A Materialist Dialectica

Pierre-Marie Pédrot

PPS/πr^2

17th September 2015

Part I.

« How Gödel became a computer scientist out of remorse »

Logic?

« The LOGICIST approach »

From AXIOMS, applying valid RULES, derive a CONCLUSION.

« 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.

$$\frac{\vdash A \to B \qquad \vdash B \to C}{\vdash A \to C}$$

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

Logic?

« 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.

$$\frac{\vdash A \to B \qquad \vdash B \to C}{\vdash A \to C}$$

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.

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.

- **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 $\mathcal{S}$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable

1931: Gödel's incompleteness theorem

Assume a set of rules $\mathcal{S}$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable

then

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

1931: Gödel's incompleteness theorem

Assume a set of rules $\mathcal{S}$ which is

1. Expressive enough
2. Consistent
3. Mechanically checkable

then

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

Quis ipsos custodiet custodes?

Logic: Rise of Computer Science I

« 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

$A \vee \neg A$	$\neg\neg A \rightarrow A$	$((A \rightarrow B) \rightarrow A) \rightarrow A$
Excluded Middle	Reductio ad Absurdum	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)

Logic: Rise of Computer Science II

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)

Logic: Rise of Computer Science II

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)

Curry-Howard isomorphism (1960's)

Intuitionistic proofs *are* λ-calculus programs

The beloved λ-calculus

The niftiest programming language of them all!

Terms	$t ::= x \mid \lambda x.\, t \mid t\ u \mid \ldots$
Types	$A ::= \alpha \mid A \to B \mid \ldots$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash B}{\Gamma \vdash \lambda x.\, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B}$$

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

Type derivations are proofs, compatible with β-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.

The mathematician

"This logic is crappy, it does not feature the following principles I am acquainted with."

Either A or not A hold.
Every bounded monotone sequence has a limit.
Two sets with same elements are equal.
Every formula is equivalent to its prenex form.
…

The computer scientist

"This language is crappy, it does not feature the following structures I am acquainted with."

```
printf("Hello world")
x <- 42
while true { … }
goto #considered_harmful
fork()
…
```

Which logic for which programs?

Standard members of each community will complain.

The mathematician

"This logic is crappy, it does not feature the following principles I am acquainted with."

Either A or not A hold.
Every bounded monotone sequence has a limit.
Two sets with same elements are equal.
Every formula is equivalent to its prenex form.
...

The computer scientist

"This language is crappy, it does not feature the following structures I am acquainted with."

```
printf("Hello world")
x <- 42
while true { … }
goto #considered_harmful
fork()
…
```

INTUITIONISTIC LOGIC **FUNCTIONAL LANGUAGE**

What Curry-Howard takes, it gives back.

A new hope

What Curry-Howard takes, it gives back.

New axioms	$\sim$	Programming primitives
$\Updownarrow$		$\Updownarrow$
Logical encoding	$\sim$	Program translation

The prototypical example: Classical logic

We can implement classical logic through this scheme.

New axiom		Programming primitives
$((A \to B) \to A) \to A$	$\sim$	`callcc`
Used by your next-door mathematician		From Scheme, though a bit arcane
$\Updownarrow$		$\Updownarrow$
Logical encoding		Program translation
double-negation translation	$\sim$	continuation-passing style
Gödel already did that by the 30's		The dullest JS programmer uses this

Classical logic as a logical translation

Gödel-like presentation

Translation from classical logic into intuitionistic logic.

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

$$\begin{array}{lcl} A^+ & \equiv & A^- \to \bot \\ \alpha^- & \equiv & \alpha \to \bot \\ (A \to B)^- & \equiv & A^+ \times B^- \\ & \ldots & \end{array}$$

1. If $\vdash \quad A$ then $\vdash \quad A^+$.
2. There is a proof of $\vdash (((A \to B) \to A) \to A)^+$.

Curry-Howard presentation

Translation from λ-calculus + cc into λ-calculus.

