A Materialist Dialectica

Pierre-Marie Pédrot

PPS/$\pi r^2$

17th September 2015
Part I.

« How Gödel became a computer scientist out of remorse »
From Axioms, applying valid Rules, derive a Conclusion.
Logic?

« The LOGICIST approach »

From Axioms, applying valid Rules, derive a Conclusion.

\[\begin{align*}
&\text{Socrates is a man.} & \text{All cats are mortal.} \\
&\text{All men are mortal.} & \text{Socrates is mortal.} \\
&\text{Thus Socrates is mortal.} & \text{Thus Socrates is a cat.}
\end{align*}\]

\[\begin{align*}
&\vdash A \rightarrow B & \vdash B \rightarrow C \\
\hline
&\vdash A \rightarrow C
\end{align*}\]

As long as rules are correct, you should be safe.
The LOGICIST approach

From Axioms, applying valid Rules, derive a Conclusion.

Socrates is a man.
All men are mortal.
Thus Socrates is mortal.

All cats are mortal.
Socrates is mortal.
Thus Socrates is a cat.

\[ \vdash A \rightarrow B \quad \vdash B \rightarrow C \]
\[ \vdash A \rightarrow C \]

As long as rules are correct, you should be safe.
Special emphasis on ensuring that they are indeed correct.
Logic: a long tradition of failure

- **3XX.** Aristotle predicts 50 years too late that Socrates had to die.

  Socrates is a man, all men are mortal, thus Socrates is mortal.
Logic: a long tradition of failure

- **3XX.** Aristotle predicts 50 years too late that Socrates had to die.
  
  Socrates is a man, all men are mortal, thus Socrates is mortal.

- **1641.** Descartes proves that God and unicorns exist.
  
  God is perfect, perfection implies existence, thus God exists.
- **3XX.** Aristotle predicts 50 years too late that Socrates had to die.

  Socrates is a man, all men are mortal, thus Socrates is mortal.

- **1641.** Descartes proves that God and unicorns exist.

  God is perfect, perfection implies existence, thus God exists.

- **1901.** Russell shows that there is no set of all sets.

  No one shall expel us from the Paradise that Cantor has created.
1931: Gödel’s incompleteness theorem

Assume a set of rules $S$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable
1931: Gödel’s incompleteness theorem

Assume a set of rules $S$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable

then

1. There is a sentence which is neither provable nor disprovable in $S$
2. The consistency of $S$ is neither provable nor disprovable in $S$
1931: Gödel’s incompleteness theorem

Assume a set of rules $S$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable

then

1. There is a sentence which is neither provable nor disprovable in $S$
2. The consistency of $S$ is neither provable nor disprovable in $S$

Quis ipsos custodiet custodes?
« Rather than trusting rules, let us trust experiments. »

- You need constructive logic
  
  From a proof of $\exists x. A[x]$ be able to recover a witness $t$ and a proof of $A[t]$.

- Suspicious principles

  $A \lor \neg A$  
  Excluded Middle

  $\neg \neg A \rightarrow A$  
  Reductio ad Absurdum

  $((A \rightarrow B) \rightarrow A) \rightarrow A$  
  Peirce’s Law

- From 1931, Gödel tried to atone for his incompleteness theorem

- Constructivizing non-constructive principles

  1. Double-negation translation (1933)
  2. Dialectica ('30s, published in 1958)
Gödel focussed on intuitionistic logic.

**The mathematician**
- A constructive logic
- Advocated by Brouwer for philosophical reasons (1920’s)
- Without the aforementioned suspicious axioms

**The computer scientist**
- Proofs are dynamic objects rather than static applications of rules (Gentzen ’33, Prawitz ’65)
- Witness extraction algorithmically recoverable
- From a proof one can extract a program (Kleene ’49)
Gödel focused on intuitionistic logic.

<table>
<thead>
<tr>
<th>The mathematician</th>
<th>The computer scientist</th>
</tr>
</thead>
<tbody>
<tr>
<td>A constructive logic</td>
<td>Proofs are dynamic objects rather than static applications of rules (Gentzen ’33, Prawitz ’65)</td>
</tr>
<tr>
<td>Advocated by Brouwer for philosophical reasons (1920’s)</td>
<td>Witness extraction algorithmically recoverable</td>
</tr>
<tr>
<td>Without the aforementioned suspicious axioms</td>
<td>From a proof one can extract a program (Kleene ’49)</td>
</tr>
</tbody>
</table>

Curry-Howard isomorphism (1960’s)

Intuitionistic proofs are \(\lambda\)-calculus programs
The beloved $\lambda$-calculus

The niftiest programming language of them all!

Terms  
\[ t ::= x \mid \lambda x. t \mid t \ u \mid \ldots \]

Types  
\[ A ::= \alpha \mid A \to B \mid \ldots \]

\[
\begin{align*}
(x : A) \in \Gamma & \quad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \lambda x. t : A \to B} \quad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash t u : B} \\
\Gamma \vdash x : A & \quad \frac{\Gamma \vdash x : A \vdash B}{\Gamma \vdash \lambda x. t : A \to B}
\end{align*}
\]

\[(\lambda x. t) \ u \to_\beta t[x := u]\]

Type derivations are proofs, compatible with $\beta$-reduction:

If $\Gamma \vdash t : A$ and $t \to_\beta r$ then $\Gamma \vdash r : A$. 
Which logic for which programs?

