An excursion Into the Proofs-as-Programs Correspondence

Hugo Herbelin

4 February 2011

4th workshop on Formal and Automated Theorem Proving and Applications

Belgrade

Outline

- Proofs, truth and computation
- Brouwer-Heyting-Kolmogorov's intuitionism
- Curry and Howard's revelations
- Martin-Löf's exploitation of Howard's revelation
- Classical logic: Griffin's revelation
- Writing sequent calculus proofs as programs

Proofs, truth and computation

$$\forall pqr\ (p \Rightarrow q \Rightarrow r) \Rightarrow (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$$

$$\forall nmp\ 3(m + 2n) - 4m = 6n - m$$

$$\forall n\ 0 + 2 \times n = n + n$$

$$\forall n > 2\ \forall pqr\ (p^n + q^n \neq r^n)$$

$$(\forall n\ \exists pq\,(prime(p) \wedge prime(q) \wedge p+q = 2n)\ \vee\ (\exists n\ \forall pq\,(prime(p) \wedge prime(q) \Rightarrow p+q \neq 2n)$$

Brouwer-Heyting-Kolmogorov intuitionism (made explicit in the 30's)

A proof of $A \wedge B$ is given by presenting a proof of A and a proof of B

A proof of $A \vee B$ is given by presenting either a proof of A or a proof of B

A proof of $A \Rightarrow B$ is a construction which transforms any proof of A into a proof of B

A proof of $\exists x\, A$ is given by presenting a proof of $A(n)$ for some n

A proof of $\forall x\, A$ is given by a construction which presents a proof of $A(n)$ for any n

Kleene's realisability (1945)

Indeed, from a proof of $\vdash \forall n \exists m\, P(n,m)$ one can extract a computable and terminating function $f : \mathbb{N}_0 \Rightarrow \mathbb{N}_0$ such that for all n, $\vdash P(n, f(m))$.

Indeed, from a proof of $\vdash \forall n(\neg P(n) \vee P(n))$ one can extract a computable and terminating function $f : \mathbb{N}_0 \Rightarrow \{0,1\}$ such that for all n, $f(n) = 0$ implique $\vdash \neg P(n)$ and $f(n) = 1$ implique $\vdash P(n)$.

Curry's revelation (around 1958)

The axiom of Hilbert-style logic are the same as the axioms of combinatory logic

$$
\begin{array}{ll}
S_{pqr} & : \ (p \Rightarrow q \Rightarrow r) \Rightarrow (p \Rightarrow q) \Rightarrow (p \Rightarrow r) \\
K_{pq} & : \ p \Rightarrow q \Rightarrow p \\
MP_{pq} & : \ \text{from } p \Rightarrow q \text{ and } p \text{ one gets } q
\end{array}
$$

Natural deduction and λ-calculus (Howard, 1968)

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \Rightarrow_I (a, p) \mid \Rightarrow_E (p, q) \\
& | & \forall_I(x, p) \mid \forall_E(p, t) \\
& | & \exists_I(t, p) \mid \exists_E(p, x, a, q) \\
& | & \vee_I^i(p) \mid \vee_E(p, a_1, p_1, a_2, p_2) \\
& | & \wedge_I(p, q) \mid \wedge_E^i(p) \\
& | & \bot_E p \\
& | & \top_I
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \Rightarrow B \\
& | & \forall x\, A(x) \\
& | & \exists x\, A(x) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
$$

Natural deduction and λ-calculus (Howard, 1968)

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \Rightarrow_I (a, p) \,|\Rightarrow_E (p, q) \\
& | & \forall_I(x, p) \mid \forall_E(p, t) \\
& | & \exists_I(t, p) \mid \exists_E(p, x, a, q) \\
& | & \vee_I^i(p) \mid \vee_E(p, a_1, p_1, a_2, p_2) \\
& | & \wedge_I(p, q) \mid \wedge_E^i(p) \\
& | & \bot_E p \\
& | & \top_I
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \Rightarrow B \\
& | & \forall x\, A(x) \\
& | & \exists x\, A(x) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
$$

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\, q \\
& | & \lambda x.p \mid p\, t \\
& | & (t, p) \mid \texttt{dest}\ p\ \texttt{as}\ (x, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \,|\, a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \to B \\
& | & \Pi x\, A(x) \\
& | & \Sigma x\, A(x) \\
& | & A + B \\
& | & A \times B \\
& | & \texttt{unit} \\
& | & \texttt{void}
\end{array}
$$

