An enumeration-free proof of Gödel's completeness theorem with side effects

(work in progress)

Hugo Herbelin

3 February 2012

FATPA 2012

Belgrade

(revised 5/2/12)

The proof-as-programs correspondence

An intuitionistic proof is a purely functional program [Curry, 1958, Howard, 1969]

- a proof of $A \to B$ is a program of the form $x => p$ where x has type A and p has type B
- a proof of $A \wedge B$ is the same as the pair of an element of A and of an element of B
- deriving B from $A \to B$ and A is the same as applying the given proof-function of type $A \to B$ to the given element of type A
- etc

The proof-as-program correspondence for classical logic

A classical proof is a program with control [Griffin, 1990]

`callcc/throw` (ML languages) or `call-with-current-continuation` (Scheme) can be used to prove $\neg\neg A \to A$

Exception mechanisms (ML languages, Java, ...), or even `setjump/longjump` (in C) provide a weak form of classical logic: they can be used to prove $\neg\neg A \to A$ when A is a formula without implication nor universal quantification type (this latter principle can be seen as a variant of Markov's principle for predicate logic) [HH, 2010]

Is classical logic a side effect?

From the point of view of pure functional programming (Haskell, or the core of F#), control operators implement side effects

To deal with side effects in Haskell, the common approach is to reason in a "monad" and to extract afterwards pure contents by "running" the monad

Can we imagine adding other side effects to logic that could be cleared by "running" the monad?

Example of monadic programming: the state monad

To simulate the ability to read and write a global memory of type S without using any memory at all, one sets

$$T_{st}(A) \triangleq S \to S \times A$$

and, whenever one wants to write a program of type A, we actually write a program of type $T_{st}(A)$

To be able to write any functional program in a "monad" T, we need two operations[1]

η	:	$A \to T(A)$	to enter the monad
*	:	$(A \to T(B)) \to T(A) \to T(B)$	to apply functions
$\mathtt{run}$	:	$T(base) \to base$	to "run" the monad on base types

which, in the case of the state monad, are implemented by:

$$\begin{array}{lcl} \eta\, x & \triangleq & \lambda s.(s, x) \\ f^*\, x & \triangleq & \lambda s.\mathbf{let}\ (s', x') = x s\ \mathbf{in}\ f\, x'\, s' \\ \mathtt{run}\, x & \triangleq & \mathbf{let}\ (\_, x') = x s_0\ \mathbf{in}\ x' \end{array}$$

where s_0 is the initial value of the memory.

[1] These operations have to satisfy the equations $\eta^* x = x$, $f^*(\eta x) = f x$, $g^*(f^* x) = (g^* \circ f)^* x$.

What did we gain?

What we gained is that in the state monad, the following two new operations are available for free!

$$\begin{array}{lcl} \texttt{read} & : & \texttt{unit} \to T_{st}(S) \\ \texttt{write} & : & S \to T_{st}(\texttt{unit}) \end{array}$$

which are implemented by:

$$\begin{array}{lcl} \texttt{read}\,() & \triangleq & \lambda s.(s, s) \\ \texttt{write}\,s' & \triangleq & \lambda \_.(s', ()) \end{array}$$

Monadic-style vs direct-style: the case of classical logic

Remark: double-negation translation $T_{cont}(A) \triangleq \neg\neg A$ is the "continuation" monad that allows to reason classically (i.e. using $\neg\neg A \to A$) inside intuitionistic logic.

Conversely, classical logic can be seen as reasoning *intuitionistically* within the continuation monad, but doing it in *direct-style*.

Monadic-style vs direct-style: the case of Markov's principle

There is a weak form of the continuation monad, namely the "exception" monad $T_{exc}(A) \triangleq A \vee E$ for supporting handing and raising exceptions in some type E.

This is enough to provide a predicate logic variant of Markov's principle (i.e. $\neg\neg A \to A$ for A without implication nor universal quantification).

Monadic-style vs direct-style

It turns out that some form of memory assignment is used in logic: both of

- Kripke semantics based translations

- Cohen's forcing translation

correspond to providing *monotonically* memory assignment in *monadic style*.

Kripke semantics based translations (i.e. $T_{kr}(A)(x) \triangleq \forall x' \geq x\, A(x')$ for x ranging over $\mathcal{W}$) is a dependent form of the environment monad ($T_{env}(A) \triangleq \mathcal{W} \to A$) known to provide Lisp-style dynamic bindings.

Cohen's forcing translation ($T_{forc}(A)(x) \triangleq \forall x' \geq x \exists x'' \geq x'\, A(x'')$ for x ranging over some domain S) is a dependent form of the state monad ($T_{st}(A) \triangleq S \to S \times A$).

What if we try to design a logical system that provides such kinds of memory assignment in *direct-style*?