Standard members of each community will complain.

<table>
<thead>
<tr>
<th>The mathematician</th>
<th>The computer scientist</th>
</tr>
</thead>
<tbody>
<tr>
<td>“This logic is crappy, it does not feature the following principles I am acquainted with.”</td>
<td>“This language is crappy, it does not feature the following structures I am acquainted with.”</td>
</tr>
<tr>
<td>Either $A$ or not $A$ hold.</td>
<td>printf(&quot;Hello world&quot;)</td>
</tr>
<tr>
<td>Every bounded monotone sequence has a limit.</td>
<td>x &lt;- 42</td>
</tr>
<tr>
<td>Two sets with same elements are equal.</td>
<td>while true { ... }</td>
</tr>
<tr>
<td>Every formula is equivalent to its prenex form.</td>
<td>goto #considered_harmful</td>
</tr>
<tr>
<td>...</td>
<td>fork()</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Which logic for which programs?

Standard members of each community will complain.

<table>
<thead>
<tr>
<th>The mathematician</th>
<th>The computer scientist</th>
</tr>
</thead>
<tbody>
<tr>
<td>“This logic is crappy, it does not feature the following principles I am acquainted with.”</td>
<td>“This language is crappy, it does not feature the following structures I am acquainted with.”</td>
</tr>
<tr>
<td>Either $A$ or not $A$ hold.</td>
<td>printf(&quot;Hello world&quot;)</td>
</tr>
<tr>
<td>Every bounded monotone sequence has a limit.</td>
<td>x &lt;- 42</td>
</tr>
<tr>
<td>Two sets with same elements are equal.</td>
<td>while true { ... }</td>
</tr>
<tr>
<td>Every formula is equivalent to its prenex form.</td>
<td>goto #considered_harmful</td>
</tr>
<tr>
<td>...</td>
<td>fork()</td>
</tr>
</tbody>
</table>

INTUITIONISTIC LOGIC

FUNCTIONAL LANGUAGE
A new hope

What Curry-Howard takes, it gives back.
What Curry-Howard takes, it gives back.

New axioms $\sim$ Programming primitives

Logical encoding $\sim$ Program translation
The prototypical example: Classical logic

We can implement classical logic through this scheme.

New axiom

\[(A \rightarrow B) \rightarrow (A \rightarrow A) \rightarrow A\]

Used by your next-door mathematician

Logical encoding

double-negation translation

Gödel already did that by the 30’s

Programming primitives

\[\text{callcc}\]

From Scheme, though a bit arcane

Program translation

continuation-passing style

The dullest JS programmer uses this
Classical logic as a logical translation

Gödel-like presentation

Translation from classical logic into intuitionistic logic.

\[ A, B ::= \alpha \mid A \rightarrow B \mid A \times B \mid \bot \]

\begin{align*}
A^+ & \equiv A^- \rightarrow \bot \\
\alpha^- & \equiv \alpha \rightarrow \bot \\
(A \rightarrow B)^- & \equiv A^+ \times B^- \\
\ldots
\end{align*}

1. If \( \vdash A \) then \( \vdash A^+ \).
2. There is a proof of \( \vdash (((A \rightarrow B) \rightarrow A) \rightarrow A)^+ \).
Curry-Howard presentation

Translation from $\lambda$-calculus + $cc$ into $\lambda$-calculus.

$$A, B ::= \alpha | A \to B | A \times B | \bot$$

$$A^+ \equiv A^- \to \bot$$

$$\alpha^- \equiv \alpha \to \bot$$

$$(A \to B)^- \equiv A^+ \times B^-$$

$$x^* \equiv \lambda \omega. x \omega$$

$$(\lambda x. t)^* \equiv \lambda(x, \omega). t^* \omega$$

$$(t \ u)^* \equiv \lambda \omega. t^*(u^*, \omega)$$

...  ... 

1. If $\vdash t : A$ then $\vdash t^* : A^+$. 
2. $\vdash cc^* : (((A \to B) \to A) \to A)^+$. 
3. If $t \equiv_{\beta} u$ then $t^* \equiv_{\beta} u^*$. (Untyped!)
More is less

You lose part of your soul in the encoding.

If there is an **intuitionistic** proof of $\vdash A \lor B$ then $\vdash A$ or $\vdash B$.

There are **classical** proofs of $\vdash A \lor B$ s.t. neither $\vdash A$ nor $\vdash B$.

You may want something more fine-grained...
More is less

You lose part of your soul in the encoding.

If there is an **intuitionistic** proof of $\vdash A \lor B$ then $\vdash A$ or $\vdash B$.

There are **classical** proofs of $\vdash A \lor B$ s.t. neither $\vdash A$ nor $\vdash B$.

You may want something more fine-grained...

**Dialectica.**

(Gödel, 1958)
What is *Dialectica*?

A realizability translation (D) from HA into HA targets System T (simply-typed $\lambda$-calculus + integers + sequences) preserves intuitionistic content (**) but offers two additional semi-classical principles:

- $\exists n \in \mathbb{N} : P(n)$
- $\forall n \in \mathbb{N} : P(n) \rightarrow Q(m)$

*Markov's principle* *Independence of premise* (P decidable, I irrelevant)

Pierre-Marie Pédrot (PPS/$\pi r^2$)  A Materialist Dialectica  17/09/2015  14 / 44
What is *Dialectica*?

