The Curry-Howard correspondence in classical analysis and set theory

Jean-Louis Krivine

PPS Group, University Paris 7, CNRS

krivine@pps.jussieu.fr

Florianopolis, July 19, 2005

Introduction

In this tutorial, we introduce the Curry-Howard (proof-program) correspondence which is usually restricted to intuitionistic logic.
We explain how to extend this correspondence to the whole of mathematics and we build a simple suitable machine for this.

1st problem

Each mathematical *proof* must give a *program*
which must be executable in this machine.

2nd problem (*specification problem)*

Understand the behaviour of these programs
i.e. the *specification* associated with a given *theorem*.

The first problem is now completely solved, but the second is far from being so.

Usual λ-calculus

The *λ-terms* are defined as follows, from a given denumerable set of *λ-variables* :

- Each variable is a λ-term.
- If t is a λ-term and x a variable, then $\lambda x\, t$ is a term (abstraction).
- If t, u are terms, then $(t)u$ is a term (application).

Notations. $((t)u_1)\ldots u_n$ is also denoted by $tu_1\ldots u_n$.

The substitution is denoted by $t[u_1/x_1, \ldots, u_n/x_n]$

(replace, in t, each free occurrence of x_i with u_i).

λ-calculus is very important in computer science, because it is the core of every programming language.

It is a very nice structure, with many properties (Church-Rosser, standardization, ...) which has been deeply investigated.

But, in the following, nothing else than the definition above is used about λ-calculus.

A machine in symbolic form

The machine is the program side of the proof-program correspondence.
In these talks, I use only a machine in symbolic form,
not an explicit implementation.
We execute a *process* $t \star \pi$; t is (provisionally) a closed λ-term,
π is a *stack*, that is a sequence $t_1.t_2 \dots t_n.\pi_0$ where
π_0 is a *stack constant*, i.e. a marker for the bottom of the stack.
We denote by $t.\pi$ the stack obtained by "pushing" t on the top of the stack π.

Execution rules for processes (weak head reduction of λ-calculus) :

$tu \star \pi \succ t \star u.\pi$ (*push*)

$\lambda x\, t \star u.\pi \succ t[u/x] \star \pi$ (*pop*)

This symbolic machine will be used to follow the execution of programs
written in an extension of λ-calculus with new instructions.

A machine in symbolic form (cont.)

We get a better approximation of a "real" machine by eliminating substitution.

The execution rules are a little more complicated (head linear reduction) :

$tu \star \pi \succ t \star u.\pi$

$\lambda x_1 \dots \lambda x_k t u \star t_1 \dots t_k.\pi \succ \lambda x_1 \dots \lambda x_k t \star t_1 \dots t_k.v.\pi$

with $v = (\lambda x_1 \dots \lambda x_k u) t_1 \dots t_k$

$\lambda x_1 \dots \lambda x_k x_i \star t_1 \dots t_k.\pi \succ t_i \star \pi.$

It is necessary to add new instructions, because such simple machines can only handle ordinary λ-terms, i.e. programs obtained from proofs in *pure intuitionistic logic*.

Observe that some of these instructions will be *incompatible with β-reduction.*

Intuitionistic Curry-Howard correspondence

Consider second order formulas with $\to$ and $\forall$ as the only logical symbols.

Intuitionistic natural deduction is given by the following usual rules :

$A_1, \ldots, A_k \vdash A_i$

$A_1, \ldots, A_k, A \vdash B \;\Rightarrow\; A_1, \ldots, A_k \vdash A \to B$

$A_1, \ldots, A_k \vdash A \to B,\; A_1, \ldots, A_k \vdash A \;\Rightarrow\; A_1, \ldots, A_k \vdash B$

$A_1, \ldots, A_k \vdash A \;\Rightarrow\; A_1, \ldots, A_k \vdash \forall x\, A$ and $\forall X\, A$
(if x, X are not free in $A_1, \ldots, A_k$)

$A_1, \ldots, A_k \vdash \forall x\, A \to A[t]$

$A_1, \ldots, A_k \vdash \forall X\, A \to A[F/Xx_1 \ldots x_k]$
(comprehension scheme)

$\bot$ is defined as $\forall X X$, X being a propositional variable (predicate of arity 0).

The axiom $\bot \to F$ is therefore a particular case of the comprehension scheme.

Intuitionistic Curry-Howard correspondence (cont.)

These rules become rules for typing λ-terms, as follows :

$x_1{:}A_1, \ldots, x_k{:}A_k \vdash x_i{:}A_i$

$x_1{:}A_1, \ldots, x_k{:}A_k, x{:}A \vdash t{:}B \;\Rightarrow\; x_1{:}A_1, \ldots, x_k{:}A_k \vdash \lambda x\, t{:}A \to B$

$x_1{:}A_1, \ldots, x_k{:}A_k \vdash t{:}A \to B,\; u{:}A \;\Rightarrow\; x_1{:}A_1, \ldots, x_k{:}A_k \vdash tu{:}B$

$x_1{:}A_1, \ldots, x_k{:}A_k \vdash t{:}A \;\Rightarrow\; x_1{:}A_1, \ldots, x_k{:}A_k \vdash t{:}\forall x\, A$ and $t{:}\forall X\, A$
(if x, X are not free in $A_1, \ldots, A_k$)

$x_1{:}A_1, \ldots, x_k{:}A_k \vdash \lambda x\, x{:}\forall x\, A \to A[t]$

$x_1{:}A_1, \ldots, x_k{:}A_k \vdash \lambda x\, x{:}\forall X\, A \to A[F/X x_1 \ldots x_k]$
(comprehension scheme)

In this way, we get programs from proofs in *pure* (i.e. without axioms) *intuitionistic logic*. It is the very first step of our work.

Realizability

We know that proofs in pure intuitionistic logic give λ-terms.
But *pure intuitionistic*, or even *classical*, logic is not sufficient
to write down mathematical proofs.
We need *axioms*, such as *extensionality, infinity, choice, ...*
Axioms are not theorems, they have no proof !
How can we find suitable programs for them ?

The solution is given by the theory of classical realizability
by means of which we define, for each mathematical formula Φ :

- the set of stacks which *are against* Φ, denoted by $\|\Phi\|$
- the set of closed terms t which *realize* Φ, which is written $t \Vdash \Phi$.

We first choose a set of processes, denoted by $\perp\!\!\!\perp$, which is *saturated*, i.e.
$t \star \pi \in \perp\!\!\!\perp,\ t' \star \pi' \succ t \star \pi \ \Rightarrow\ t' \star \pi' \in \perp\!\!\!\perp.$

Realizability (cont.)

The set $\|\Phi\|$ and the property $t \Vdash \Phi$ are defined by induction on the formula Φ.
They are connected as follows :

$$t \Vdash \Phi \Leftrightarrow (\forall \pi \in \|\Phi\|)\, t \star \pi \in \perp\!\!\!\perp$$

Two steps of induction, because we use only two logical symbols : $\to$, $\forall$.

1. $\|\Phi \to \Psi\| = \{t.\pi \,;\, t \Vdash \Phi, \pi \in \|\Psi\|\}$. In words :
if the term t realizes the formula Φ and the stack π is against the formula Ψ
then the stack $t.\pi$ (push t on the top of π) is against the formula $\Phi \to \Psi$.

2. $\|\forall x\, \Phi(x)\| = \bigcup_{a \in A} \|\Phi(a)\|$ where A is the domain of the variable x
(it may be the integers, or the whole universe of sets, ...).
In words : a stack is against $\forall x\, \Phi(x)$ if it is against $\Phi(a)$ for some a.
It follows that $t \Vdash \forall x\, \Phi(x) \Leftrightarrow t \Vdash \Phi(a)$ for all a.

Realizability (cont.)

In the definition of realizability, we need, in fact, one more step which concerns *atomic formulas*.

Now, realizability theory is exactly model theory, in which the truth value set is $\mathcal{P}(\Pi)$ instead of $\{0,1\}$, Π being the set of stacks.

Thus, the notion of *(first or second order) model* is the usual one, except that an *n-ary predicate symbol* is now interpreted in $\mathcal{P}(\Pi)^{M^n}$ instead of $\{0,1\}^{M^n}$ (where M is the base set of the model).

The truth values $\emptyset$ and Π are denoted by $\top$ and $\bot$. Therefore :

$t \Vdash \top$ for every term t ; $t \Vdash \bot \Rightarrow t \Vdash F$ for every F.

n-ary function symbols have their usual interpretation : in M^{M^n}.

