thus univalence):

$$\mathbf{CIC}^D \vdash \neg(\Pi f\, g.\, (\Pi x.\, f\ x = g\ x) \to f = g)$$

It is fairly trivial, because of the second component of arrows. E.g.:

$$\lambda\_.\,() \cong 0 \quad \text{vs.} \quad \lambda().\,() \cong 1 \quad \text{in} \quad 1 \to 1 \cong \mathbb{N}$$

Yet it is not that badly behaved w.r.t. functions:

$\mathbf{CIC}^D$ preserves η-expansion.

Towards implicit complexity in Type Theory?

A generalization of the previous constatation:

$\mathbf{CIC}^{D}$ allows to count the uses of a function argument.

Indeed, the size of the multisets corresponds to the number of uses.

- It is not trivial, because very higher-order-ish
- In particular, the number of uses depends on the argument

E.g.: $\lambda b : \mathbb{B}.\, \mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ \mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ ()$

Towards implicit complexity in Type Theory?

A generalization of the previous constatation:

$\mathbf{CIC}^D$ allows to count the uses of a function argument.

Indeed, the size of the multisets corresponds to the number of uses.

- It is not trivial, because very higher-order-ish
- In particular, the number of uses depends on the argument

E.g.: $\lambda b : \mathbb{B}.\,\mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ \mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ ()$

Actually, somehow already known:

- by the proof mining community (majorability)
- by the linear logic community (quantitative semantics)

Towards implicit complexity in Type Theory?

A generalization of the previous constatation:

$\mathbf{CIC}^D$ allows to count the uses of a function argument.

Indeed, the size of the multisets corresponds to the number of uses.

- It is not trivial, because very higher-order-ish
- In particular, the number of uses depends on the argument

E.g.: $\lambda b : \mathbb{B}.\, \mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ \mathtt{if}\ b\ \mathtt{then}\ ()\ \mathtt{else}\ ()$

Actually, somehow already known:

- by the proof mining community (majorability)
- by the linear logic community (quantitative semantics)

Can we use it to implement implicit complexity in $\mathbf{CIC}^D$?

Food for thought

- Why is Dialectica inherently dependent?
- Can $\mathbf{LL}$ be encoded with dependent elimination?
- Does the implicit complexity stuff really requires multisets?
- How much can we fiddle with the $\mathbf{CBPV}$ decomposition?
- Can we merge $\mathbf{CBPV}$ and $\mathbf{LL}$?

Thanks for your attention.