Towards a logic with side effects

We shall now describe a (sound) extension of second-order intuitionistic arithmetic with the following two effects:

- exceptions (i.e. a weak form of classical logic)

- monotonically updatable memory

Note: Soundness comes by embedding into second-order intuitionistic arithmetic using a combination of Kripke-style (dependent) environment monad, Friedman-style exception monad, and Coquand-Hofmann's translation [1999].

Logical rules providing delimited *direct-style* exceptions and monotone memory updates

A rule to simultaneously declare a memory x with initial value t and monotonically updatable along a preorder $\geq$ and an handler of exceptions of name $\hat{\alpha}$ in type U (this delimits the *direct-style* use of the memory update and exception effects):

$$\frac{\Gamma, \hat{\alpha} : \neg U(x), b : x \geq t \vdash q : U(x) \qquad \Gamma \vdash r : \mathtt{preorder}\ \geq \qquad \Gamma \vdash s : \mathtt{monotone}_{\geq}\, U(x)}{\Gamma \vdash \mathtt{set}\, x := t\, \mathtt{as}\, b\, \mathtt{using}\, (r, s)\, \mathtt{in}\, \#_{\hat{\alpha}}\, q : U(t)}\ \textsc{seteff}$$

A rule to monotonically update the value of the memory x (this provides in *direct-style* what Kripke-style translation T_{kr} provides in monadic style):

$$\frac{\Gamma, b : x \geq u(x') \vdash q : U(x) \qquad \Gamma \vdash r : u(x) \geq x \qquad (\hat{\alpha} : \neg U(x)) \in \Gamma \qquad x' \text{ fresh}}{\Gamma \vdash \mathtt{update}\, x := u(x)\, \mathtt{as}\, (x', b)\, \mathtt{by}\, r\, \mathtt{in}\, \#_{\hat{\alpha}}\, q : U(u(x))}\ \textsc{update}$$

A rule to raise an exception of name $\hat{\alpha}$ in type U (this provides in *direct-style* what the exception monad T_{exc} provides in monadic style):

$$\frac{\Gamma \vdash p : U(x) \qquad (\hat{\alpha} : \neg U(x)) \in \Gamma}{\Gamma \vdash \mathtt{raise}_{\hat{\alpha}}\, p : B}\ \textsc{abort}$$

Application: an enumeration-free proof of Gödel's completeness

(Weak) completeness: if A is true in all models $\mathcal{M}$, then A is provable

Let C_0 be a formula and prove $\dot{\neg}C_0 \dot{\vdash} \dot{\bot}$. The idea is to consider an updatable variable Γ initialized to $\dot{\neg}C_0$ with $\Gamma \dot{\vdash} \dot{\bot}$ as objective and to take the syntactic model $\mathcal{M}_0$ defined by $A \in \mathcal{M}_0$ iff $\Gamma \dot{\vdash} A$. We now have to prove the following:

$$\begin{array}{lll}
(A \dot{\to} B) \in \mathcal{M}_0 & \text{iff} & A \in \mathcal{M}_0 \to B \in \mathcal{M}_0 \\
(\dot{\forall} x\, A) \in \mathcal{M}_0 & \text{iff} & \forall t\, A[t/x] \in \mathcal{M}_0 \\
\dot{\bot} \in \mathcal{M}_0 & \text{iff} & \bot \\
\dot{\neg}\dot{\neg} A \in \mathcal{M}_0 & \text{iff} & A \in \mathcal{M}_0
\end{array}$$

i.e.

$$\begin{array}{lll}
\Gamma \dot{\vdash} A \dot{\to} B & \text{iff} & \Gamma \dot{\vdash} A \to \Gamma \dot{\vdash} B \\
\Gamma \dot{\vdash} (\dot{\forall} x\, A) & \text{iff} & \forall t\, \Gamma \dot{\vdash} A[t/x] \\
\Gamma \dot{\vdash} \dot{\bot} & \text{iff} & \bot \\
\Gamma \dot{\vdash} \dot{\neg}\dot{\neg} A & \text{iff} & \Gamma \dot{\vdash} A
\end{array}$$

where Γ is a monotonically *updatable* variable.

Application: an enumeration-free proof of Gödel's completeness

The two statements that use effects are:

$$\Leftarrow_{\dot\to} \;:\; (\Gamma \dot\vdash A \;\to\; \Gamma \dot\vdash B) \;\to\; \Gamma \dot\vdash A \dot\to B$$
$$\Rightarrow_{\dot\bot} \;:\; \Gamma \dot\vdash \dot\bot \;\to\; \bot$$