- A *realizability* translation \((-)^D\) from HA into HA\(^\omega\)
- Targets System T (simply-typed \(\lambda\)-calculus + integers + sequences)
- Preserves intuitionistic content (\(\forall + \exists\))
What is *Dialectica*?

- A realizability translation \((-)^D\) from HA into HA\(^\omega\)
- Targets System T (simply-typed \(\lambda\)-calculus + integers + sequences)
- Preserves intuitionistic content \((\forall + \exists)\)
- But offers two additional semi-classical principles:

\[
\text{MP} \quad \frac{\neg (\forall n \in \mathbb{N}. \neg P \, n)}{\exists n \in \mathbb{N}. \, P \, n}
\]

\[
\text{IP} \quad \frac{I \rightarrow \exists m \in \mathbb{N}. \, Q \, m}{\exists m \in \mathbb{N}. \, I \rightarrow Q \, m}
\]

« Markov’s principle »

« Independence of premise »

\((P \text{ decidable, } I \text{ irrelevant})\)
The Good Old Gödel’s translation

\[ \vdash A \quad \iff \quad \vdash A^D \equiv \exists \vec{u}. \forall \vec{x}. A_D[\vec{u}, \vec{x}] \]

\((-)_D \) essentially commutes with the connectives
The Good Old Gödel’s translation

\[ \vdash A \quad \mapsto \quad \vdash A^D \equiv \exists \vec{u}. \forall \vec{x}. A_D[\vec{u}, \vec{x}] \]

- \((\neg)^D\) essentially commutes with the connectives except for the arrow

\[
\begin{align*}
(A \land B)^D & \equiv \exists \vec{u}_A, \vec{v}_B. \forall \vec{x}_A, \vec{y}_B. A_D[\vec{u}_A, \vec{x}_A] \land B_D[\vec{v}_B, \vec{y}_B] \\
(A \lor B)^D & \equiv \exists \vec{u}_A, \vec{v}_B, b. \forall \vec{x}_A, \vec{y}_B. \left\{ \begin{array}{ll} A_D[\vec{u}_A, \vec{x}_A] & \text{if } b = 0 \\ B_D[\vec{v}_B, \vec{y}_B] & \text{if } b \neq 0 \end{array} \right. \\
(A \to B)^D & \equiv \exists \vec{f}, \vec{\varphi}. \forall \vec{u}_A, \vec{y}_B. A_D[\vec{u}_A, \vec{\varphi} \vec{u}_A \vec{y}_B] \to B_D[\vec{f} \vec{u}_A, \vec{y}_B]
\end{align*}
\]

1. If \(\vdash_H A\) then \(\vdash_H \omega \ A^D\).
2. MP and IP are provable through \((\neg)^D\).
Successes and failures

Dialectica has been used a lot

- Bar recursion (Spector ’62)
- Dialectica categories (De Paiva 89’)
- Proof mining (Oliva, Kohlenbach 2000’s)
Successes and failures

Dialectica has been used a lot

- Bar recursion (Spector ’62)
- Dialectica categories (De Paiva 89’)
- Proof mining (Oliva, Kohlenbach 2000’s)

Yet, a reputation of complexity and an aura of mystery.
In this Thesis

« *Dialectica* does not fit in the Curry-Howard paradigm. »

(U. Kohlenbach, TYPES 2013, 25th April 2013.)
In this Thesis

« Dialectica does not fit in the Curry-Howard paradigm. »
(U. Kohlenbach, TYPES 2013, 25th April 2013.)

New axiom

\[ \overline{\text{MP}} \]

Strong normalization

Logical encoding

Dialectica

Programming primitives

\[ \overline{\text{observe variable use}} \]

Program translation

CPS-like accumulating stacks
Contributions

- Reformulation of *Dialectica* as an untyped program translation
- Description of its computational content in a classical realizability style
- Extension to dependent type theories
Contributions

- Reformulation of *Dialectica* as an untyped program translation
- Description of its computational content in a classical realizability style
- Extension to dependent type theories
- Study of variants (call-by-value, classical-by-name, ...) (Ch. 10)
- Study of relationship with similar translations (Ch. 12)
- A more canonical call-by-need with control (Ch. 5)
Contributions

- Reformulation of *Dialectica* as an untyped program translation

- Description of its computational content in a classical realizability style

- Extension to dependent type theories

- Study of variants (call-by-value, classical-by-name, ...) (Ch. 10)
- Study of relationship with similar translations (Ch. 12)
- A more canonical call-by-need with control (Ch. 5)

- A lot of delightful Coq hacking! (not in the manuscript)
Contributions

Part II:
- Reformulation of *Dialectica* as an untyped program translation

Part III:
- Description of its computational content in a classical realizability style

Epilogue:
- Extension to dependent type theories
Part II.