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

$$\begin{array}{lcl}
A^+ & \equiv & A^- \to \bot \\
\alpha^- & \equiv & \alpha \to \bot \\
(A \to B)^- & \equiv & A^+ \times B^- \\
& \dots &
\end{array}
\qquad
\begin{array}{lcl}
x^\bullet & \equiv & \lambda \omega.\, x\, \omega \\
(\lambda x.\, t)^\bullet & \equiv & \lambda (x, \omega).t^\bullet\ \omega \\
(t\ u)^\bullet & \equiv & \lambda \omega.\, t^\bullet (u^\bullet, \omega) \\
& \dots &
\end{array}$$

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

More is less

You lose part of your soul in the encoding.

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

There are **classical** proofs of $\vdash A \vee B$ 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 $\vdash A \vee B$ then $\vdash A$ or $\vdash B$.

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

You may want something more fine-grained...

Dialectica.

(Gödel, 1958)

What is *Dialectica*?

What is *Dialectica*?

- A *realizability* translation $(-)^D$ from HA into HA$^\omega$
- Targets System T (simply-typed λ-calculus + integers + sequences)
- Preserves intuitionistic content ($\vee + \exists$)

What is *Dialectica*?

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

$$\text{MP}\ \frac{\neg(\forall n \in \mathbb{N}.\, \neg P\, n)}{\exists n \in \mathbb{N}.\, P\, n} \qquad\qquad \frac{I \to \exists m \in \mathbb{N}.\, Q\, m}{\exists m \in \mathbb{N}.\, I \to Q\, m}\ \text{IP}$$

« Markov's principle » « Independence of premise »

(P decidable, I irrelevant)

The Good Old Gödel's translation

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

- $(-)_D$ essentially commutes with the connectives

The Good Old Gödel's translation

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

- $(-)_D$ essentially commutes with the connectivesexcept for the arrow

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

1. If $\vdash_{HA} A$ then $\vdash_{HA^\omega} A^D$.
2. MP and IP are provable through $(-)^D$.

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.)

New axiom	$\sim$	~~Programming primitives~~
MP + IP		
$\Updownarrow$		
Logical encoding	$\sim$	~~Program translation~~
Dialectica		

In this Thesis

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

~~(U. Kohlenbach, TYPES 2013, 25th April 2013.)~~

New axiom		Programming primitives
$\widetilde{\mathsf{MP}}$	$\sim$	observe variable use
$\Updownarrow$		$\Updownarrow$
Logical encoding		Program translation
Dialectica	$\sim$	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 »

Types, types, TYPES! (De Paiva, '89)

$$\exists \vec{u}.\, \forall \vec{x}.\, A_D[\vec{u}, \vec{x}]$$

$\mathbb{W}(A)$	$\mathbb{C}(A)$	$\perp_A$
`Type`	`Type`	$\mathbb{W}(A) \to \mathbb{C}(A) \to$ `Prop`
Witness	Counter	Orthogonality

A proof $u \Vdash A$ is a term $\vdash u : \mathbb{W}(A)$ such that $\forall x : \mathbb{C}(A).\, u \perp_A x$

Cleaning up

Gödel used System T + sequences as a target.

	$\mathbb{W}$	$\mathbb{C}$
$A \to B$	$\begin{cases} \mathbb{W}(A) \to \mathbb{W}(B) \\ \mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{C}(A) \end{cases}$	$\mathbb{W}(A) \times \mathbb{C}(B)$
$A \times B$	$\mathbb{W}(A) \times \mathbb{W}(B)$	$\mathbb{C}(A) \times \mathbb{C}(B)$
$A + B$	$\mathbb{W}(A) \times \mathbb{W}(B) \times \mathbb{B}$	$\mathbb{C}(A) \times \mathbb{C}(B)$

Gödel used System T + sequences as a target.
We use true datatypes!

	$\mathbb{W}$	$\mathbb{C}$
$A \to B$	$\begin{cases} \mathbb{W}(A) \to \mathbb{W}(B) \\ \mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{C}(A) \end{cases}$	$\mathbb{W}(A) \times \mathbb{C}(B)$
$A \times B$	$\mathbb{W}(A) \times \mathbb{W}(B)$	$\begin{cases} \mathbb{W}(A) \times \mathbb{W}(B) \to \mathbb{C}(A) \\ \mathbb{W}(A) \times \mathbb{W}(B) \to \mathbb{C}(B) \end{cases}$
$A + B$	$\mathbb{W}(A) + \mathbb{W}(B)$	$\begin{cases} \mathbb{W}(A) \to \mathbb{C}(A) \\ \mathbb{W}(B) \to \mathbb{C}(B) \end{cases}$