Natural deduction and λ-calculus (Howard, 1968)

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\,q \\
& | & \lambda x.p \mid p\,t \\
& | & (t, p) \mid \texttt{dest}\ p\ \texttt{as}\ (x, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \Rightarrow B \\
& | & \forall x\, A(x) \\
& | & \exists x\, A(x) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
$$

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \Rightarrow_I (a, p) \mid \Rightarrow_E (p, q) \\
& | & \forall_I(x, p) \mid \forall_E(p, t) \\
& | & \exists_I(t, p) \mid \exists_E(p, x, a, q) \\
& | & \vee_I^i(p) \mid \vee_E(p, a_1, p_1, a_2, p_2) \\
& | & \wedge_I(p, q) \mid \wedge_E^i(p) \\
& | & \bot_E p \\
& | & \top_I
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \to B \\
& | & \Pi x\, A(x) \\
& | & \Sigma x\, A(x) \\
& | & A + B \\
& | & A \times B \\
& | & \texttt{unit} \\
& | & \texttt{void}
\end{array}
$$

Natural deduction and λ-calculus (Howard, 1968)

$$
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\,q \\
& | & \lambda x.p \mid p\,t \\
& | & (t, p) \mid \texttt{dest}\ p\ \texttt{as}\ (x, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
\qquad
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \Rightarrow B \\
& | & \forall x\, A(x) \\
& | & \exists x\, A(x) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
$$

$$\frac{(a : A) \in \Gamma}{\Gamma \vdash a : A}$$

$$\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : A \Rightarrow B} \qquad \frac{\Gamma \vdash p : A \Rightarrow B \qquad \Gamma \vdash q : A}{\Gamma \vdash p\, q : B}$$

$$\frac{\Gamma \vdash p : A(x) \qquad x \text{ fresh}}{\Gamma \vdash \lambda x.p : \forall x\, A(x)} \qquad \frac{\Gamma \vdash p : \forall x\, A(x)}{\Gamma \vdash p\, t : A(t)}$$

$$\frac{\Gamma \vdash p : A(t)}{\Gamma \vdash (t, p) : \exists x\, A(x)} \qquad \frac{\Gamma \vdash p : \exists x\, A(x) \qquad \Gamma, a : A(x) \vdash q : B \qquad x \text{ fresh}}{\Gamma \vdash \texttt{dest}\ p\ \texttt{as}\ (x, a)\ \texttt{in}\ q : B}$$

$$\frac{\Gamma \vdash p_1 : A_1 \qquad \Gamma \vdash p_2 : A_2}{\Gamma \vdash (p_1, p_2) : A_1 \wedge A_2} \qquad \frac{\Gamma \vdash p : A_1 \wedge A_2}{\Gamma \vdash \pi_1 p : A_i}$$

$$\frac{\Gamma \vdash p : A_i}{\Gamma \vdash \iota_i(p) : A_1 \vee A_2} \qquad \frac{\Gamma \vdash p : A_1 \vee A_2 \qquad \Gamma, a_1 : A_1 \vdash p_1 : B \qquad \Gamma, a_2 : A_2 \vdash p_2 : B}{\Gamma \vdash \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \,|\, a_2 \Rightarrow p_2] : B}$$

$$\frac{}{\Gamma \vdash () : \top} \qquad \frac{\Gamma \vdash p : \bot}{\Gamma \vdash \texttt{abort}\ p : C}$$

Example

$$\lambda x.\lambda a.\texttt{dest}\ a\ \texttt{as}\ (y, b)\ \texttt{in case}\ b\ \texttt{of}\ [b_1 \Rightarrow (y, b_1) \,|\, b_2 \Rightarrow (y, b_2)]$$

is a proof of

$$\forall x\,(\exists y\,[B_1(x, y) \vee B_2(x, y)] \Rightarrow [\exists y\,B_1(x, y)] \vee [\exists y\,B_1(x, y)])$$

where

a is an hypothesis of type $\exists y\,[B_1(x, y) \vee B_2(x, y)]$

b_i is an hypothesis of type $B_i(x, y)$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

$$
\begin{array}{rcl}
A, B & ::= & P(\vec{t}) \\
& | & A \Rightarrow B \\
& | & \forall x\, A(x) \\
& | & \exists x\, A(x) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
\qquad
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\, q \\
& | & \lambda x.p \mid p\, t \\
& | & (t, p) \mid \texttt{dest}\ p\ \texttt{as}\ (x, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
$$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