The adequation lemma

Given a model, we can still choose the saturated set $\perp\!\!\!\perp$.

The case $\perp\!\!\!\perp = \emptyset$ is degenerate : we get back the usual two-valued model theory.

The lemma below is the analog of the *soundness lemma* for our notion of model.

It is an essential tool for the proof-program correspondence.

Adequation lemma.

If $x_1{:}\Phi_1, \ldots, x_n{:}\Phi_n \vdash t{:}\Phi$ *and if* $t_i \Vdash \Phi_i \, (1 \leq i \leq n)$

then $t[t_1/x_1, \ldots, t_n/x_n] \Vdash \Phi$.

In particular : *If* $\vdash t{:}\Phi$ *then* $t \Vdash \Phi$.

The proof is a simple induction on the length of the derivation of $\ldots \vdash t{:}\Phi$.

In the following, we shall more and more use semantic realizability $t \Vdash \Phi$

instead of syntactic typability $\vdash t{:}\Phi$.

The language of mathematics

The proof-program correspondence is well known for *intuitionistic logic*. Now we have

Mathematics $\equiv$ Classical logic + some axioms that is

Mathematics $\equiv$ Intuitionistic logic + Peirce's law + some axioms

For each axiom $\mathcal{A}$, we choose a closed λ-term which realizes $\mathcal{A}$, *if there is one*.

If not, *we extend our machine* with some *new instruction* which realizes $\mathcal{A}$,

if we can devise such an instruction.

Now, there are essentially two possible axiom systems for mathematics :

1. *Analysis*, i.e. second order classical logic with dependent choice.
2. *ZFC*, i.e. Zermelo-Fraenkel set theory with the full axiom of choice.

Thus, we now have many axioms to deal with.

First of all, we must settle the *law of Peirce* : $((A \to \bot) \to A) \to A$.

Peirce's law

We adapt to our machine the solution found by Tim Griffin in 1990.

We add to the λ-calculus an instruction denoted by cc. Its reduction rule is :

$$\mathsf{cc} \star t.\pi \succ t \star \mathsf{k}_\pi.\pi$$

k_π is a *continuation*, i.e. a pointer to a location where the stack π is saved.

In our symbolic machine, it is simply a λ-constant, indexed by π.

Its execution rule is $\mathsf{k}_\pi \star t.\pi' \succ t \star \pi$.

Therefore cc saves the current stack and k_π restores it.

Using the theory of classical realizability, we show that $\mathsf{cc} \Vdash (\neg A \to A) \to A$.

In this way, we extend the Curry-Howard correspondence to every proof in *pure* (i.e. without axiom) *classical logic* : we now have the new typing rule

$$x_1{:}A_1, \ldots, x_k{:}A_k \vdash \mathsf{cc}{:}(\neg A \to A) \to A$$

Peirce's law (cont.)

Let us check that cc $\Vdash (\neg A \to A) \to A$: take $t \Vdash \neg A \to A$ and $\pi \in \|A\|$.
For every $u \Vdash A$, we have $u \star \pi \in \bot\!\!\!\bot$, thus $\mathsf{k}_\pi \star u.\pi' \in \bot\!\!\!\bot$ for every stack π'.
Thus $\mathsf{k}_\pi \Vdash A \to \bot$ and $\mathsf{k}_\pi.\pi \in \|\neg A \to A\|$.
It follows that $t \star \mathsf{k}_\pi.\pi \in \bot\!\!\!\bot$ thus $\mathsf{cc} \star t.\pi \in \bot\!\!\!\bot$. QED

We have now an extended λ-calculus, which we call *λ_c-calculus*.
A *proof-like term* is a closed λ_c-term which contains no continuation.
We say that *the formula Φ is realized* if there is a proof-like term τ such that $\tau \Vdash \Phi$ *for every choice of* $\bot\!\!\!\bot$. Thus :

- Every λ_c-term which comes from a proof is proof-like.
- If the axioms are realized, every provable formula is realized.

If $\bot\!\!\!\bot \neq \emptyset$, then $\tau \Vdash \bot$ for some λ_c-term τ : take $t \star \pi \in \bot\!\!\!\bot$ and $\tau = k_\pi t$.
Observe that it is not a proof-like term.

First simple theorems

The choice of $\perp\!\!\!\perp$ is generally done according to the theorem Φ for which we want to solve the specification problem. Let us take two simple examples.

Theorem. If θ comes from a proof of $\forall X(X \to X)$ (with any realized axioms) then $\theta \star t.\pi \succ t \star \pi$ i.e. θ behaves like $\lambda x\, x$.

Proof. Take $\perp\!\!\!\perp = \{\mathsf{p}\ ;\ \mathsf{p} \succ t \star \pi\}$ and $\|X\| = \{\pi\}$.

Thus $t \Vdash X$ and $\theta \star t.\pi \in \perp\!\!\!\perp$. QED

Example : $\theta = \lambda x\, \mathsf{cc}\lambda k\, kx$.

Dual proof. Take $\perp\!\!\!\perp^c = \{\mathsf{p}\ ;\ \theta \star t.\pi \succ \mathsf{p}\}$ and $\|X\| = \{\pi\}$.

Thus, $\theta \star t.\pi \in \perp\!\!\!\perp^c$; since $\pi \in \|X\|$ and $\theta \Vdash X \to X$, we have $t \nVdash X$ and therefore $t \star \pi \in \perp\!\!\!\perp^c$. QED

First simple theorems (cont.)

The formula Bool$(x) \equiv \forall X(X1, X0 \to Xx)$ is equivalent to $x = 1 \vee x = 0$.

Theorem. If θ comes from a proof of Bool(1), then $\theta \star t.u.\pi \succ t \star \pi$

i.e. θ behaves like the boolean $\lambda x \lambda y\, x$.

Proof. Take $\bot\!\!\!\bot = \{\mathsf{p}\ ;\ \mathsf{p} \succ t \star \pi\}$, $\|X1\| = \{\pi\}$ and $\|X0\| = \emptyset = \|\top\|$.

Thus $t \Vdash X1$, $u \Vdash X0$ and $\theta \star t.u.\pi \in \bot\!\!\!\bot$. QED

Dual proof. Take $\bot\!\!\!\bot^c = \{\mathsf{p}\ ;\ \theta \star t.u.\pi \succ \mathsf{p}\}$, $\|X1\| = \{\pi\}$ and $\|X0\| = \emptyset = \|\top\|$.

We have $u \Vdash X0$, $\pi \in \|X1\|$, $\theta \Vdash X1, X0 \to X1$ and $\theta \star t.u.\pi \in \bot\!\!\!\bot^c$.

Thus $t \not\Vdash X1$ and $t \star \pi \in \bot\!\!\!\bot^c$. QED

Another example : $\exists x(Px \to \forall y\, Py)$

Write this theorem $\forall x[(Px \to \forall y\, Py) \to \bot] \to \bot$. We must show :
$z{:}\forall x[(Px \to \forall y\, Py) \to \bot] \vdash\ ?{:}\bot$. We get $z{:}(Px \to \forall y\, Py) \to \bot$,
$z{:}(Px \to \forall y\, Py) \to Px$, cc $z{:}Px$, cc $z{:}\forall x\, Px$, λd cc $z{:}Px \to \forall y\, Py$
and $z\lambda d$ cc $z{:}\bot$. Finally we have obtained the program $\theta = \lambda z\, z\lambda d$ cc z.
Let us find a characteristic feature in the behaviour of *all terms θ*
such that $\vdash \theta{:}\exists x(Px \to \forall y\, Py)$. Let $\alpha_0, \alpha_1, \ldots$ and $\varpi_0, \varpi_1, \ldots$
be a fixed sequence of terms and of stacks. We define a *new instruction κ* ;
its reduction rule uses two players named $\exists$ and $\forall$ and is as follows :

$$\kappa \star \xi.\pi \succ \xi \star \alpha_i.\varpi_j$$

where i is first chosen by $\exists$, then j by $\forall$.
The player $\exists$ wins iff the execution arrives at $\alpha_i \star \varpi_i$ for some $i \in \mathbb{N}$.

$\exists x(Px \to \forall y\, Py)$ (cont.)

Theorem. If $\vdash \theta{:}\forall x[(Px \to \forall y\, Py) \to \bot] \to \bot$, there is a winning strategy for $\exists$ when we execute the process $\theta \star \kappa.\pi$ (for any stack π).

Proof. Let $\Vdash$ be the set of processes for which there is a winning strategy for $\exists$. Define a realizability model on $\mathbb{N}$, by setting $\|Pn\| = \{\varpi_n\}$. Thus $\alpha_n \Vdash Pn$.