« Reverse engineering Gödel's hacks »
A proof $u \vdash A$ is a term $\vdash u : W(A)$ such that $\forall x : C(A). u \perp A x$
Gödel used System T + sequences as a target.

\[
\begin{align*}
W(A) &\rightarrow W(B) \\
W(A) &\rightarrow C(B) \rightarrow C(A)
\end{align*}
\]

\[
\begin{align*}
A \rightarrow B \\
A \times B \\
A + B
\end{align*}
\]

\[
\begin{align*}
W(A) \times W(B) \\
W(A) \times W(B) \times \mathbb{B} \\
W(A) \times C(B)
\end{align*}
\]

\[
\begin{align*}
C(A) \times C(B) \\
C(A) \times C(B)
\end{align*}
\]
Gödel used System T + sequences as a target. We use true datatypes!

\[
\begin{array}{c}
\text{W} \\
A \rightarrow B \quad \left\{ \begin{array}{c}
\text{W}(A) \rightarrow \text{W}(B) \\
\text{W}(A) \rightarrow \text{C}(B) \rightarrow \text{C}(A)
\end{array} \right.
\end{array}
\text{C}
\begin{array}{c}
\text{W}(A) \times \text{C}(B) \\
A \times B \quad \left\{ \begin{array}{c}
\text{W}(A) \times \text{W}(B) \rightarrow \text{C}(A) \\
\text{W}(A) \times \text{W}(B) \rightarrow \text{C}(B)
\end{array} \right.
\end{array}
\begin{array}{c}
\text{W}(A) \rightarrow \text{C}(A) \\
\text{W}(B) \rightarrow \text{C}(B)
\end{array}
\begin{array}{c}
A + B \\
\text{W}(A) + \text{W}(B)
\end{array}
\]

Coming from a linear decomposition due to De Paiva and Hyland (1989). We restrict to propositional logic (for now).
A scrutiny into the term translation

\[ \Gamma \vdash t : A \quad \longrightarrow \quad \left\{ \begin{align*}
\mathcal{W}(\Gamma) & \vdash t^\bullet : \mathcal{W}(A) \\
\mathcal{W}(\Gamma) & \vdash t_{x_1} : \mathcal{C}(A) \rightarrow \mathcal{C}(\Gamma_1) \\
\quad \quad \quad \quad \quad \cdots \\
\mathcal{W}(\Gamma) & \vdash t_{x_n} : \mathcal{C}(A) \rightarrow \mathcal{C}(\Gamma_n)
\end{align*} \right\} \]

t is essentially \( t \), except \( (x : t)(x : t) \).

t depends on two families of terms.

\( \emptyset \) is used when weakening occurs (resp. duplication).
A scrutiny into the term translation

\[ \Gamma \vdash t : A \quad \rightarrow \quad \begin{cases} 
  W(\Gamma) \vdash t^\bullet : W(A) \\
  W(\Gamma) \vdash t_{x_1} : C(A) \rightarrow C(\Gamma_1) \\
  \cdots \\
  W(\Gamma) \vdash t_{x_n} : C(A) \rightarrow C(\Gamma_n) 
\end{cases} \]

- \( t^\bullet \) is essentially \( t \), except \((\lambda x. t)^\bullet \equiv (\lambda x. t^\bullet, \lambda x. \pi. t_x \pi)\)
- \( t_x \) depends on two families of terms

\[
\emptyset_A : C(A) \\
\@_A : C(A) \rightarrow C(A) \rightarrow W(A) \rightarrow C(A)
\]

s.t. \[ u \perp_A \pi_1 @^u_A \pi_2 \iff u \perp_A \pi_1 \land u \perp_A \pi_2 \]

- \( \emptyset \) (resp. \( \@ \)) is used when weakening occurs (resp. duplication)
Almost there

If $\vdash t : A$ then

1. $\vdash t^\bullet : \mathbf{W}(A)$.
2. For all $\pi : \mathcal{C}(A)$, $t^\bullet \perp_A \pi$.
Almost there

If $\vdash t : A$ then
1. $\vdash t^\bullet : W(A)$.
2. For all $\pi : C(A)$, $t^\bullet \perp_A \pi$.

There exists $t_1$ and $t_2$ s.t. $t_1 \equiv_\beta t_2$ but $t_1^\bullet \not\equiv_\beta t_2^\bullet$.

We would need equations that do not hold, such as

$$\pi \ @^t_A \emptyset_A \equiv_\beta \pi \equiv_\beta \emptyset_A \ @^t_A \pi$$

because they are defined in a quite ad-hoc, non parametric way.
The usual suspects

- As we have just seen, the problem appears because of @ and ∅.
- Actually already criticized because it requires decidability of ⊥.
- Solved by the Diller-Nahm variant using finite sets (1974)
- They are Gödel’s workarounds defined in a very hackish way
The usual suspects

- As we have just seen, the problem appears because of @ and ∅.
- Actually already criticized because it requires decidability of ⊥.
- Solved by the Diller-Nahm variant using finite sets (1974)
- They are Gödel’s workarounds defined in a very hackish way

We use a similar trick by using finite multisets.

