Programming with an ultrafilter

Christophe Raffalli

LAMA, UMR 5127, Université Savoie Mont Blanc

PLRR 2016, Marseille

What is realizability

- A way to prove safety of type systems and programming languages.
- Building new models (independance results).
- Remark: both approaches use a different meaning of *realizing*.
- In the first case, we do not construct a model.

What is realizability

- A way to prove safety of type systems and programming languages.
- Building new models (independance results).
- Remark: both approaches use a different meaning of *realizing*.
- In the first case, we do not construct a model.
- Discovering algorithm (and getting correct implementation).

Ultrafilter axiom

Programming with the axiom

There is an ultrafilter on $\mathscr{P}(\mathbb{N})$.

- This axiom is *weak* because not any filter may be extended in an ultrafilter.
- We will do it in higher-order logic,
- using inductive and coinductive types and
- call-by-value.
- This way, we will use the program in OCaml.
- Thanks to a Decap syntax extension providing a CPS.
- As application we use Ramsey theorem.

The logic

- The individuals: naturals and infinite sets of naturals.
- The sorts ω for naturals, ζ for infinite sets, o for propositions and $\sigma_1 \rightarrow \sigma_2$.
- Expressions: simply typed terms with
- $A \rightarrow B : o$, $A \wedge B : o$ and $A \vee B : o$ if $A, B : o$,
- $\forall^\sigma x\ A : o$, $\exists^\sigma x\ A : o$ if $x : \sigma \vdash A : o$,
- $\mathbb{N} : \omega \rightarrow o, \mathbb{B} : \omega \rightarrow o$,
- $n < m : \omega$ if $n, m : \omega$, and many other function symbols using ω and ζ.
- $A \cup_{u=v} B$ in sort o if $u, v : \omega$ and $A, B : o$ (NEW).
- Rules HOL rules except for the last connective.

Semantics

- $[\![\omega]\!] = \mathbb{N}$, $[\![\zeta]\!] = \mathscr{P}(\mathbb{N})$, $[\![o]\!] = \mathscr{P}(\mathscr{V})$, $\sigma_1 \to \sigma_2 = [\![\sigma_2]\!]^{[\![\sigma_1]\!]}$.
- $[\![A]\!]$ is therefore a set of values.
- $[\![A]\!]^{\perp}$ is a set of stacks, $[\![A]\!]^{\perp} = \{\pi \mid \forall\, v \in [\![A]\!], v * \pi \in \perp\!\!\!\perp\}$.
- $[\![A]\!]^{\perp\perp}$ is a set of terms, $[\![A]\!]^{\perp\perp} = \{t \mid \forall\, \pi \in [\![A]\!]^{\perp}, t * \pi \in \perp\!\!\!\perp\}$.
- $[\![A \to B]\!] = \{\lambda x\, t \mid \forall\, v \in [\![A]\!], t[x := v] \in [\![B]\!]^{\perp\perp}\}$.
- $[\![\forall\, x^{\sigma} A]\!] = \bigcap_{\varphi \in [\![\sigma]\!]} [\![A[x := \varphi]]\!]$.
- $[\![\exists\, x^{\sigma} A]\!] = \bigcup_{\varphi \in [\![\sigma]\!]} [\![A[x := \varphi]]\!]$.
- $[\![\mathbb{N}(n)]\!] = \{n\}$.
- ...
- $[\![A \cup_{u=v} B]\!] = [\![A]\!]$ if $[\![u]\!] = [\![v]\!]$, $[\![B]\!]$ otherwise.