Suppose that $\xi \Vdash Pi \to \forall y\, Py$ for some $i \in \mathbb{N}$. Then :

$\xi \star \alpha_i.\varpi_j \in \Vdash$ for every j and it follows that $\kappa \star \xi.\pi \in \Vdash$, for any stack π : indeed, a strategy for $\exists$ is to play i and to continue with a strategy for $\xi \star \alpha_i.\varpi_j$ if $\forall$ plays j.

It follows that $\kappa \Vdash (Pi \to \forall y\, Py) \to \bot$ and therefore :

$\kappa \Vdash \forall x[(Px \to \forall y\, Py) \to \bot]$. Thus, $\theta \star \kappa.\pi \in \Vdash$ for every stack π. QED

$\exists x(Px \to \forall y\, Py)$ (cont.)

For instance, if $\theta = \lambda z\, z\lambda d\, \mathsf{cc}\, z$, we have :

$\theta \star \kappa.\pi \succ \kappa \star \lambda d\, \mathsf{cc}\, \kappa.\pi \succ \lambda d\, \mathsf{cc}\, \kappa \star \alpha_{i_0}.\varpi_{j_0}$ if $\exists$ plays i_0 and $\forall$ plays j_0.

We get $\mathsf{cc} \star \kappa.\varpi_{j_0} \succ \kappa \star k_{\varpi_{j_0}}.\varpi_{j_0}$. A winning strategy for $\exists$ is now to play j_0 : if $\forall$ plays j_1, this gives $k_{\varpi_{j_0}} \star \alpha_{j_0}.\varpi_{j_1} \succ \alpha_{j_0} \star \varpi_{j_0}$.

Remark. The program θ does not give explicitly a winning strategy.

Programs associated with proofs of *arithmetical theorems* will give such strategies, i.e. will play in place of $\exists$. Examples in the following.

Axioms for mathematics

Let us now consider the usual axiomatic theories which formalize mathematics.

• Analysis is written in second order logic. There are three groups of axioms :

1. Equations such as $x + 0 = x$, $x + sy = s(x + y)$, ...

and inequations such as $s0 \neq 0$.

2. The recurrence axiom $\forall x\, \mathrm{int}(x)$, which says that each individual (1st order object) is an integer. The formula $\mathrm{int}(x)$ is : $\forall X\{\forall y(Xy \to Xsy), X0 \to Xx\}$.

3. The axiom of dependent choice :

If $\forall X \exists Y\, F(X, Y)$, then there exists a sequence X_n such that $F(X_n, X_{n+1})$.

Analysis is sufficient to formalize a very important part of mathematics including the theory of functions of real or complex variables, measure and probability theory, partial differential equations, analytic number theory, Fourier analysis, etc.

Axioms for mathematics (cont.)

- Axioms of ZFC can be classified in three groups :

1. Equality, extensionality, foundation.
2. Union, power set, substitution, infinity.
3. Choice : Any product of non void sets is non void ;
   possibly other axioms such as CH, GCH, large cardinals.

In order to realize axioms 1 and 2 (i.e. ZF), we must interpret ZF
in another theory called ZF_ε which is much simpler to realize.
The λ_c-terms for ZF are rather complicated, but do not use new instructions.
The solution for AC and CH has been found very recently.
We need new instructions and get very complicated programs for these axioms.

Realizability models

In the following, we consider realizability models of 2nd order logic and of set theory.
For models of 2nd order logic, the domain of individuals is always $\mathbb{N}$
and the domain of n-ary predicate variables is always $\mathcal{P}(\Pi)^{M^n}$.
The only left free choice is $\perp\!\!\!\perp$.
But it is important to remember that these domains are used only
for computing the truth values of formulas : $\|\forall x\, \Phi(x)\| = \bigcup_{n\in\mathbb{N}} \|\Phi(n)\|$.
For example, it does not mean that the formula : " every individual is an integer ",
that is the recurrence axiom $\forall x \forall X [\forall y(Xy \to Xsy), X0 \to Xx]$ is realized.
Indeed, for the most interesting choices of $\perp\!\!\!\perp$, the negation of this formula is realized.

Coherence

In fact, there are very interesting examples of $\perp\!\!\!\perp$ for which $\perp$ is realized :
for instance, $\perp\!\!\!\perp =$ the set of processes the reduction of which is infinite ;
in this case, we have $\delta\delta \Vdash \perp$.

Now, by adequation lemma, we know that the set of realized formulas
is closed by classical deduction. If this set is consistent, we say that $\perp\!\!\!\perp$ is *coherent*.
It means that there is no proof-like term θ such that $\theta \Vdash \perp$.
In other words, for every proof-like term θ, there is a stack π such that $\theta \star \pi \notin \perp\!\!\!\perp$.
In what follows, we are mainly concerned with coherent $\perp\!\!\!\perp$.
Examples : let p_o be some given process ; then $\perp\!\!\!\perp = \{\mathrm{p};\ \mathrm{p} \succ \mathrm{p}_o\}$ is coherent if
there is at least 2 stack constants ; $\perp\!\!\!\perp = \{\mathrm{p};\ \mathrm{p}_o \not\succ \mathrm{p}\}$ is not coherent in general.

2-valued realizability models

Let $\perp\!\!\!\perp$ be a coherent saturated set of processes. Then the set of realized closed formulas is closed under derivation in classical logic and does not contain $\perp$.
It is therefore consistent and we obtain, in this way, 2-valued models
of second order logic or of set theory.
We shall see that these models are very different from the model we started with.
As said before, there exist individuals which are not integers ; but there are also non-standard integers in the following strong sense : there is a unary predicate P such that the formulas $\exists x[\mathrm{int}(x) \wedge Px]$, $\neg\, Pn$ are realized for each integer n.

The Boolean algebra $\mathcal{P}(\Pi)$

Every coherent $\perp\!\!\!\perp$ gives a *Boolean structure* on the set $\mathcal{P}(\Pi)$ of truth values :

for $\mathcal{X}, \mathcal{Y} \subset \Pi$, define :

$$\mathcal{X} \leq \mathcal{Y} \Leftrightarrow \text{there is a proof-like term } \theta \text{ s.t. } \theta \Vdash \mathcal{X} \to \mathcal{Y} \quad .$$

It is easy to prove that it is a Boolean preorder on $\mathcal{P}(\Pi)$, with $\mathcal{X}^c = \|\neg\mathcal{X}\|$ and

$\inf(\mathcal{X}, \mathcal{Y}) = \|\mathcal{X} \wedge \mathcal{Y}\| = \|\forall X((\mathcal{X}, \mathcal{Y} \to X) \to X)\|$ or $\|(\mathcal{X}, \mathcal{Y} \to \perp) \to \perp\|$,

$\sup(\mathcal{X}, \mathcal{Y}) = \|\mathcal{X} \vee \mathcal{Y}\| = \|\forall X((\mathcal{X} \to X), (\mathcal{Y} \to X) \to X)\|$

or $\|(\mathcal{X} \to \perp), (\mathcal{Y} \to \perp) \to \perp\|$.

Let $\mathcal{B} = \mathcal{P}(\Pi)/\simeq$ be this Boolean algebra.

Every closed formula has a value in $\mathcal{P}(\Pi)$ and therefore a value in $\mathcal{B}$.

We get, in this way, *Boolean models* of second order logic or set theory.

Using any ultrafilter on $\mathcal{B}$, we obtain again the 2-valued realizability models described in the last slide.

Axioms of analysis : equations

Axioms : $\neg(0 = s0)$; $p0 = 0$; $\forall x(psx = x)$; $\forall x(x + 0 = x)$; $\forall x(x.0 = 0)$;
$\forall x \forall y(x + sy = s(x + y))$; $\forall x \forall y(x.sy = xy + x)$

Such equations and inequations are very easy to realize.

Theorem. Any true equation is realized by $\lambda x\, x$.

Any true inequation is realized by $\lambda x\, xt$ for an arbitrary t.

Proof. $x = y$ is defined by $\forall X(Xx \to Xy)$ in second order logic. QED

Useful definition. Define a new predicate $x \neq y$ by setting :
$\|n \neq p\| = \emptyset = \|\top\|$ if $n \neq p$ and $\|n \neq p\| = \sqcap = \|\bot\|$ if $n = p$.

Theorem. $\lambda x \lambda y\, yx \Vdash \forall x \forall y[x \neq y \to \neg(x = y)]$ and
$\lambda x\, xt \Vdash \forall x \forall y[\neg(x = y) \to x \neq y]$ for any t.