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

$$\Gamma \vdash t : A \qquad \longrightarrow \qquad \left\{ \begin{array}{l} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_1) \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_n) \end{array} \right.$$

A scrutiny into the term translation

$$\Gamma \vdash t : A \qquad \longrightarrow \qquad \begin{cases} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_1) \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_n) \end{cases}$$

- $t^\bullet$ 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

$$\varnothing_A : \mathbb{C}(A)$$

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

$$\text{s.t.} \qquad u \perp_A \pi_1 \ @_A^u\ \pi_2 \qquad \leftrightarrow \qquad u \perp_A \pi_1 \wedge u \perp_A \pi_2$$

- $\varnothing$ (resp. $@$) is used when weakening occurs (resp. duplication)

Almost there

If $\vdash t : A$ then

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

Almost there

If $\vdash t : A$ then

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

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

$$\pi \ @^t_A \ \varnothing_A \quad \equiv_\beta \quad \pi \quad \equiv_\beta \quad \varnothing_A \ @^t_A \ \pi$$

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 $\varnothing$.
- Actually already criticized because it requires decidability of $\perp$.
- 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 $\varnothing$.
- Actually already criticized because it requires decidability of $\perp$.
- 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 **multi**sets.

The desired equations hold naturally and parametrically.

Revising the translation

Before	After
$\varnothing_A : \mathbb{C}(A)$	$\varnothing_A : \boxed{\mathfrak{M}\,\mathbb{C}(A)}$
$@_A : \mathbb{C}(A) \to \mathbb{C}(A) \to \mathbb{W}(A) \to \mathbb{C}(A)$	$@_A : \boxed{\mathfrak{M}\,\mathbb{C}(A)} \to \boxed{\mathfrak{M}\,\mathbb{C}(A)} \to \boxed{\mathfrak{M}\,\mathbb{C}(A)}$
$\left\{\begin{array}{c} \mathbb{W}(A) \to \mathbb{W}(B) \\ \mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{C}(A) \end{array}\right.$	$\left\{\begin{array}{c} \mathbb{W}(A) \to \mathbb{W}(B) \\ \mathbb{W}(A) \to \mathbb{C}(B) \to \boxed{\mathfrak{M}\,\mathbb{C}(A)} \end{array}\right.$
$\left\{\begin{array}{l} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_1) \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \mathbb{C}(\Gamma_n) \end{array}\right.$	$\left\{\begin{array}{l} \mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A) \\ \mathbb{W}(\Gamma) \vdash t_{x_1} : \mathbb{C}(A) \to \boxed{\mathfrak{M}\,\mathbb{C}(\Gamma_1)} \\ \ldots \\ \mathbb{W}(\Gamma) \vdash t_{x_n} : \mathbb{C}(A) \to \boxed{\mathfrak{M}\,\mathbb{C}(\Gamma_n)} \end{array}\right.$
$\bot$	$\ldots$

Finally!

$$
\begin{array}{lcl}
x^\bullet & \equiv & x \\
(\lambda x.\, t)^\bullet & \equiv & \begin{cases} \lambda x.\, t^\bullet \\ \lambda x\, \pi.\, t_x\, \pi \end{cases} \\
(t\, u)^\bullet & \equiv & (\texttt{fst}\ t^\bullet)\, u^\bullet
\end{array}
$$

Finally!

$$x_x \equiv \lambda\pi.\, \{\pi\}$$

$$\begin{array}{lcl} x^{\bullet} & \equiv & x \\ (\lambda x.\, t)^{\bullet} & \equiv & \begin{cases} \lambda x.\, t^{\bullet} \\ \lambda x\, \pi.\, t_x\, \pi \end{cases} \\ (t\, u)^{\bullet} & \equiv & (\texttt{fst}\ t^{\bullet})\, u^{\bullet} \end{array}$$

Finally!