The type of streams

- $h : \zeta \to \omega$, $q : \zeta \to \zeta$
- $[\![h]\!]$: the least element, $[\![q]\!]$ the other elements.
- Validates some equations: $h(s) < h(q(s)) = 1$.
- $\mathbb{S}_0(s) := \exists X\ X$.
- $\mathbb{S}_{n+1}(s) = \top \to \mathbb{N}(h(s)) \wedge \mathbb{S}(q(s))$.
- $\mathbb{S}(s) = \forall^{\omega} n\ \mathbb{S}_n(s)$.
- $\Omega : (Y \lambda r \lambda n \lambda u\, (n\, ,\, r\, (n+1)))\, 0 \simeq_{\beta} \lambda u\, (0\, ,\, \lambda u\, (1,\, \lambda u\, (2\, ,\, \dots))) : \mathbb{S}(\omega)$
- This definition can be generalized to $\mathbb{S}(X, s)$.

An inductive false, and a predicate for the ultrafilter

- $\bot_0(s) := \forall X\ X.$
- $\bot_{n+1}(s) = \top \to \mathbb{N}(h(s)) \wedge \bot(q(s)).$
- $\bot(s) = \exists^{\omega} n\ \bot_n(s).$

Ultra filter predicate

$$\mathbb{U}(s) = \mathbb{S}(s) \cup_{s \in \mathscr{U}} \bot(s)$$

$s \in \mathscr{U} = 1$ iff $[\![s]\!]$ is in the ultrafilter $\mathscr{U}$ chosen in the initial model.

Warning: $\bot(x) \leftrightarrow \bot$ but $\mathbb{U}(s) \leftrightarrow (\mathbb{S}(s) \cup_{s \in \mathscr{U}} \bot)$!

Ultrafilter axiom.

- Free at the equation level: $s_1 \in \mathscr{U}$ and $s_2 \in \mathscr{U}$ implies $s_1 \cap s_2 \in \mathscr{U}$.
- The equational level can not be used for programming !
- The definition of $\mathbb{U}(s)$ means s infinite.
- $\Omega : \mathbb{U}(\omega)$ is provable from $\Omega : \mathbb{S}(\omega)$ and $\omega \in \mathscr{U}$.
- $J : \forall^{\zeta} s_1, s_2 \mathbb{U}(s_1) \to \mathbb{U}(s_2) \to \mathbb{U}(s_1 \cap s_2)$.
- $K : \forall^{\zeta} s\ \forall^{\omega \to \omega} c\ \mathbb{B}^{\mathbb{N}}(c) \to \mathbb{U}(s) \to \mathbb{U}\big(s \upharpoonright_{c=0}\big) \vee \mathbb{U}\big(s \upharpoonright_{c=1}\big)$.
- Comprehension $C : \forall X\ \big(\forall n\ \mathbb{N}(n) \to \exists p\ \big(\mathbb{N}(p) \upharpoonright_{p>n} X(p)\big) \leftrightarrow \exists s\ \mathbb{S}(X, s)\big)$
- $\mathbb{B}^{\mathbb{N}}(c) := \forall^{\omega} n\ \mathbb{N}(n) \to \mathbb{B}(c(n))$.

The term for intersection

$$
\begin{aligned}
J \coloneqq \lambda s_1 \lambda s_2\, & \mathrm{let}\,(n_1, s_1') = s_1()\,\mathrm{and}\,(n_2, s_2') = s_2()\,\mathrm{in} \\
& \quad \mathrm{if}\, n_1 = n_2\, \mathrm{then}\, \lambda u\,(n_1, J s_1' s_2') \\
& \quad \mathrm{else\,if}\, n_1 < n_2\, \mathrm{then}\, J s_1' s_2 \\
& \quad \mathrm{else}\, J s_1 s_2' \\
& : \mathbb{U}(s_1) \to \mathbb{U}(s_2) \to \mathbb{U}(s_1 \cap s_2)
\end{aligned}
$$