This means we can use the predicate $x \neq y$ in place of $\neg(x = y)$.

Axioms of analysis : recurrence

The proper recurrence axiom is $\forall x\,\mathrm{int}(x)$, where $\mathrm{int}(x)$ is the formula :

$$\forall X[X0, \forall x(Xx \to Xsx) \to Xx]$$

This axiom is *impossible to realize*, even by means of new instructions.
This means that there must be individuals which are not integers.
There are two solutions, which are logically equivalent for integers ;
but they correspond to very different programming styles.
The first method is to *discard* the recurrence axiom
and restrict first order quantifiers to the formula $\mathrm{int}(x)$.
The second method is the same we shall use to realize axioms of ZF.
We define a new equality $\simeq$ on individuals, which allows to realize
the recurrence axiom : every individual becomes *equivalent* to an integer.
It uses a fixpoint combinator and the programming style is that of LISP.

Recurrence axiom, 1st method

The language has a function symbol for each recursive function.

The set of individuals is $\mathbb{N}$. Let $\mathrm{int}(x) \equiv \forall X[\forall y(Xy \to Xsy), X0 \to Xx]$.

Theorem. If a second order formula Φ is provable with the recurrence axiom, then the restricted formula Φ^{int} is provable without it, using the axioms $\forall x_1 \ldots \forall x_k\{\mathrm{int}(x_1), \ldots, \mathrm{int}(x_k) \to \mathrm{int}(f(x_1, \ldots, x_k))\}$ for each symbol f.

Now, we only need to realize these new axioms. There are two ways of doing this :

- Prove this formula from *true equations*.

Examples. The successor s : $\mathrm{int}(x) \to \mathrm{int}(sx)$ is provable with no equation.

Addition : $\mathrm{int}(x), \mathrm{int}(y) \to \mathrm{int}(x + y)$ is provable with the equations :

$x + 0 = x$; $x + sy = s(x + y)$, ...

This works for a very large class of recursive functions :

the provably total functions in second order arithmetic.

Recurrence axiom (cont.)

• The second method works for *every recursive function* f.

Assume, for simplicity, that f is unary. We have two lemmas.

Lemma. If τ is a closed λ-term, $\tau \simeq_\beta \underline{n}$ (Church integer), then $\tau \Vdash \mathrm{int}(s^n 0)$.

Define $T = \lambda f \lambda n(n) \lambda g\, g \circ s.f.0$ (*storage operator* [5]).

Storage lemma. *If* $(\forall \pi \in \|X\|)\ \phi \star s^n 0.\pi \in \perp\!\!\!\perp$ *then* $T\phi \Vdash \mathit{int}(n) \to X$.

Proof. Let $\|Pj\| = \{s^{n-j}0.\pi;\ \pi \in \|X\|\}$ for $0 \leq j \leq n$;

$\|Pj\| = \emptyset$ for $j > n$. Then $\lambda g\, g \circ s \Vdash \forall x(Px \to Psx)$ and $\phi \Vdash P0$.

Thus, if $\nu \Vdash \mathrm{int}(n)$ then $\nu \star \lambda g\, g \circ s.\phi.\pi \in \perp\!\!\!\perp$ which gives $T\phi \star \nu.\pi \in \perp\!\!\!\perp$. QED

Theorem. Let τ be a closed λ-term which computes the recursive function f.

Then $T\lambda x\, \tau x \Vdash \forall x[\mathrm{int}(x) \to \mathrm{int}(f(x))]$.

By the storage lemma, we only need to prove that $\lambda x\, \tau x \star s^n 0.\pi \in \perp\!\!\!\perp$

for $\pi \in \|\mathrm{int}(f s^n 0)\|$. But this follows from the first lemma,

since $\tau s^n 0 \simeq_\beta \underline{r}$ with $r = f(n)$. QED

Imperative call-by-value

Let $\nu \in \Lambda_c^0$ such that $\vdash \nu$:int$(s^n 0)$; i.e. ν "behaves like" the integer n.
In the λ_c-term $\phi\nu$ this data is *called by name* by the program ϕ.
In the λ_c-term $T\phi\nu$ the same data is *called by value* by ϕ,
which means it is computed first (in the form $s^n 0$).

Theorem. If $\vdash \nu$:int$(s^n 0)$, then $T\phi \star \nu.\pi \succ \phi \star s^n 0.\pi$.

Take $\perp\!\!\!\perp = \{\mathsf{p};\ \mathsf{p} \succ \phi \star s^n 0.\pi\}$. Then $T\phi \star \nu.\pi \in \perp\!\!\!\perp$, by the storage lemma. QED

I name this behaviour *imperative* call-by-value, to avoid confusion with
the well-known notion of (functional) call-by-value, and because
it is very similar to the usual notion of call-by-value in imperative languages.
It is only defined for data types (booleans, integers, trees, ...)

Computing recursive functions

So, we can discard the recurrence axiom and replace it with the formulas :
$\forall x_1 \dots \forall x_k \{\mathsf{int}(x_1), \dots, \mathsf{int}(x_k) \to \mathsf{int}(f(x_1, \dots, x_k))\}$ for each symbol f.
Theorem. If $\vdash \phi : \forall \vec{x} \{\vec{\mathsf{int}}(\vec{x}) \to \mathsf{int}(f\vec{x})\}$, then ϕ computes the function f, i.e. :
if $\underline{\vec{n}}$ is a sequence of Church integers, then $T\kappa \star \phi \underline{\vec{n}}.\pi \succ \kappa \star s^p 0.\pi$ with $p = f(\vec{n})$.
This works for every data types : Booleans, integers, sums, product and lists of data types, etc. In this tutorial, we only use the type of Booleans :
$\mathsf{Bool}(x) \equiv \forall X(X1, X0 \to Xx)$. For this type we have :
Theorem. If $\vdash \phi : \forall x \{\mathsf{int}(x) \to \mathsf{Bool}(f(x))\}$, then
$\phi \star \widehat{n}.t.u.\pi \succ t \star \pi$ if $f(n) = 1$
$\phi \star \widehat{n}.t.u.\pi \succ u \star \pi$ if $f(n) = 0$
where $\widehat{n}$ is any closed λ-term β-equivalent to the Church integer n.

Recurrence axiom, 2nd method

The following formula tells that the relation $sy = x$ is well founded :

$$\forall x[\forall y(Xy \to sy \neq x) \to \neg Xx] \to \forall x\, \neg Xx$$

It is realized by the Turing fixpoint combinator Y, wich has the reduction rule :

$$\mathsf{Y} \star t.\pi \succ t \star \mathsf{Y}t.\pi.$$

Define now a new predicate $x \leq y$ by the recursive equation :

$\|x \leq y\| = \|\forall x'[\forall y'(x' \leq y' \to y \neq sy') \to x \neq sx']\|$ that is :

$\|0 \leq n\| = \|\top\|$ $(= \emptyset)$, $\|m + 1 \leq 0\| = \|\top \to \bot\|$,

$\|m + 1 \leq n + 1\| = \|(m \leq n \to \bot) \to \bot\|$ for every $m, n \in \mathbb{N}$.

Obviously $\lambda x\, x \Vdash \forall x \forall y\{x \leq y \leftrightarrow \forall x'[\forall y'(x' \leq y' \to y \neq sy') \to x \neq sx']\}$

Define $x \simeq y$ by $x \leq y \wedge y \leq x$. Then, we can *prove* $\forall x(\exists! y \simeq x)\, \mathsf{int}(y)$.

Fixpoint and well foundedness

Let $x \sqsubset y$ be well founded on integers and $\phi(x,y)$ its characteristic function. Then

$$\mathsf{Y} \Vdash \forall X\{\forall x[\forall y(Xy \to \phi(y,x) \neq 1) \to \neg Xx] \to \forall x\, \neg Xx\}$$

Proof. Let $t \Vdash \forall x[\forall y(\mathcal{X}(y) \to \phi(y,x) \neq 1) \to \neg\mathcal{X}(x)]$
for some $\mathcal{X} : \mathbb{N} \to \mathcal{P}(\Pi)$. We prove $\mathsf{Y}t \Vdash \neg\mathcal{X}(n)$ by induction on n,
following $\sqsubset$. Let $u \Vdash \mathcal{X}(n)$, we must prove $\mathsf{Y} \star tu\pi \in \perp\!\!\!\perp$, i.e. $t \star \mathsf{Y}t.u.\pi \in \perp\!\!\!\perp$.
It is sufficient to prove $\mathsf{Y}t \Vdash \forall y(\mathcal{X}(y) \to \phi(y,n) \neq 1)$.
Now, if $y \sqsubset n$, it is true because $\mathsf{Y}t \Vdash \mathcal{X}(y) \to \bot$, by induction hypothesis ;
else it is true because $\|\phi(y,n) \neq 1\| = \emptyset$. Q.E.D.