The desired equations hold naturally and parametrically.
Revising the translation

Before

\[ \emptyset_A : \mathbb{C}(A) \]

@_A : \mathbb{C}(A) \to \mathbb{C}(A) \to \mathbb{W}(A) \to \mathbb{C}(A) \]

\[
\begin{cases} 
\mathbb{W}(A) \to \mathbb{W}(B) \\
\mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{C}(A)
\end{cases}
\]

\[
\begin{cases} 
\mathbb{W}(\Gamma) \vdash t^* : \mathbb{W}(A) \\
\mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_1) \\
\ldots \\
\mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_n)
\end{cases}
\]

\[ \perp \]

After

\[ \emptyset_A : \mathbb{M}\mathbb{C}(A) \]

@_A : \mathbb{M}\mathbb{C}(A) \to \mathbb{M}\mathbb{C}(A) \to \mathbb{M}\mathbb{C}(A) \]

\[
\begin{cases} 
\mathbb{W}(A) \to \mathbb{W}(B) \\
\mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{M}\mathbb{C}(A)
\end{cases}
\]

\[
\begin{cases} 
\mathbb{W}(\Gamma) \vdash t^* : \mathbb{W}(A) \\
\mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathbb{M}\mathbb{C}(\Gamma_1) \\
\ldots \\
\mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathbb{M}\mathbb{C}(\Gamma_n)
\end{cases}
\]

\[ \ldots \]
Finally!

\[
x^* \quad \equiv \quad x
\]
\[
(\lambda x. t)^* \quad \equiv \quad \begin{cases} 
\lambda x. t^* \\
\lambda x\pi. t_x\pi
\end{cases}
\]
\[
(tu)^* \quad \equiv \quad (\text{fst } t^*)u^*
\]
Finally!

\[x_x \equiv \lambda \pi. \{ \pi \}\]

\[x^\bullet \equiv x\]

\[(\lambda x. t)^\bullet \equiv \begin{cases} \lambda x. t^\bullet \\ \lambda x \pi. t_x \pi \end{cases}\]

\[(tu)^\bullet \equiv (\text{fst } t^\bullet) u^\bullet\]
Finally!

\[
\begin{align*}
x^\bullet & \equiv x \\
(\lambda x. t)^\bullet & \equiv \begin{cases} 
    \lambda x. t^\bullet \\
    \lambda x \pi. t_x \pi 
\end{cases} \\
(t u)^\bullet & \equiv (\text{fst } t^\bullet) u^\bullet \\
x_x & \equiv \lambda \pi. \{ \pi \} \\
y_x & \equiv \lambda \pi. \emptyset
\end{align*}
\]
Finally!

\[ x^\bullet \equiv x \]
\[ (\lambda x. t)^\bullet \equiv \begin{cases} \lambda x. t^\bullet \\ \lambda x \pi . t_x \pi \end{cases} \]
\[ (t u)^\bullet \equiv (\text{fst } t^\bullet) u^\bullet \]

\[ x_x \quad \equiv \quad \lambda \pi . \{ \pi \} \]
\[ y_x \quad \equiv \quad \lambda \pi . \emptyset \]
\[ (\lambda y. t)_x \quad \equiv \quad \lambda (y, \pi) . t_x \pi \]
Finally!

\[ x^* \equiv x \]
\[ (\lambda x. t)^* \equiv \begin{cases} \lambda x. t^* \\ \lambda x \pi. t_x \pi \end{cases} \]
\[ (tu)^* \equiv (\text{fst } t^*) u^* \]
\[ x_x \equiv \lambda \pi. \{ \pi \} \]
\[ y_x \equiv \lambda \pi. \emptyset \]
\[ (\lambda y. t)_x \equiv \lambda (y, \pi). t_x \pi \]
\[ (tu)_x \equiv \lambda \pi. \left( (\text{snd } t^*) u^* \pi \Rightarrow u_x \right) \]
\[ \begin{array}{c} \alpha \\ t_x (u^*, \pi) \end{array} \]
Finally!

\[ x^\bullet \equiv x \]
\[ (\lambda x. t)^\bullet \equiv \begin{cases} \lambda x. t^\bullet \\ \lambda x \pi . t_x \pi \end{cases} \]
\[ (t u)^\bullet \equiv (\text{fst } t^\bullet) u^\bullet \]
\[ x_x \equiv \lambda \pi . \{ \pi \} \]
\[ y_x \equiv \lambda \pi . \emptyset \]
\[ (\lambda y. t)_x \equiv \lambda (y, \pi). t_x \pi \]
\[ (t u)_x \equiv \lambda \pi . \left( (\text{snd } t^\bullet) u^\bullet \pi \gg u_x \right) \]
\[ t_x (u^\bullet, \pi) \]

1. If \( \vdash t : A \) then
   - \( \vdash t^\bullet : \mathcal{W}(A) \).
   - For all \( \pi : \mathcal{C}(A) \), \( t^\bullet \perp_A \pi \).

2. If \( t_1 \equiv_\beta t_2 \) then \( t_1^\bullet \equiv_\beta t_2^\bullet \).
A note on orthogonality

- The orthogonality can be expressed in this setting.
- Yet it is no more useful for now.
- The Gödel-style *Dialectica* used Friedmann’s trick for $\emptyset_0$:
  \[ W(0) \equiv 1 \quad \text{and} \quad C(0) \equiv 1 \]
- You needed to rule out invalid proofs of $W(0)$.
- Not the case anymore with multisets, $W(0) \equiv 0$.
- Soundness for free!
Part III.

« When Krivine meets Gödel »
You are here

New axiom    ~    Programming primitives

?            ?
⇔

Logical encoding

Revised *Dialectica*    ~    Something involving \{ multisets, counters, free variables \}

What is actually doing the translation as a program?
What rôle on earth have the counters?
New axiom
?
⇕
Logical encoding

Programming primitives
?
⇕
Program translation

Revised *Dialectica* ~ Something involving
\{ multisets, counters, free variables \}

What is actually doing the translation as a program?