The proofs are:

$$\begin{aligned}
\Leftarrow_{\dot\to} f \;&\triangleq\; \dot{\mathtt{IMPI}}_{\Gamma,A,B}\, \dot{\mathtt{DN}}_{(\Gamma,A),B} \\
&\qquad \mathtt{update}\,\Gamma := (\Gamma, A, \dot\neg B)\,\mathsf{as}\,(\Gamma', b)\,\mathsf{by}\,r_0\,\mathtt{in} \\
&\qquad\quad \#_{\hat\alpha}\,\dot{\mathtt{IMPE}}_{\Gamma,B,\dot\bot}(\dot{\mathtt{AX}}_{(\Gamma',A),\dot\neg B,\Gamma}\, b,\, f\,(\dot{\mathtt{AX}}_{\Gamma',A,\Gamma}\,\phi(b))) \\
\Rightarrow_{\dot\bot} p \;&\triangleq\; \mathtt{raise}_{\hat\alpha}\, p
\end{aligned}$$

where r_0 proves $\Gamma \subset (\Gamma, A, \dot\neg B)$ while b of type $(\Gamma', A, \dot\neg B) \subset \Gamma$ and $\phi(b)$, obtained from b and of type $(\Gamma', A) \subset \Gamma$, respectively justify the correctness of the axiom rules. The relevant excerpt of inference rules of the object language is:

$$\frac{p : (\Delta, A \dot\vdash B)}{\dot{\mathtt{IMPI}}_{\Delta,A,B}\, p : (\Delta \dot\vdash A \dot\to B)} \qquad \frac{p : ((\Delta', A) \subset \Delta)}{\dot{\mathtt{AX}}_{\Delta',A,\Delta}\, p : (\Delta \dot\vdash A)} \qquad \frac{p : (\Delta, \dot\neg A \dot\vdash \dot\bot)}{\dot{\mathtt{DN}}_{\Delta,A}\, p : (\Delta \dot\vdash A)} \qquad \frac{p : (\Delta \dot\vdash A \dot\to B) \quad q : (\Delta \dot\vdash A)}{\dot{\mathtt{IMPE}}_{\Delta,A,B}\,(p, q) : (\Delta \dot\vdash B)}$$

Application: an enumeration-free proof of Gödel's completeness

In short, if we call H the proof of $\forall \mathcal{M}\ \mathit{model}(\mathcal{M}) \to C_0 \in \mathcal{M}$, o_0 and s_0 proofs respectively of the ordering of $\subset$ and of $(\Gamma \subset \Gamma') \to (\Gamma \dot{\vdash} A) \to (\Gamma' \dot{\vdash} A)$ and ok_0 the combination of $\Leftarrow_{\dot{\to}}$, $\Leftarrow_{\dot{\forall}}$, $\Leftarrow_{\dot{\bot}}$, $\Rightarrow_{\dot{\to}}$, $\Rightarrow_{\dot{\forall}}$ and $\Rightarrow_{\dot{\bot}}$ that shows that $\mathcal{M}_0$ is a model, then the proof of $\dot{\vdash} C_0$ written as a program is

$$\mathtt{compl}\ H \triangleq \dot{\mathtt{DN}}_{\emptyset,C_0}\ \mathtt{set}\ \Gamma := \dot{\neg} C_0\ \mathtt{as}\ b\ \mathtt{using}\ (o_0, s_0)\ \mathtt{in}\ \#_{\hat{\alpha}}\ \dot{\mathtt{IMPE}}_{\Gamma,C_0,\dot{\bot}}\ (\dot{\mathtt{AX}}_{\emptyset,\dot{\neg}C_0,\Gamma}\ b, H\ \mathcal{M}_0\ ok_0)$$

This proof is constructive... we can compute with it using exceptions and assignment!

The proof is also extensible to strong completeness: if A is true in all models $\mathcal{M}$ satisfying theory $\mathcal{T}$, then A is provable in some finite subset of $\mathcal{T}$.

Application: an enumeration-free proof of Gödel's completeness

Alternatively, the previous proof expresses in direct-style that the completeness theorem holds for a model defined from its value on atoms by

$$\begin{array}{lcl} \Gamma \vDash_{\mathcal{M}} \dot{\bot} & \triangleq & \bot \\ \Gamma \vDash_{\mathcal{M}} A \dot{\to} B & \triangleq & \forall \Gamma' \supset \Gamma \; \Gamma' \vDash_{\mathcal{M}} A \to \Gamma' \vDash_{\mathcal{M}} B \\ \Gamma \vDash_{\mathcal{M}} \forall x \, A & \triangleq & \forall t \in Dom(\mathcal{M}) \; \Gamma \vDash_{\mathcal{M}} A[t/x] \end{array}$$

and such that $\Gamma \vDash_{\mathcal{M}} \neg\neg A$ implies $\Gamma \vDash_{\mathcal{M}} A$.

We recognise here that the proof with effects is similar to a *direct-style* formulation of the completeness wrt Kripke semantics for the negative fragment of intuitionistic logic, this fragment where intuitionistic logic precisely coincides with classical logic (in particular atoms are taken prefixed by a double negation).