We use this principle in a more general form.

Notation. $\vec{X} = (X_1, \ldots, X_n)$ denotes a finite sequence of predicates.

$$\mathsf{Y} \Vdash \forall x[\forall y(\forall z(\vec{X}(y,z) \to \phi(y,x) \neq 1) \to \forall z(\vec{X}(x,z) \to \bot)] \to \forall x \forall z(\vec{X}(x,z) \to \bot)$$

Remark. This formula is clearly derivable from the formula above,
but the interesting fact is that Y realizes it.

Examples of arithmetical theorems

Theorem. Let $\vdash \theta : \exists x[\mathrm{int}(x) \wedge f(x) = 0]$, with f recursive. Let κ be a stop instruction. Then $\theta \star T\kappa.\pi \succ \kappa \star s^n 0.\pi$ with $f(n) = 0$; T is the storage operator.

Proof. We have $\theta \Vdash \forall x[\mathrm{int}(x) \to f(x) \neq 0] \to \bot$.

Now take $\perp\!\!\!\perp = \{\mathrm{p}\ ;\ \mathrm{p} \succ \kappa \star s^n 0.\pi \text{ with } f(n) = 0\}$.

We simply have to show that $T\kappa \Vdash \forall x[\mathrm{int}(x) \to f(x) \neq 0]$

i.e. by the storage lemma, that $\kappa \star s^n 0.\pi \in \perp\!\!\!\perp$ for every n such that $\pi \in \|f(n) \neq 0\|$.

But this means that $\|f(n) \neq 0\| \neq \emptyset$ and thus $f(n) = 0$. QED

Remark. κ is clearly a *pointer to an integer*. In the program, we wrote $T\kappa$, because we want it to point to a *computed* integer.

It is the intuitive meaning of *imperative call-by-value*.

Examples of arithmetical theorems (cont.)

We consider now an arithmetical theorem $\{\exists x \forall y [f(x,y) \neq 0]\}^{\mathsf{int}}$.

Define a game with two players $\exists$ and $\forall$: $\exists$ plays an integer m, $\forall$ answers by n ;

the play stops as soon as $f(m,n) \neq 0$ and then $\exists$ won ;

thus $\forall$ wins if and only if the play does not stop.

Intuitively, $\exists$ is the " defender " of the theorem and

$\forall$ " attacks " it, searching to exhibit a counter-example.

It is clear that $\exists$ has a winning strategy if and only if $\mathbb{N} \models \exists x \forall y [f(x,y) \neq 0]$;

then, there is an obvious strategy for $\exists$: simply play successively $0, 1, 2, \ldots$

We show that a proof of $\{\exists x \forall y [f(x,y) \neq 0]\}^{\mathsf{int}}$ gives an explicit programming of a winning strategy for the " defender ".

Usually, this strategy is much more efficient than the trivial one.

Programming a winning strategy

Let us add to our symbolic machine, an instruction κ which allows an interactive execution. Its execution rule is :

$$\kappa \star s^n 0.\xi.\pi \succ \xi \star s^p 0.\pi_{np}$$

for $n, p \in \mathbb{N}$; π_{np} is a stack constant.

This execution rule is non deterministic since p is arbitrary. Intuitive meaning :

in the left hand side, the program (the player $\exists$), plays the integer n and prepares a handler ξ for the answer of $\forall$; in the right hand side, the attacker $\forall$ plays p ;

π_{np} store the stroke.

Theorem. If $\vdash \theta : \{\exists x \forall y (f(x,y) \neq 0)\}^{\mathsf{int}}$, then *every reduction* of $\theta \star T\kappa.\pi$ ends up into $\xi \star s^p 0.\pi_{np}$ with $f(n,p) \neq 0$ (T is the storage operator).

This means that the process $\theta \star T\kappa.\pi$ acts as a winning strategy for $\exists$.

Programming a winning strategy (cont.)

Proof. Take for $\perp\!\!\!\perp$ the set of processes *every* reduction of which ends up into $\xi \star s^p 0.\pi_{np}$ with $f(n,p) \neq 0$. We must show that $\theta \star T\kappa.\pi \in \perp\!\!\!\perp$.

Now $\theta \Vdash \forall x[\mathsf{int}(x), \forall y(\mathsf{int}(y) \to f(x,y) \neq 0) \to \bot] \to \bot$.

Therefore, by definition of $\Vdash$, it is sufficient to show that :

$T\kappa \Vdash \forall x[\mathsf{int}(x), \forall y(\mathsf{int}(y) \to f(x,y) \neq 0) \to \bot]$.

By the storage lemma, we only need to show that :

if $\xi \Vdash \forall y(\mathsf{int}(y) \to f(n,y) \neq 0)$ then $\kappa \star s^n 0.\xi.\pi \in \perp\!\!\!\perp$, i.e.

$\xi \star s^p 0.\pi_{np} \in \perp\!\!\!\perp$ for every $p \in \mathbb{N}$.

If $f(n,p) \neq 0$, it is true by definition of $\perp\!\!\!\perp$.

Else, $\pi_{np} \in \|f(n,p) \neq 0\| = \Pi$, hence the result, by hypothesis on ξ. QED

Programming a winning strategy (cont.)

Remark. κ can be considered as a *pointer* to the *object* (n, ξ) consisting of the integer n and the handler ξ (data and method). Moreover, the integer n is *called by value* which is guaranteed by writing $T\kappa$ instead of κ.

Example. We take the theorem $\{\exists x \forall y[f(x) \leq f(y)]\}^{\mathsf{int}}$ where f is recursive.
Let $\phi(x, y)$ be the characteristic function of the well founded relation $f(x) < f(y)$.
The theorem is $\forall x[\mathsf{int}(x), \forall y(\mathsf{int}(y) \to \phi(y, x) \neq 1) \to \bot] \to \bot$. Now :
$\mathsf{Y} \Vdash \forall x[\forall y(\mathsf{int}(y) \to \phi(y, x) \neq 1) \to \neg\mathsf{int}(x)] \to \forall x\, \neg\mathsf{int}(x)$
and we get $\theta = \lambda h(\mathsf{Y}\lambda x \lambda n\, hnx)0$. It is easily checked that the process $\theta \star T\kappa.\pi$ gives the following strategy, much better than the trivial one :
$\exists$ plays 0 ; if $\forall$ plays p and if $f(p) < f(0)$, then $\exists$ plays p and so on.

Programming a winning strategy (cont.)

We now treat the same example by a proof. For simplicity, we assume $f : \mathbb{N} \to \{0, 1\}$. We have a proof-like term ϕ which computes f, i.e. $\vdash \phi{:}\forall x(\mathsf{int}(x) \to \mathsf{Bool}(fx))$. We prove $\vdash \{\exists x \forall y[f(x)f(y)^c \neq 1]\}^{\mathsf{int}}$, i.e. $h{:}\forall x\{\mathsf{int}(x), \forall y(\mathsf{int}(y) \to f(x)f(y)^c \neq 1) \to \bot\} \vdash ? : \bot$.

Let us declare $x{:}\mathsf{int}(x)$, $y{:}\mathsf{int}(y)$ and $r{:}\forall x[\mathsf{int}(x) \to f(x) \neq 0]$ (because there are two cases according to : f takes the value 0 or not). Then : $\phi y{:}\ \mathsf{Bool}(fy)$ and therefore : $\phi y{:}(1{\neq}0 \to 0{\neq}1), (0{\neq}0 \to f(x){\neq}1) \to [f(y){\neq}0 \to f(x)f(y)^c{\neq}1]$.

Since $ry{:}f(y){\neq}0$, we get $\lambda y(\phi y a_o I)(r)y{:}\forall y[\mathsf{int}(y) \to f(x)f(y)^c{\neq}1]$ and $(hn_o)\lambda y(\phi y a_o I)(r)y{:}\bot$; a_o (resp. n_o) is an arbitrary closed term (resp. integer).

Let $\xi_h = \lambda r(hn_o)\lambda y(\phi y a_o I)(r)y$; then $\xi_h{:}\forall x[\mathsf{int}(x) \to f(x) \neq 0] \to \bot$.