What rôle on earth have the counters?
Stay on the scene

Let us introduce you to the Krivine Abstract Machine (KAM).

Closures \( c ::= (t, \sigma) \)

Environments \( \sigma ::= \emptyset | \sigma + (x := c) \)

Stacks \( \pi ::= \varepsilon | c \cdot \pi \)

Processes \( p ::= \langle c | \pi \rangle \)
Let us introduce you to the Krivine Abstract Machine (KAM).

Closures \( c ::= (t, \sigma) \)

Environments \( \sigma ::= \emptyset \mid \sigma + (x := c) \)

Stacks \( \pi ::= \varepsilon \mid c \cdot \pi \)

Processes \( p ::= \langle c \mid \pi \rangle \)

**Push**

\[ \langle (t \ u, \sigma) \mid \pi \rangle \rightarrow \langle (t, \sigma) \mid (u, \sigma) \cdot \pi \rangle \]
Stay on the scene

Let us introduce you to the Krivine Abstract Machine (KAM).

Closures \( c ::= (t, \sigma) \)

Environments \( \sigma ::= \emptyset \mid \sigma + (x := c) \)

Stacks \( \pi ::= \varepsilon \mid c \cdot \pi \)

Processes \( p ::= \langle c \mid \pi \rangle \)

POP \( \langle (\lambda x. t, \sigma) \mid c \cdot \pi \rangle \rightarrow \langle (t, \sigma + (x := c)) \mid \pi \rangle \)

The Krivine Abstract Machine™
Stay on the scene

Let us introduce you to the Krivine Abstract Machine (KAM).

Closures \( c ::= (t, \sigma) \)
Environments \( \sigma ::= \emptyset | \sigma + (x := c) \)
Stacks \( \pi ::= \varepsilon | c \cdot \pi \)
Processes \( p ::= \langle c | \pi \rangle \)

\textbf{GRAB} \quad \langle (x, \sigma + (x := c)) | \pi \rangle \rightarrow \langle c | \pi \rangle

The Krivine Abstract Machine™
Stay on the scene

Let us introduce you to the Krivine Abstract Machine (KAM).

Closures \( c \ ::= \ (t, \sigma) \)

Environments \( \sigma \ ::= \ \emptyset \ | \ \sigma + (x := c) \)

Stacks \( \pi \ ::= \ \varepsilon \ | \ c \cdot \pi \)

Processes \( p \ ::= \ \langle c \ | \ \pi \rangle \)

**Garbage** \( \langle (x, \sigma + (y := c)) \ | \ \pi \rangle \rightarrow \langle (x, \sigma) \ | \ \pi \rangle \)

The Krivine Abstract Machine™
Let us introduce you to the Krivine Abstract Machine (KAM).

<table>
<thead>
<tr>
<th>Closures</th>
<th>$c ::= (t, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environments</td>
<td>$\sigma ::= \emptyset</td>
</tr>
<tr>
<td>Stacks</td>
<td>$\pi ::= \varepsilon</td>
</tr>
<tr>
<td>Processes</td>
<td>$p ::= \langle c</td>
</tr>
</tbody>
</table>

**Push**

$$\langle (t \ u, \sigma) | \pi \rangle \rightarrow \langle (t, \sigma) | (u, \sigma) \cdot \pi \rangle$$

**Pop**

$$\langle (\lambda x. t, \sigma) | c \cdot \pi \rangle \rightarrow \langle (t, \sigma + (x := c)) | \pi \rangle$$

**Grab**

$$\langle (x, \sigma + (x := c)) | \pi \rangle \rightarrow \langle c | \pi \rangle$$

**Garbage**

$$\langle (x, \sigma + (y := c)) | \pi \rangle \rightarrow \langle (x, \sigma) | \pi \rangle$$

The Krivine Abstract Machine™
A word on the KAM

- The KAM is call-by-name, implementing linear head reduction
- Used amongst other things to do classical realizability (Krivine ’02)
- Features stacks and environments as first-class objects
- In particular, typing can be extended to stacks and environments

\[ \Gamma \vdash \pi : A^\perp \quad \sigma \vdash \Gamma \]

- Double-negation and forcing already explained through the KAM
  - Double-negation using \texttt{callcc} (Griffin ’90, Krivine ’03)
  - Forcing using a monotonous global variable (Miquel ’11)
What has been seen cannot be unseen

\[ C(A \to B) \equiv W(A) \times C(B) \]
What has been seen cannot be unseen

\[ \mathcal{C}(A \rightarrow B) \equiv \mathcal{W}(A) \times \mathcal{C}(B) \]
What has been seen cannot be unseen

\[ \mathcal{C}(A \to B) \equiv \mathcal{W}(A) \times \mathcal{C}(B) \]
What has been seen cannot be unseen

\[ \text{C}(A \rightarrow B) \equiv \text{W}(A) \times \text{C}(B) \]
What has been seen cannot be unseen

\[ \mathbb{C}(A \to B) \equiv \mathbb{W}(A) \times \mathbb{C}(B) \]

\[ \langle (\lambda x. t, \sigma) \mid c \cdot \pi \rangle \to \langle (t, \sigma + (x := c)) \mid \pi \rangle \]
What has been seen cannot be unseen

\[ \mathcal{C}(A \rightarrow B) \equiv \mathbb{W}(A) \times \mathcal{C}(B) \]

\[ \langle (\lambda x. t, \sigma) | c \cdot \pi \rangle \rightarrow \langle (t, \sigma + (x := c)) | \pi \rangle \]

Counters are stacks!
What has been seen cannot be unseen

\[ C(A \rightarrow B) \equiv W(A) \times C(B) \]

\[ \langle (\lambda x. t, \sigma) \mid c \cdot \pi \rangle \rightarrow \langle (t, \sigma + (x := c)) \mid \pi \rangle \]

Counters are stacks!