Replace individuals by programs (which obey the same syntax as proofs!)

$$
\begin{array}{rcl}
A, B & ::= & P(\vec{p}) \\
& | & A \Rightarrow B \\
& | & \forall a\, A(a) \\
& | & \exists a\, A(a) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
\qquad\qquad
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\, q \\
& | & \lambda b.p \mid p\, t \\
& | & (q, p) \mid \texttt{dest}\ p\ \texttt{as}\ (b, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
$$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

Observe that $\Rightarrow$ is now a particular case of $\forall$

$$
\begin{array}{rcl}
A, B & ::= & P(\vec{p}) \\
& | & \forall a\, A(a) \\
& | & \exists a\, A(a) \\
& | & A \vee B \\
& | & A \wedge B \\
& | & \bot \\
& | & \top
\end{array}
\qquad\qquad
\begin{array}{rcl}
p, q & ::= & a \\
& | & \lambda a.p \mid p\, q \\
& | & (q, p) \mid \texttt{dest}\ p\ \texttt{as}\ (b, a)\ \texttt{in}\ q \\
& | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & (p, q) \mid \pi_i(p) \\
& | & \texttt{abort}\ p \\
& | & ()
\end{array}
$$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

Add a few interesting computational types such as $\mathbb{N}_0$ and interesting logical types such as equality

$$
\begin{array}{rcl@{\qquad}rcl}
A, B & ::= & P(\vec{p}) & p, q & ::= & a \\
& | & \forall x\, A(x) & & | & \lambda a.p \mid p\, q \\
& | & \exists x\, A(x) & & | & (q, p) \mid \texttt{dest}\ p\ \texttt{as}\ (b, a)\ \texttt{in}\ q \\
& | & A \vee B & & | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & A \wedge B & & | & (p, q) \mid \pi_i(p) \\
& | & \bot & & | & \texttt{abort}\ p \\
& | & \top & & | & () \\
& | & \mathbb{N}_0 & & | & \texttt{zero} \mid \texttt{succ} \mid \texttt{rec} \\
& | & p = p & & | & \texttt{subst} \mid \texttt{refl}
\end{array}
$$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

One can go one level further and reflect propositions into the terms: add a new type `prop` for propositions and consider defining new predicates themselves as functions to `prop` (e.g. $\lambda y.\forall x\,(x = y)$ is a predicate of type $\mathbb{N}_0 \to \texttt{prop}$ if y is a natural number)

$$
\begin{array}{rcl@{\qquad}rcl}
A, B & ::= & P(\vec{p}) & P, p, q & ::= & a \\
& | & \forall x\, A(x) & & | & \lambda a.p \mid p\, q \\
& | & \exists x\, A(x) & & | & (q, p) \mid \texttt{dest}\ p\ \texttt{as}\ (b, a)\ \texttt{in}\ q \\
& | & A \vee B & & | & \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \mid a_2 \Rightarrow p_2] \\
& | & A \wedge B & & | & (p, q) \mid \pi_i(p) \\
& | & \bot & & | & \texttt{abort}\ p \\
& | & \top & & | & () \\
& | & \mathbb{N}_0 & & | & \texttt{zero} \mid \texttt{succ} \mid \texttt{rec} \\
& | & p = p & & | & \texttt{subst} \mid \texttt{refl} \\
& | & \texttt{prop} & & | & A
\end{array}
$$

Martin-Löf's intuitionistic type theory (70's)

One can mix logic and programming languages!

Finally merge proofs/programs and types/formulae too!

$$
\begin{array}{rcll}
A, B, P, p, q & ::= & a & \\
& | & \forall x\, A(x) \;\mid\; \lambda a.p \mid p\, q & \\
& | & \exists x\, A(x) \;\mid\; (q, p) \mid \texttt{dest}\ p\ \texttt{as}\ (b, a)\ \texttt{in}\ q & \\
& | & A \vee B \;\mid\; \iota_i(p) \mid \texttt{case}\ p\ \texttt{of}\ [a_1 \Rightarrow p_1 \,|\, a_2 \Rightarrow p_2] & \\
& | & A \wedge B \;\mid\; (p, q) \mid \pi_i(p) & \\
& | & \bot \;\mid\; \texttt{abort}\ p & \\
& | & \top \;\mid\; () & \\
& | & \mathbb{N}_0 \;\mid\; \texttt{zero} \mid \texttt{succ} \mid \texttt{rec} & \\
& | & p = p \;\mid\; \texttt{subst} \mid \texttt{refl} & \\
& | & \texttt{prop} &
\end{array}
$$

and add also strong projections of $\exists$ and reason over propositions modulo evaluation of programs...
(the typing system will be in charge to sort out what is type/formula and what is term/proof)

New revelation: classical logic is constructive (Griffin 90)

One knew on the paper from Gentzen and Prawitz that cuts could be eliminated in classical logic. Griffin revealed us that this can be effective on machine too, thanks to the `callcc` and `throw` operators.