$$
\begin{array}{lcl}
x^{\bullet} & \equiv & x \\
(\lambda x.\, t)^{\bullet} & \equiv & \begin{cases} \lambda x.\, t^{\bullet} \\ \lambda x\, \pi.\, t_x\, \pi \end{cases} \\
(t\, u)^{\bullet} & \equiv & (\mathtt{fst}\ t^{\bullet})\, u^{\bullet}
\end{array}
\qquad
\begin{array}{lcl}
x_x & \equiv & \lambda \pi.\, \{\pi\} \\
y_x & \equiv & \lambda \pi.\, \varnothing
\end{array}
$$

Finally!

$$\begin{array}{lcl} x^\bullet & \equiv & x \\ (\lambda x.\, t)^\bullet & \equiv & \begin{cases} \lambda x.\, t^\bullet \\ \lambda x \pi.\, t_x\, \pi \end{cases} \\ (t\, u)^\bullet & \equiv & (\texttt{fst}\ t^\bullet)\, u^\bullet \end{array}$$

$$\begin{array}{lcl} x_x & \equiv & \lambda \pi.\, \{\pi\} \\ y_x & \equiv & \lambda \pi.\, \varnothing \\ (\lambda y.\, t)_x & \equiv & \lambda (y, \pi).\, t_x\, \pi \end{array}$$

Finally!

$$
\begin{aligned}
x^\bullet &\equiv x \\
(\lambda x.\, t)^\bullet &\equiv \begin{cases} \lambda x.\, t^\bullet \\ \lambda x\,\pi.\, t_x\,\pi \end{cases} \\
(t\,u)^\bullet &\equiv (\texttt{fst}\ t^\bullet)\, u^\bullet
\end{aligned}
$$

$$
\begin{aligned}
x_x &\equiv \lambda\pi.\, \{\pi\} \\
y_x &\equiv \lambda\pi.\, \varnothing \\
(\lambda y.\, t)_x &\equiv \lambda(y,\pi).\, t_x\,\pi \\
(t\,u)_x &\equiv \lambda\pi.\, \begin{pmatrix} (\texttt{snd}\ t^\bullet)\, u^\bullet\,\pi \gg\!= u_x \\ @ \\ t_x\,(u^\bullet,\pi) \end{pmatrix}
\end{aligned}
$$

Finally!

$$x^\bullet \equiv x$$

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

$$(t\, u)^\bullet \equiv (\mathtt{fst}\ t^\bullet)\ u^\bullet$$

$$x_x \equiv \lambda \pi.\, \{\pi\}$$

$$y_x \equiv \lambda \pi.\, \varnothing$$

$$(\lambda y.\, t)_x \equiv \lambda (y, \pi).\, t_x\, \pi$$

$$(t\, u)_x \equiv \lambda \pi. \begin{pmatrix} (\mathtt{snd}\ t^\bullet)\ u^\bullet\ \pi \ggg u_x \\ @ \\ t_x\, (u^\bullet, \pi) \end{pmatrix}$$

1. If $\vdash t : A$ then
   - $\vdash t^\bullet : \mathbb{W}(A)$.
   - For all $\pi : \mathbb{C}(A)$, $t^\bullet \perp_A \pi$.
2. If $t_1 \equiv_\beta t_2$ then $t_1{}^\bullet \equiv_\beta t_2{}^\bullet$

A note on orthogonality

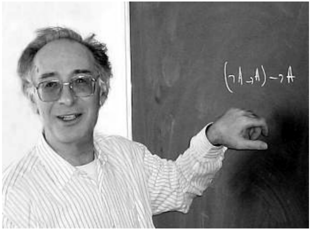
- The orthogonality can be expressed in this setting.
- Yet it is no more useful for now.
- The Gödel-style *Dialectica* used Friedmann's trick for $\varnothing_0$:

$$\mathbb{W}(0) \equiv 1 \quad \text{and} \quad \mathbb{C}(0) \equiv 1$$

- You needed to rule out invalid proofs of $\mathbb{W}(0)$.
- Not the case anymore with multisets, $\mathbb{W}(0) \equiv 0$.
- Soundness for free!

Part III.