Now, we have also $\phi x{:}\mathsf{Bool}(f(x))$ and therefore

Programming a winning strategy (cont.)

ϕx:$[\neg\forall y(\mathsf{int}(y) \to f(y)^c{\neq}1) \to 1{\neq}0], [\neg\forall y(\mathsf{int}(y) \to 0{\neq}1) \to 0{\neq}0]$
$\to [\neg\forall y(\mathsf{int}(y) \to f(x)f(y)^c{\neq}1) \to f(x){\neq}0]$

It follows that $((\phi x a_o)\lambda z\, z a_o)(h)x : f(x){\neq}0$. Let us set
$\rho_h = \lambda x((\phi x a_o)\lambda z\, z a_o)(h)x$; then $\rho_h : \forall x[\mathsf{int}(x) \to f(x){\neq}0]$ and $\xi_h\rho_h : \bot$.
Finally, set $\theta = \lambda h\, \xi_h\rho_h$. Then we have $\vdash \theta : \{\exists x\forall y[f(x)f(y)^c \neq 1]\}^{\mathsf{int}}$.
Let us check the behaviour of the process $\theta \star h.\pi$ which gives
$h \star n_o.\lambda y(\phi y a_o I)(\rho_h)y.\pi$. Thus $\exists$ plays n_o, then $\forall$ plays p_o and we get
$\lambda y(\phi y a_o I)(\rho_h)y \star p_o.\pi_{n_o p_o} \succ \phi p_o \star a_o.I.\rho_h p_o.\pi_{n_o p_o}$
If $f(p_o) = 1$, $\exists$ wins and the play stops. Else, we get $\rho_h \star p_o.\pi_{n_o p_o}$, then
$\phi p_o \star a_o.\lambda z\, z a_o.h p_o.\pi_{n_o p_o} \succ \lambda z\, z a_o \star h p_o.\pi_{n_o p_o} \succ h \star p_o.a_o.\pi_{n_o p_o}$.
It means that $\exists$ *plays* p_o and wins because $f(p_o) = 0$; $\forall$ plays q_o
and the execution ends with $a_o \star q_o.\pi_{p_o q_o}$.

The axiom of dependent choice

We need a *new instruction* in our machine. Any of the following two will work :

1. The signature. Let $t \mapsto n_t$ be a function from closed terms into the integers, which is *very easily computable* and *"practically" one-to-one*. It means that the one-to-one property has to be true only for the terms which appear during the execution of a given process. And also that we never try to compute the inverse function.
We define an instruction σ with the following reduction rule :

$$\sigma \star t.\pi \succ t \star \underline{n_t}.\pi.$$

A simple way to implement such an instruction is to take for n_t the *signature* of the term t, given by a standard algorithm, such as MD5 or SHA1.
Indeed, these functions are almost surely one-to-one for the terms which appear during a finite execution of a given process.

The axiom of dependent choice (cont.)

2. The clock. It is denoted as $\hbar$ and its reduction rule is :

$$\hbar \star t.\pi \succ t \star \underline{n}.\pi$$

where $\underline{n}$ is a Church integer which is the current time (for instance, the number of reduction steps from the boot).

Both instructions, the clock and the signature, can be given (realize) the same type, which is not DC but a formula DC' which *implies DC in classical logic*.
By means of this proof, we get a λ-term $\gamma[\mathsf{cc}, \sigma]$ or $\gamma[\mathsf{cc}, \hbar]$ which has the type DC.
The instructions σ, $\hbar$ appear only inside this λ-term γ.
By looking at its behavior, we find that the integers produced by these instructions are only compared with each other. No other operation is performed on these integers.

" Proof " of the countable choice axiom

For simplicity, we consider only the *countable choice axiom* :

$$\exists Z \forall x (F[x, Z(x,y)/Xy] \to \forall X\, F[x,X])$$

and a variant of the instruction σ with the following reduction rule :

$$\sigma \star t.\pi \succ t \star \underline{n_\pi}.\pi$$

($\pi \mapsto n_\pi$ is a given recursive bijection of Π onto $\mathbb{N}$).

Theorem. There exists a " predicate " $U : \mathbb{N}^3 \to \mathcal{P}(\Pi)$ such that
$\sigma \Vdash \forall x \{\forall n(\mathsf{int}[n] \to F[x, U(x,n,y)/Xy]) \to \forall X\, F[x,X]\}$.

The usual countable choice axiom follows easily, *but not intuitionistically*.

Simply define, for each x, the unary predicate $Z(x,\bullet)$ as $U(x,n,\bullet)$ for the first integer n s.t. $\neg F[x, U(x,n,y)/Xy]$, or as $\mathbb{N}$ if there is no such integer :

$$Z(x,z) \equiv \forall n\{\mathsf{int}(n), \forall p(\mathsf{int}(p), p < n \to F[x, U(x,p,y)/Xy]),$$
$$\neg F[x, U(x,n,y)/Xy] \to U(x,n,z)\}.$$

The countable choice axiom (cont.)

Proof. By definition of $\|\forall X\, F[x,X]\|$, we have :

$\pi \in \|\forall X\, F[x,X]\| \Leftrightarrow (\exists R \in \mathcal{P}(\Pi)^{\mathbb{N}})\, \pi \in \|F[x,R/X]\|$.

By countable choice, we get a function $U : \mathbb{N}^3 \to \mathcal{P}(\Pi)$ such that

$\pi \in \|\forall X\, F[x,X]\| \Leftrightarrow \pi \in \|F[x,U(x,n_\pi,y)/Xy]\|$.

Let $x \in \mathbb{N}$, $t \Vdash \forall n(\mathsf{int}[n] \to F[x,U(x,n,y)/Xy])$ and $\pi \in \|\forall X\, F[x,X]\|$.

We must show that $\sigma \star t.\pi \in \perp\!\!\!\perp$ and, by the rule for σ,

it suffices to show $t \star \underline{n}_\pi.\pi \in \perp\!\!\!\perp$. But this follows from

$\underline{n}_\pi \Vdash \mathsf{int}(s^{n_\pi}0)$, $\pi \in \|F[x,U(x,n_\pi,y)/Xy]\|$ (by definition of U) and

$t \Vdash \mathsf{int}(s^{n_\pi}0) \to F[x,U(x,n_\pi,y)/Xy]$. QED

A program for the CCA

The explicit writing of the program $\gamma[\mathsf{cc}, \hbar]$ of type CCA is as follows :

$$\gamma = \lambda f(\hbar)(\mathsf{Y})\lambda x\lambda n(\mathsf{cc})\lambda k\, f\tau_0\tau_1$$

with $\tau_0 = \lambda z\, zxnk$, $\tau_1 = \lambda y\lambda x'\lambda n'\lambda k'\, \mathsf{Comp}\, nn'\alpha\alpha' y$,

$\alpha = (k)(x')n$, $\alpha' = (k')(x)n'$,

$\mathsf{Comp}\, nn'\alpha\alpha' y = \alpha$ if $n < n'$, α' if $n' < n$, y if $n = n'$.

Consider a process $\gamma \star f.\pi$ in which γ is in head position. We have :

$\gamma \star f.\pi \succ \hbar \star \mathsf{Y}\xi_f.\pi$ where $\xi_f = \lambda x\lambda n(\mathsf{cc})\lambda k\, f\tau_0\tau_1$ depends only on f

$\succ \mathsf{Y}\xi_f \star n_f.\pi \succ \xi_f \star \eta_f.n_f.\pi$, with $\eta_f = \mathsf{Y}\xi_f$. Therefore

$$\gamma \star f.\pi \succ f \star \tau_0^{f\pi}.\tau_1^{f\pi}.\pi$$

with $\tau_i^{f\pi} = \tau_i[\eta_f/x, n_f/n, k_\pi/k]$.

A program for the CCA (cont.)

Now $\tau_0^{f\pi}$ is simply the triple $<\eta_f, n_f, k_\pi>$. In other words

$\tau_0^{f\pi}$ stores the current state and time f, π, n_f when γ comes in head position.

$\tau_1^{f\pi}$ performs the real job : it looks at two such states $f.\pi$ and $f'.\pi'$ and compare their times n_f and $n_{f'}$. If $n_f = n_{f'}$ it does nothing.

If $n_f < n_{f'}$ (resp. $n_{f'} < n_f$) it restarts with $f' \star \tau_0^{f'\pi}.\tau_1^{f'\pi}.\pi$ (resp. $f \star \tau_0^{f\pi'}.\tau_1^{f\pi'}.\pi'$) : in each case, *the second file with the first stack*.