Example:

$$\texttt{callcc}_\alpha.(y_0, \lambda y.\lambda a.\texttt{throw}_\alpha(y, \lambda z.\lambda b.a)$$

is a proof of the Drinker's paradox

$$\exists x\, \forall y.(P(x) \Rightarrow P(y))$$

A "programming" language for the sequent calculus

$$
\begin{array}{lll}
t & ::= & x \mid \mu\alpha.c \mid \lambda x.t \mid (t,t) \mid \iota_i(t) \qquad \text{terms} \\
e & ::= & \alpha \mid \tilde{\mu}x.c \mid t \cdot e \mid \pi_i[e] \mid [e|e] \qquad \text{evaluation contexts} \\
c & ::= & \langle t \| e \rangle \qquad \text{commands}
\end{array}
$$

The corresponding sequent calculus has:

- two axioms

- no contraction: simulated by cuts with the axioms

- three kinds of sequents
  - terms: distinguished formula on the right
  - ev. contexts: distinguished formula on the left
  - commands: no distinguished formula

Sequent calculus

$$Ax_R \ \frac{}{\Gamma, x : A \vdash x : A \,|\, \Delta} \qquad Ax_L \ \frac{}{\Gamma \,|\, \alpha : A \vdash \alpha : A, \Delta}$$

$$\mu \ \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \,|\, \Delta} \qquad \tilde{\mu} \ \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \,|\, \tilde{\mu}x.c : A \vdash \Delta}$$

$$Cut \ \frac{\Gamma \vdash v : A \,|\, \Delta \quad \Gamma \,|\, e : A \vdash \Delta}{\langle v \| e \rangle : (\Gamma \vdash \Delta)}$$

$$\frac{\Gamma, x : A \vdash v : B \,|\, \Delta}{\Gamma \vdash \lambda x.v : A \Rightarrow B \,;\, \Delta} \qquad \frac{\Gamma \vdash v : A \,|\, \Delta \qquad \Gamma \,|\, e : B \vdash \Delta}{\Gamma \,;\, v \cdot e : A \Rightarrow B \vdash \Delta}$$

...

Example

Here is how to write a proof by resolution as a "program":

$$\dfrac{\dfrac{\dfrac{}{\neg l_1, l_2, l_3}\qquad\dfrac{}{\neg l_1, \neg l_3}}{\neg l_1, l_2}\qquad\dfrac{}{l_1, l_2}}{l_2}\qquad\dfrac{}{\neg l_2}$$

The CNF here is $A \triangleq \neg l_2 \wedge (l_1 \vee l_2) \wedge (l_1 \Rightarrow l_2 \vee l_3) \wedge \neg(l_1 \wedge l_3)$

Example

Here is how to write a proof by resolution as a "program":

$$\dfrac{\dfrac{}{A, l_2 \vdash \bot} \qquad \dfrac{\dfrac{}{A \vdash l_1, l_2} \qquad \dfrac{\dfrac{}{A, l_1 \vdash l_2, l_3} \qquad \dfrac{}{A, l_1, l_3 \vdash \bot}}{A, l_1 \vdash l_2}}{A \vdash l_2}}{A \vdash \bot}$$

The CNF here is $A \triangleq \neg l_2 \wedge (l_1 \vee l_2) \wedge (l_1 \Rightarrow l_2 \vee l_3) \wedge \neg(l_1 \wedge l_3)$

Example

Here is how to write a proof by resolution as a "program":

$$\dfrac{\dfrac{}{a : A, a_2 : l_2 \vdash l_2} \qquad \dfrac{\dfrac{}{a : A \vdash \alpha_1 : l_1, \alpha_2 : l_2} \qquad \dfrac{\dfrac{}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2, \alpha_3 : l_3} \qquad \dfrac{}{a : A, a_1 : l_1, a_3 : l_3 \vdash \bot}}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2}}{a : A \vdash \alpha_2 : l_2}}{a : A \vdash \bot}$$