« When Krivine meets Gödel »



You are here

New axiom	$\sim$	Programming primitives
?		**?**
$\Updownarrow$		$\Updownarrow$
Logical encoding		Program translation
Revised *Dialectica*	$\sim$	Something involving $\left\{\begin{array}{l}\text{multisets}\\ \text{counters}\\ \text{free variables}\end{array}\right.$

You are here

$$\begin{array}{ccc}
\text{New axiom} & \sim & \text{Programming primitives} \\
? & & ? \\
\Updownarrow & & \Updownarrow \\
\text{Logical encoding} & & \text{Program translation}
\end{array}$$

$$\text{Revised } \mathcal{D}ialectica \quad \sim \quad \text{Something involving} \begin{cases} \text{multisets} \\ \text{counters} \\ \text{free variables} \end{cases}$$

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).

$$
\begin{array}{llll}
\text{Closures} & c & ::= & (t, \sigma) \\
\text{Environments} & \sigma & ::= & \emptyset \mid \sigma + (x := c) \\
\text{Stacks} & \pi & ::= & \varepsilon \mid c \cdot \pi \\
\text{Processes} & p & ::= & \langle c \mid \pi \rangle
\end{array}
$$

The Krivine Abstract Machine™

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

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

$$\textsc{Push} \qquad \langle (t\,u, \sigma) \mid \pi \rangle \quad \rightarrow \quad \langle (t, \sigma) \mid (u, \sigma) \cdot \pi \rangle$$

The Krivine Abstract Machine™

Stay on the scene

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

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

$$\textsc{Pop} \qquad \langle (\lambda x.\, t, \sigma) \mid c \cdot \pi \rangle \quad \to \quad \langle (t, \sigma + (x := c)) \mid \pi \rangle$$

The Krivine Abstract Machine™

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

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

$$\textsc{Grab} \qquad \langle (x, \sigma + (x := c)) \mid \pi \rangle \quad \to \quad \langle c \mid \pi \rangle$$

The Krivine Abstract Machine™

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

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

$$\textsc{Garbage} \quad \langle (x, \sigma + (y := c)) \mid \pi \rangle \quad \rightarrow \quad \langle (x, \sigma) \mid \pi \rangle$$

The Krivine Abstract Machine™

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

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

Push	$\langle (t\,u, \sigma) \mid \pi \rangle$	$\rightarrow$	$\langle (t, \sigma) \mid (u, \sigma) \cdot \pi \rangle$
Pop	$\langle (\lambda x.\, t, \sigma) \mid c \cdot \pi \rangle$	$\rightarrow$	$\langle (t, \sigma + (x := c)) \mid \pi \rangle$
Grab	$\langle (x, \sigma + (x := c)) \mid \pi \rangle$	$\rightarrow$	$\langle c \mid \pi \rangle$
Garbage	$\langle (x, \sigma + (y := c)) \mid \pi \rangle$	$\rightarrow$	$\langle (x, \sigma) \mid \pi \rangle$

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

$$\vdash \pi : A^{\perp} \qquad \sigma \vdash \Gamma$$

- Double-negation and forcing already explained through the KAM
  - Double-negation using `callcc` (Griffin '90, Krivine '03)
  - Forcing using a monotonous global variable (Miquel '11)

What has been seen cannot be unseen

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

What has been seen cannot be unseen

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

What has been seen cannot be unseen

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

What has been seen cannot be unseen

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

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

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

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

Counters are stacks!

What has been seen cannot be unseen

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

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

Counters are stacks!

Dialectica gives access to stacks!

Let:

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

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. $\pi^{\bullet} : \mathbb{C}(A)$)

Then

$$(t_{x_i}\{\vec{x} := \sigma^{\bullet}\})\,\pi^{\bullet} : \mathfrak{M}\,\mathbb{C}(\Gamma_i) \equiv \{\rho_1; \ldots; \rho_m\}$$

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

$$\begin{array}{ccc}
\langle (t, \sigma) \mid \pi \rangle & \longrightarrow^{*} & \langle (x_i, \sigma_1) \mid \rho_1 \rangle \\
& \vdots & \vdots \\
& \longrightarrow^{*} & \langle (x_i, \sigma_m) \mid \rho_m \rangle
\end{array}$$