*Dialectica* gives access to stacks!
Closures all the way down

Let:
- a term \( \bar{x} : \Gamma \vdash t : A \)
- a closure \( \sigma \vdash \Gamma \)
- a stack \( \vdash \pi : A^\perp \) (i.e. \( \pi^* : \mathcal{C}(A) \))
Closures all the way down

Let:
- a term $\vec{x} : \Gamma \vdash t : A$
- a closure $\sigma \vdash \Gamma$
- a stack $\vdash \pi : A^\perp$ (i.e. $\pi : \mathbb{C}(A)$)

Then

$$(t_{\vec{x}i}\{\vec{x} := \sigma\}) \pi : \mathcal{M}(\mathbb{C}(\Gamma_i) \equiv \{\rho_1; \ldots; \rho_m\}$$

are the stacks encountered by $x_i$ while evaluating $\langle (t, \sigma) \mid \pi \rangle$, i.e.

$$
\langle (t, \sigma) \mid \pi \rangle \rightarrow^* \langle (x_i, \sigma_1) \mid \rho_1 \rangle \\
\vdots \quad \vdots \\
\rightarrow^* \langle (x_i, \sigma_m) \mid \rho_m \rangle
$$

The $(-)_x$ translation tracks the uses of $x$ as delimited continuations.
Look around you

\[ x_x \equiv \lambda \pi. \{\pi\} \]
\[ y_x \equiv \lambda \pi. \emptyset \]
\[ (\lambda y. t)_x \equiv \lambda (y, \pi). t_x \pi \]
\[ (t u)_x \equiv \lambda \pi. \left( \begin{array}{l}
(\text{snd} \ t^\bullet) \ u^\bullet \ \pi \trianglerighteq u_x \\
\@ \\
t_x (u^\bullet, \pi)
\end{array} \right) \]
A little issue

The stacks produced by the KAM are ordered by sequentiality.

The *Dialectica* produces multisets of stacks without regard to order.
A little issue

The stacks produced by the KAM are ordered by sequentiality.

The *Dialectica* produces multisets of stacks without regard to order.

Not possible to fix easily (or at all?)
A defect of linear logic?
Revisiting MP + IP

In the historical presentation:

- IP essentially obtained by $\emptyset +$ realizability
- MP is more magical

$$\mathcal{W}(\text{MP}) \cong \mathbb{N} \to \mathbb{N}$$

$$\text{mp} := \lambda x. x$$
Revisiting MP + IP

In the historical presentation:

- IP essentially obtained by $\emptyset +$ realizability
- MP is more magical

$$W(\text{MP}) \cong \mathbb{N} \to \mathbb{N}$$
$$\text{mp} := \lambda x. x$$

In the revised *Dialectica*:

- Not a realizability and no $\emptyset \rightsquigarrow$ no IP
- The historical realizer of the MP takes another flavour
First issue: \( \mathbf{W}(\neg A) \cong \neg \mathbf{W}(A) \)

Need to weaken \( \neg A \) into \( \sim A \equiv A \rightarrow \bot \) where

\[
\mathbf{W}(\bot) \equiv 1 \quad \mathbf{C}(\bot) \equiv 1 \quad (\not\mathbf{\bot} (\not\mathbf{\bot}) \quad (\text{no rule for } \bot)
\]
1 First issue: \( \mathsf{W}(\neg A) \cong \neg \mathsf{W}(A) \)

Need to weaken \( \neg A \) into \( \sim A \equiv A \rightarrow \bot \) where
\[
\mathsf{W}(\bot) \equiv 1 \quad \mathsf{C}(\bot) \equiv 1 \quad (\_ \not\equiv \bot) \quad \text{(no rule for } \bot)\]

2 Second issue: \( \mathsf{W}(\sim \sim A) \cong (\mathsf{W}(A) \rightarrow \mathsf{M C}(A)) \rightarrow \mathsf{M W}(A) \)

You need the orthogonality to ensure that the returned multiset is **classically** not empty!

\[
f \vdash \sim \sim A \leftrightarrow \\
\forall \varphi : \mathsf{W}(A) \rightarrow \mathsf{M C}(A). \neg (\forall u : \mathsf{W}(A) \in f \varphi. \neg (\forall \pi : \mathsf{C}(A) \in \varphi \ u. \ u \bot_A \pi))
\]
First issue: \( \mathsf{W}(\neg A) \cong \neg \mathsf{W}(A) \)

Need to weaken \( \neg A \) into \( \sim A \equiv A \rightarrow \bot \) where
\[
\mathsf{W}(\bot) \equiv 1 \quad \mathsf{C}(\bot) \equiv 1 \quad (\not\not) \not\bot (\not\not) \quad \text{(no rule for } \bot)\]