Thus, the main function of this program is to *update files*.

If we take the signature σ instead of the clock $\hbar$, the program can be used *to choose a suitable version of a file*.

The axiom of dependent choice is a very general and useful principle in mathematics. The program associated with it is also a very general and useful tool to update files or choose the suitable release of a file.

Zermelo-Fraenkel set theory

A first order theory. Its axioms can be classified in three groups :

1. Equality, extensionality, foundation.
2. Union, power set, substitution, infinity.
3. Choice ; possibly other axioms such as CH, GCH, large cardinals.

We can realize the first two groups by λ_c-terms,
i.e. no new instruction is necessary besides cc.
Curiously, equality and extensionality are the most difficult ones. For example,
the first axiom of equality $\forall x(x = x)$ is realized by a λ-term τ
with the reduction rule : $\tau \star t.\pi \succ t \star \tau.\tau.\pi$ (fixed point of $\lambda x \lambda f\ fxx$).
Therefore, we first consider a theory with a strong membership relation ε,
without extensionality ; in some sense, $\in$ is defined by means of ε.

ZF_ε set theory

Three binary symbols $\in, \subset$ and ε (strong membership) ; $x = y$ is $x \subset y \wedge y \subset x$.

- "Definition" of $\in$ and $\subset$:

$\forall x \forall y [x \in y \leftrightarrow (\exists z \, \varepsilon \, y) x = z]$; $\forall x \forall y [x \subset y \leftrightarrow (\forall z \, \varepsilon \, x) z \in y]$.

- Foundation : $\forall a [(\forall x \, \varepsilon \, a) F(x) \rightarrow F(a)] \rightarrow \forall a \, F(a)$ (for every formula F).
- Comprehension : $\forall a \exists b \forall x [x \, \varepsilon \, b \leftrightarrow (x \, \varepsilon \, a \wedge F(x))]$ (")
- Pair : $\forall a \forall b \exists x [a \, \varepsilon \, x \wedge b \, \varepsilon \, x]$
- Union : $\forall a \exists b (\forall x \, \varepsilon \, a)(\forall y \, \varepsilon \, x) \, y \, \varepsilon \, b$.
- Power set : $\forall a \exists b \forall x (\exists y \, \varepsilon \, b) \forall z (z \, \varepsilon \, y \leftrightarrow (z \, \varepsilon \, a \wedge F(z, x)))$ (")
- Collection : $\forall a \exists b (\forall x \, \varepsilon \, a) [\exists y \, F(x, y) \rightarrow (\exists y \, \varepsilon \, b) F(x, y)]$ (")
- Infinity : $\forall a \exists b \{a \, \varepsilon \, b \wedge (\forall x \, \varepsilon \, b) [\exists y \, F(x, y) \rightarrow (\exists y \, \varepsilon \, b) F(x, y)]\}$ (")

This theory is a *conservative extension* of ZF :

1. If $ZF_\varepsilon \vdash F$ (formula of ZF), then $ZF \vdash F$: simply replace ε by $\in$ in ZF_ε.
2. We must show that each axiom of ZF is a consequence of ZF_ε.

ZF_ε set theory (cont.)

Example. $ZF_\varepsilon \vdash a \subset a$ (and thus $a = a$).

By foundation, assume $\forall x(x \,\varepsilon\, a \to x \subset x)$; this gives $\forall x(x \,\varepsilon\, a \to x = x)$, thus $\forall x[x \,\varepsilon\, a \to (\exists y \,\varepsilon\, a)\, x = y]$, i.e. $\forall x(x \,\varepsilon\, a \to x \in a)$, and therefore $a \subset a$.

Now, we define *realizability models for ZF_ε*, which will therefore be also realizability models for ZF. We only need to define $\|F\|$ for atomic formulas F.

Of course, we start with a model of ZF, and we take as atomic formulas : $a \notin b$, $a \notin b$ and $a \subset a$. Then define : $\|a \notin b\| = \{\pi \in \Pi;\ (a, \pi) \in b\}$.

We check that all the axioms of ZF_ε, except the first, are realized, without knowing the precise definition of $\|a \notin b\|$, $\|a \subset b\|$, simply because they are defined in ZF.

Foundation. $\mathsf{Y} \Vdash \forall a[\forall x(F(x) \to x \notin a) \to \neg F(a)] \to \forall a \neg F(a)$.

This explains why we find $\mathsf{Y}\,\lambda x \lambda f\, f x x \Vdash \forall x(x = x)$.

ZF_ε set theory (cont.)

Comprehension. For every set a and every formula $F(x)$, set :

$b = \{(x, t.\pi);\ (x,\pi) \in a,\ t \Vdash F(x)\}$. We easily get $\|x \notin b\| = \|F(x) \to x \notin a\|$.

It follows that $(I, I) \Vdash \forall x[x \notin b \leftrightarrow (F(x) \to x \notin a)]$.

Other axioms of ZF_ε are realized in the same way. For example :

Collection. Let a be a set, $Cl(a)$ its transitive closure and $F(x, y)$ a formula.

We set $b = \bigcup\{\Phi(x, t) \times Cl(a);\ x \in Cl(a),\ t \in \Lambda_c\}$ with

$\Phi(x, t) = \{y$ of minimum rank ; $t \Vdash F(x, y)\}$, or $\emptyset$ if there is no such y.

We show that $\|\forall y(F(x, y) \to x \notin a)\| \subset \|\forall y(F(x, y) \to y \notin b)\|$. Indeed :

suppose $t \Vdash F(x, y)$, $(x,\pi) \in a$. Then $x, \pi \in Cl(a)$, and therefore :

$(y', \pi) \in b$ for some $y' \in \Phi(x, t)$; it follows that $t \Vdash F(x, y')$ and $\pi \in \|y' \notin b\|$.

Therefore $t.\pi \in \|\forall y(F(x, y) \to y \notin b)\|$.

We have proved that $I \Vdash \forall y(F(x, y) \to y \notin b) \to \forall y(F(x, y) \to x \notin a)$.

ZF_ε set theory (cont.)

We must now realize the first axioms of ZF_ε and therefore define the truth values of the atomic formulas : $\|a \notin b\|$, $\|a \subset b\|$, where a, b vary in *a given model of ZFC*.

It would be nice to have :

$\|a \notin b\| = \|\forall z(z \subset a, a \subset z \to z \notin b)\|$ and $\|a \subset b\| = \|\forall z(z \notin b \to z \notin a)\|$

because we should deduce immediately that I realizes the axioms we need.

Now $\|c \notin a\| = \emptyset$ if $rk(a) \leq rk(c)$. Thus, the above equations may be written as :

$\|a \notin b\| = \bigcup_{rk(c)<rk(b)} \|(c \subset a, a \subset c \to c \notin b)\|$

$\|a \subset b\| = \bigcup_{rk(c)<rk(a)} \|(c \notin b \to c \notin a)\|$ i.e.

$\|a \notin b\| = \bigcup_{rk(c)<rk(b)} \Phi(a, b, c, \|c \subset a\|, \|a \subset c\|)$

$\|a \subset b\| = \bigcup_{rk(c)<rk(a)} \Psi(a, b, c, \|c \subset a\|, \|a \subset c\|)$

where Φ, Ψ are functionals defined in ZF.

We simply observe now that this is a correct inductive definition on the ordered pair of ordinals : $(rk(a) \cup rk(b), rk(a) \cap rk(b))$.

ZF_ε set theory (cont.)

Remark. It is also possible to *define* the relations $x \notin y$, $x \subset y$ by formulas with the only symbol $\notin$ and then to *prove* the first axioms of ZF_ε from the others axioms.
We cannot use induction to define these relations, because ordinals are not definable in ZF_ε. But we can use *coinduction.*
Anyway, this method gives complicated λ-terms for the first axioms of ZF_ε, so that we prefer the above method.

Remark. The definition of $t \Vdash x \notin y$ and $t \Vdash x \subset y$ is very similar to the defintion of forcing. In fact, the generic models of set theory, which are defined in forcing, are particular cases of realizability models.
Thus, the theory presented here gives completely new models of set theory.
The fact that forcing is a case of realizability, is used to find programs associated with the *axiom of choice* and the *continuum hypothesis*. We build a model by combining both methods ; we call this *iterated realizability* by analogy with *iterated forcing*.