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

Look around you

$$
\begin{array}{rcl}
x_x & \equiv & \lambda\pi.\,\{\pi\} \\
y_x & \equiv & \lambda\pi.\,\varnothing \\
(\lambda y.\,t)_x & \equiv & \lambda(y,\pi).\,t_x\,\pi
\end{array}
$$

$$
(t\,u)_x \quad \equiv \quad \lambda\pi.\,\begin{pmatrix} (\mathtt{snd}\ t^\bullet)\,u^\bullet\,\pi \ggg\!= u_x \\ @ \\ t_x\,(u^\bullet,\pi) \end{pmatrix}
$$

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 $\varnothing$ + realizability
- MP is more magical

$$\begin{array}{lcl} \mathbb{W}(\mathtt{MP}) & \cong & \mathbb{N} \to \mathbb{N} \\ \mathtt{mp} & := & \lambda x.\, x \end{array}$$

Revisiting MP + IP

In the historical presentation:

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

$$\begin{array}{lcl} \mathbb{W}(\mathtt{MP}) & \cong & \mathbb{N} \to \mathbb{N} \\ \mathtt{mp} & := & \lambda x.\, x \end{array}$$

In the revised *Dialectica*:

- Not a realizability and no $\varnothing$ $\rightsquigarrow$ no IP
- The historical realizer of the MP takes another flavour

MP?

1. First issue: $\mathbb{W}(\neg A) \cong \neg\mathbb{W}(A)$

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

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

MP?

1. First issue: $\mathbb{W}(\neg A) \cong \neg \mathbb{W}(A)$

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

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

2. Second issue: $\mathbb{W}(\sim\sim A) \cong (\mathbb{W}(A) \to \mathfrak{M}\,\mathbb{C}(A)) \to \mathfrak{M}\,\mathbb{W}(A)$

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

   $$f \Vdash \sim\sim A \leftrightarrow$$
   $$\forall \varphi : \mathbb{W}(A) \to \mathfrak{M}\,\mathbb{C}(A).\, \neg(\forall u : \mathbb{W}(A) \in f\,\varphi.\, \neg(\forall \pi : \mathbb{C}(A) \in \varphi\ u.\ u \perp_A \pi))$$

MP?

1. First issue: $\mathbb{W}(\neg A) \cong \neg\mathbb{W}(A)$

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

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

2. Second issue: $\mathbb{W}(\sim\sim A) \cong (\mathbb{W}(A) \to \mathfrak{M}\,\mathbb{C}(A)) \to \mathfrak{M}\,\mathbb{W}(A)$

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

$$f \Vdash \sim\sim A \leftrightarrow$$

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

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

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

MP?

1. First issue: $\mathbb{W}(\neg A) \cong \neg\mathbb{W}(A)$

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

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

2. Second issue: $\mathbb{W}(\sim\sim A) \cong (\mathbb{W}(A) \to \mathfrak{M}\,\mathbb{C}(A)) \to \mathfrak{M}\,\mathbb{W}(A)$

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

$$f \Vdash \sim\sim A \leftrightarrow$$

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

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

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

4. Extract using $\varphi \equiv \lambda\_.\,\varnothing$ 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

$$\begin{array}{rcl} \mathbb{W}(\Pi x : A.\, B) & \equiv & \Pi x : \mathbb{W}(A).\mathbb{W}(B) \times (\mathbb{C}(B) \to \mathfrak{M}\,\mathbb{W}(A)) \\ \mathbb{C}(\Pi x : A.\, B) & \equiv & \Sigma x : \mathbb{W}(A).\mathbb{C}(B) \end{array}$$

- What about dependent elimination?
  - Hints in this thesis
  - Actually solved after the manuscript was submitted (kudos to Andrej)
  - Essentially make $\mathbb{C}(A)$ highly depend on some value $u : \mathbb{W}(A)$
  - 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.

Intuitionistic enough translations

- *Dialectica* shares common properties with (intuitionistic) forcing.

  In particular, $\mathbb{W}(-)$ commutes with positives.
- This is a moral requirement to preserve dependent elimination.

Intuitionistic enough translations

- *Dialectica* shares common properties with (intuitionistic) forcing.

  In particular, $\mathbb{W}(-)$ 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 $(\mathbb{P}, \leq)$)
  - Intuitionistic CPS (similar but with stacks)
  - Dialectica (a rich writer on $\mathbb{C}(A) \to \mathfrak{M}\,\mathbb{C}(\Gamma)$)
  - 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.