The CNF here is $A \triangleq \neg l_2 \wedge (l_1 \vee l_2) \wedge (l_1 \Rightarrow l_2 \vee l_3) \wedge \neg(l_1 \wedge l_3)$

Example

Here is how to write a proof by resolution as a "program":

$$\dfrac{\dfrac{}{a : A, a_2 : l_2 \vdash l_2} \qquad \dfrac{\dfrac{}{a : A \vdash \alpha_1 : l_1, \alpha_2 : l_2} \qquad \dfrac{\dfrac{}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2, \alpha_3 : l_3} \qquad \dfrac{}{a : A, a_1 : l_1, a_3 : l_3 \vdash \bot}}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2}}{a : A \vdash \alpha_2 : l_2}}{a : A \vdash \bot}$$

The CNF here is $A \triangleq \neg l_2 \wedge (l_1 \vee l_2) \wedge (l_1 \Rightarrow l_2 \vee l_3) \wedge \neg(l_1 \wedge l_3)$

A clause, say the third clause $\neg l_1, l_2, l_3$, is seen as a proof of $a : A, a_1 : l_1 \vdash \alpha_2 : l_2, \alpha_3 : l_3$. This proof is $c_3 \triangleq \langle a \| \pi_3 \cdot a_1 \cdot [\alpha_2 | \alpha_3] \rangle$

The whole proof is a proof of $\neg A$:

$$\lambda a. \langle \mu\alpha_2. \langle \mu\alpha_1. c_2 \| \tilde{\mu} a_1. \langle \mu\alpha_3. c_3 \| \tilde{\mu} a_3. c_4 \rangle \rangle \| \tilde{\mu} a_2. c_1 \rangle$$

By reducing the cuts, one would get a normal proof of $\neg A$

Example

Here is how to write a proof by resolution as a "program":

$$\dfrac{\dfrac{}{a : A, a_2 : l_2 \vdash l_2} \qquad \dfrac{\dfrac{}{a : A \vdash \alpha_1 : l_1, \alpha_2 : l_2} \qquad \dfrac{\dfrac{}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2, \alpha_3 : l_3} \qquad \dfrac{}{a : A, a_1 : l_1, a_3 : l_3 \vdash \bot}}{a : A, a_1 : l_1 \vdash \alpha_2 : l_2}}{a : A \vdash \alpha_2 : l_2}}{a : A \vdash \bot}$$

The CNF here is $A \triangleq \neg l_2 \wedge (l_1 \vee l_2) \wedge (l_1 \Rightarrow l_2 \vee l_3) \wedge \neg(l_1 \wedge l_3)$

A clause, say the third clause $\neg l_1, l_2, l_3$, is seen as a proof of $a : A, a_1 : l_1 \vdash \alpha_2 : l_2, \alpha_3 : l_3$. This proof in "intuitive" syntax is $c_3 \triangleq \texttt{case}\ \pi_3(a)\, a_1\ \texttt{of}\ [b_2 \Rightarrow \texttt{throw}_{\alpha_2}\, b_2 \,|\, b_3 \Rightarrow \texttt{throw}_{\alpha_3}\, b_3]$

The whole proof (in "intuitive", but slightly approximative, syntax) is a proof of $\neg A$:

$$\lambda a.\texttt{let}\ a_2 = \texttt{callcc}_{\alpha_2}.\texttt{let}\ a_1 = \texttt{callcc}_{\alpha_1}.c_2\ \texttt{in}\ \texttt{let}\ a_3 = \texttt{callcc}_{\alpha_3}.c_3\ \texttt{in}\ c_4\ \texttt{in}\ c_1$$

By reducing the cuts, one would get a normal proof of $\neg A$

Further developments

Markov's principle ($\neg\neg\exists x\, A(x) \Rightarrow \exists x\, A(x)$ for $A(x)$ decidable): a weakly classical principle that corresponds to exceptions

Axiom of dependent choice ($\forall x \exists y\, A(x,y) \Rightarrow \exists f\, \forall n\, A(f(n), f(n+1))$): computable e.g. using lazy evaluation of streams

A few issues under progress:

- Combining Martin-Löf's type theory and classical logic (with application to Coq)

- Understanding the computational content of Cohen's forcing method (Krivine, Miquel)

- Understanding the role of side-effects in logic, if ever (memory assignment, ...)

- Computing with the full axiom of choice

...