Second issue: \( \mathsf{W}(\sim \sim A) \cong (\mathsf{W}(A) \rightarrow \mathcal{M} \mathsf{C}(A)) \rightarrow \mathcal{M} \mathsf{W}(A) \)

You need the orthogonality to ensure that the returned multiset is \textbf{classically} not empty!

\[
f \vdash \sim \sim A \iff \\
\forall \varphi : \mathsf{W}(A) \rightarrow \mathcal{M} \mathsf{C}(A). \neg (\forall u : \mathsf{W}(A) \in f \varphi. \neg (\forall \pi : \mathsf{C}(A) \in \varphi \ u. \ u \perp_A \pi))
\]

If \( A \) has a decidable orthogonality then this is equivalent to
\[
\forall \varphi : \mathsf{W}(A) \rightarrow \mathcal{M} \mathsf{C}(A). \exists u : \mathsf{W}(A) \in f \varphi. \forall \pi : \mathsf{C}(A) \in \varphi \ u. \ u \perp_A \pi
\]
1. First issue: $\mathbb{W}(\neg A) \cong \neg \mathbb{W}(A)$

Need to weaken $\neg A$ into $\sim A \equiv A \rightarrow \bot$ where

$\mathbb{W}(\bot) \equiv 1 \quad \mathbb{C}(\bot) \equiv 1 \quad (\emptyset \not\in \bot) \quad ($no rule for $\bot$)$

2. Second issue: $\mathbb{W}(\sim\sim A) \cong (\mathbb{W}(A) \rightarrow \mathcal{M} \mathbb{C}(A)) \rightarrow \mathcal{M} \mathbb{W}(A)$

You need the orthogonality to ensure that the returned multiset is **classically** not empty!

$$f \vdash \sim \sim A \iff$$

$$\forall \varphi : \mathbb{W}(A) \rightarrow \mathcal{M} \mathbb{C}(A). \neg (\forall u : \mathbb{W}(A) \in f \varphi. \neg (\forall \pi : \mathbb{C}(A) \in \varphi \ u. u \perp_A \pi))$$

3. If $A$ has a decidable orthogonality then this is equivalent to

$$\forall \varphi : \mathbb{W}(A) \rightarrow \mathcal{M} \mathbb{C}(A). \exists u : \mathbb{W}(A) \in f \varphi. \forall \pi : \mathbb{C}(A) \in \varphi \ u. u \perp_A \pi$$

4. Extract using $\varphi \equiv \lambda_. \emptyset$ and enjoy (if $A$ is first-order).

$$\exists u : \mathbb{W}(A) \in f \varphi$$
Comparison with historical *Dialectica*

- We’ve got rid of IP: it’s not a bug, it’s a feature!
- MP has a clearer meaning now.
  - The delimited continuation part extracts argument access to functions
  - We made explicit a crawling over finite multisets that were hard-wired into `@` in the historical version
- In particular we may do fancier things now
  - Counting the number of accesses of a function to a variable
  - More ...?
Epilogue.

« You're not done yet. »
Dependently-typed *Dialectica*

- This translation naturally lifts to dependent types

\[ W(\Pi x : A. B) \equiv \Pi x : W(A).W(B) \times (C(B) \to mW(A)) \]

\[ C(\Pi x : A. B) \equiv \Sigma x : W(A).C(B) \]

- What about dependent elimination?
  - Hints in this thesis
  - Actually solved after the manuscript was submitted (kudos to Andrej)
  - Essentially make \( C(A) \) highly depend on some value \( u : W(A) \)
  - Decomposes through CBPV (Levy ’01)

- Difficult to implement in practice?
  - Computational implementation of multisets?
  - We should be using HITs (HoTT Book)
  - TODO: Write an implementation of *Dialectica* in Coq
Intuitionism is the new linear

- *Dialectica* is the prototypical model of LL
  - « Double-glueing, le mot est lâché ! »
- Towards a « free model » of LL in LJ?
  - « A new linear logic: intuitionistic logic »
- Delimited continuations and dependency probably needed
- Linear logic is not about linearity
Intuitionistic enough translations

- *Dialectica* shares common properties with (intuitionistic) forcing. In particular, $\mathbb{W}(-)$ commutes with positives.
Dialectica shares common properties with (intuitionistic) forcing. In particular, $\mathcal{W}(-)$ commutes with positives. This is a moral requirement to preserve dependent elimination.
Intuitionistic enough translations

- *Dialectica* shares common properties with (intuitionistic) forcing. In particular, $\mathcal{W}(\_)$ commutes with positives.
- This is a moral requirement to preserve dependent elimination.
- Occurs strikingly often with commutative / idempotent monads.
  - Forcing (a monotonous reader monad on $\langle \mathbb{P}, \leq \rangle$)
  - Intuitionistic CPS (similar but with stacks)
  - Dialectica (a rich writer on $\mathcal{C}(A) \to \mathcal{M} \mathcal{C}(\Gamma)$)
  - Sheafification (?)
- Probably something deep there
Conclusion

- We demystified Gödel’s Dialectica translation.
- Actually using concepts inexistents at the time of Gödel:
  - Computational content of proofs
  - Stacks
  - Explicit substitutions
- We described computationally the contents of Markov’s principle.
- This presentation allows for a lot of extensions and future work.
Thanks for your attention.