The full axiom of choice

We get a program for the axiom of dependent choice in the same way as in Analysis.
The problem for the *full axiom of choice* is more difficult. It has been solved very recently (not yet published). As a bonus, we get also the *continuum hypothesis*.
The proof is too long to be given here ; the result is as follows :
we need two new instructions χ and χ' which appear inside
two very complex λ-terms, together with cc and the clock (or the signature).
The behaviour of these programs is not yet understood.
These new instructions χ, χ' work on the *bottom of the stack*.
Their reduction rules is as follows :

$$\chi \star t.\tau.t_1 \dots t_n.\pi_0 \succ t \star t_1 \dots t_n.\tau.\pi_0$$
$$\chi' \star t.t_1 \dots t_n.\tau.\pi_0 \succ t \star \tau.t_1 \dots t_n.\pi_0$$

where π_0, as before, is a marker for the bottom of the stack.

The axiom of choice (cont.)

In order to understand the behaviour of these new instructions, we consider processes of the form $<t \star \pi,\ \tau>$ where τ is a closed term.

The execution rules are as follows :

$<tu \star \pi,\ \tau> \succ <t \star u.\pi,\ \alpha_0\tau>$ $\qquad$ $<\lambda x\, t \star u.\pi, \tau> \succ <t[u/x] \star \pi,\ \alpha_1\tau>$

$<\mathsf{cc} \star t.\pi,\ \tau> \succ <t \star \mathsf{k}_\pi.\pi,\ \alpha_2\tau>$ $\qquad$ $<\mathsf{k}_\pi \star t.\rho,\ \tau> \succ <t \star \pi,\ \alpha_3\tau>$

$<\chi \star t.\tau.t_1 \ldots t_n.\pi_0,\ \tau'> \succ <t \star t_1 \ldots t_n.\tau'.\pi_0,\ \tau>$

$<\chi' \star t.t_1 \ldots t_n.\tau'.\pi_0,\ \tau> \succ <t \star \tau.t_1 \ldots t_n.\pi_0,\ \tau'>$

The α_i are fixed closed terms, which we shall not write explicitly here.

In fact, we get a parallel execution ; χ and χ' are communication instructions.

The standard realizability model in analysis

Realizability models are obtained by choosing a set $\perp\!\!\!\perp$ which must be *saturated* and *coherent*. Let $\perp\!\!\!\perp^c$ be the complement of $\perp\!\!\!\perp$. The conditions on $\perp\!\!\!\perp^c$ are :

$p \in \perp\!\!\!\perp^c$, $p \succ q \Rightarrow q \in \perp\!\!\!\perp^c$ (saturation) ;

for every proof-like term ξ there is a stack π s.t. $\xi \star \pi \in \perp\!\!\!\perp^c$ (coherence).

Let $\xi \mapsto \pi_\xi$ be a one-one map from proof-like terms into stack constants.

If $\xi \star \pi_\xi \in \perp\!\!\!\perp^c$ for every ξ, the set $\perp\!\!\!\perp$ is obviously coherent. The set of all processes obtained by executing $\xi \star \pi_\xi$ will be called the *thread* generated by the proof-like term ξ, and $\xi \star \pi_\xi$ is the *boot* of this thread.

Thus, $\perp\!\!\!\perp^c =$ the union of all threads is a somewhat canonical way to define $\perp\!\!\!\perp$.

We have thus $\perp\!\!\!\perp^c = \{p;$ there is a proof-like ξ s.t. $\xi \star \pi_\xi \succ p\}$

We call this model the *standard realizability model*.

Nevertheless, as we shall see, it contains non standard integers.

A generic non-standard integer

Let $n \mapsto \xi_n$ be a fixed recursive enumeration of proof-like terms. We define a unary predicate G by setting :

$\|Gn\| = \Pi_n$ i.e. the set of stacks which end with the constant π_{ξ_n}.

We assume there is no instruction which changes the stack constant. It follows that π_ξ is the only one which appears in the thread $\xi \star \pi_\xi$.

Since $\bigcup_n \Pi_n = \Pi$, we get $\|\forall x\, Gx\| = \Pi$, thus $I \Vdash \neg\forall x\, Gx$.

We show that Gn is realized for each integer n. Indeed suppose that :

$\delta\delta 0 \not\Vdash Gn$ and $\delta\delta 1 \not\Vdash Gn$ with $\delta = \lambda x\, xx$.

Then, $\xi_n \star \pi_{\xi_n} \succ \delta\delta 0 \star \pi_0$ and $\xi_n \star \pi_{\xi_n} \succ \delta\delta 1 \star \pi_1$ which is impossible.

It follows that the predicate G contains every *standard* integer, but not every individual.

Does it contain every integer ?

A generic non-standard integer (cont.)

Let ς (for "self") be a new instruction with the following reduction rule :

$$\varsigma \star t.\pi \succ t \star \underline{n}.\pi \text{ ; } n \text{ is the integer such that } \pi \in \Pi_{\xi_n}.$$

Then $$\varsigma \Vdash \forall x(\mathsf{int}(x) \to Gx) \to \bot.$$

Indeed, if $t \Vdash \forall x(\mathsf{int}(x) \to Gx)$ and $\pi \in \Pi_n$, then $\underline{n} \Vdash \mathsf{int}(n)$ and $\pi \in \|Gn\|$.

Thus $t \star \underline{n}.\pi \in \perp\!\!\!\perp$ and $\varsigma \star t.\pi \in \perp\!\!\!\perp$.

It follows that the predicate $\neg G$ contains at least one *non-standard integer*.

In the next slide, we show that the formula $\forall x \forall y\{\neg Gx, x \neq y \to Gy\}$ is realized.

Thus, the predicate $\neg Gx$ consists in *exactly one* individual

and it is a non-standard integer. We call it the *generic integer*.

We add a new individual constant g to our language, and replace Gx with $x \neq \mathsf{g}$.

The non-standard proof-like term ξ_{g} has remarkable properties.

A generic non-standard integer (cont.)

Lemma. If $\xi \star k_\pi.\rho \in \perp\!\!\!\perp$ for all $\pi \in \|A\|$ and $\rho \in \|B\|$, then $\gamma\xi \Vdash \neg A \to B$ with $\gamma = \lambda x \lambda y \, \mathsf{cc} \lambda h \, y \, \mathsf{cc} \, h \circ x$.

The hypothesis gives $\mathsf{cc} \, k_\rho \circ \xi \Vdash A$. If $t \Vdash \neg A$, we get $t \, \mathsf{cc} k_\rho \circ \xi \Vdash \bot$, therefore $\mathsf{cc} \, \lambda h \, t \, \mathsf{cc} h \circ \xi \star \rho \in \perp\!\!\!\perp$ for every $\rho \in \|B\|$. Thus, $\gamma\xi \star t.\rho \in \perp\!\!\!\perp$, because it reduces to this process. QED

We want to show that $\forall x \forall y [\neg Gx, x \neq y \to Gy]$ is realized. By the preceding lemma, it is sufficient to show that :

$0 \star k_\pi.t.\rho \in \perp\!\!\!\perp$ with $0 = \lambda x \lambda y \, y$, $\pi \in \|Gn\| = \Pi_n$, $\rho \in \|Gp\| = \Pi_p$, $t \Vdash n \neq p$.

If $n \neq p$, this process is in no thread, because it contains two different stack constants π_{ξ_n} and π_{ξ_p}. If $n = p$, then $t \Vdash \bot$ and $0 \star k_\pi.t.\rho \succ t \star \rho$, hence the result. QED

Conclusion

The conclusion is that we can translate *every mathematical proof*
into a program. We can execute this program in a lazy λ-calculus machine
extended with only four new instructions : cc, σ (or $\hbar$), χ and χ'.
This machine can be implemented rather easily.

The challenge, now, is to understand all these programs,
first of all the ones we obtained for the axioms of ZFC.
It is very plausible that we shall find, in this way, programs analogous
to the core of an operating system like Unix.
This would give a method to implement such a core on a very firm basis.

References

1. **S. Berardi, M. Bezem, T. Coquand** *On the computational content of the axiom of choice.* J. Symb. Log. 63, pp. 600-622 (1998).
2. **U. Berger, P. Oliva** *Modified bar recursion and classical dependent choice.* Preprint.
3. **J.-L. Krivine** *Typed lambda-calculus in classical Zermelo-Fraenkel set theory.*
Arch. Math. Log. 40, 3, pp. 189-205 (2001)
4. **J.-L. Krivine** *Dependent choices, 'quote' and the clock.*
Th. Comp. Sc. 308, pp. 259-276 (2003)
5. **J.-L. Krivine** *Realizability in classical logic.*
To appear in Panoramas et Synthèses. Société mathématique de France.

Pdf files at http://www.pps.jussieu.fr/~krivine