- If $s_1, s_2 \in \mathscr{U}$, $J : \mathbb{S}(s_1) \to \mathbb{S}(s_2) \to \mathbb{S}_n(s_1 \cap s_2)$ by induction on $(n, f(s_1, s_2))$ where $f(s_1, s_2)$ is the number of ignored elements.
- If $s_1, s_2 \notin \mathscr{U}$, $J : \bot_n(s_1) \to \bot_m(s_2) \to \bot(s_1 \cap s_2)$ by induction on $n + m$.
- If $s_1 \in \mathscr{U}$ and $s_2 \notin \mathscr{U}$, $J : \mathbb{S}(s_1) \to \bot_n(s_2) \to \bot(s_1 \cap s_2)$ by induction on $(n, h(s_2) - h(s_1))$.
- fourth case is symmetric

The term for separation

$$K := \lambda c\,\lambda s\,\mathrm{CC}\,\lambda k\,(\mathrm{Inl}\,(\mathrm{CC}\,\lambda k_1\,(k\,(\mathrm{Inr}\,(\mathrm{CC}\,\lambda k_2\,L\,c\,s\,k_1\,k_2)))))$$
$$: \mathbb{B}^{\mathbb{N}}(c) \to \mathbb{U}(s) \to \mathbb{U}\big(s \restriction_{c=0}\big) \vee \mathbb{U}\big(s \restriction_{c=1}\big)$$

$$L := \lambda c\,\lambda s\,\lambda k_1\,\lambda k_2\,\mathrm{let}\,(n\,,\,s') = s\,()\,\mathrm{in}$$
$$\mathrm{if}\,c(n) = 0\,\mathrm{then}\,k_1\,\lambda u\,(n\,,\,\mathrm{CC}\,\lambda k_1'\,L\,c\,s\,k_1'\,k_2)$$
$$\mathrm{else}\,k_2\,\lambda u\,(n\,,\,\mathrm{CC}\,\lambda k_2'\,L\,c\,s\,k_1\,k_2')$$

- If $s \restriction_{c=0} \in \mathscr{U}$, $J : \mathbb{B}^{\mathbb{N}}(c) \to \mathbb{S}(s) \to \neg\,\mathbb{S}_n\big(s \restriction_{c=0}\big) \to \neg\bot\big(s \restriction_{c=1}\big) \to \bot$ by induction on $(n\,,\,f(s))$ where $f(s)$ is the number of elements x such that $c(x) = 1$ at the beginning of s.
- Second case $s \restriction_{c=1} \in \mathscr{U}$ is symmetric.
- It is here we need $\bot(s)$.

Intuitive meaning

K is a parallel composition

$K : \mathbb{B}^{\mathbb{N}}(c) \rightarrow \mathbb{U}(s) \rightarrow \mathbb{U}\left(s \upharpoonright_{c=0}\right) \vee \mathbb{U}\left(s \upharpoonright_{c=1}\right)$

evaluates both alternatives in parallel.

J is somehow a scheduler (together with K)

$J : \mathbb{U}(s_1) \rightarrow \mathbb{U}(s_2) \rightarrow \mathbb{U}(s_1 \cap s_2)$

Trigger context switching by consuming the input streams.

Translation to OCaml : J

```
type 'a stream = unit ⇒ ('a * 'a stream)

let classical inter (s1:(int * 'b) stream) (s2:(int * 'a) stream) () =
  let ((n1, _), s1) = s1 () in let ((n2,x2), s2) = s2 () in
  aux2 n1 s1 n2 x2 s2

and auxL s1 n2 x2 s2 = let ((n1, _ ), s1) = s1 () in aux2 n1 s1 n2 x2 s2

and auxR n1 s1    s2 = let ((n2, x2), s2) = s2 () in aux2 n1 s1 n2 x2 s2

and aux2 n1 s1 n2 x2 s2 =
  if val (n1 < n2) then auxL s1 n2 x2 s2 else
  if val (n1 > n2) then auxR n1 s1 s2 else ((n2,x2), inter s1 s2)
```

Translation to OCaml : K

```
type ('a, 'b) sum = Inl of 'a | Inr of 'b

let classical split (c : int ⇒ ('a,'b) sum) (s : int stream) =
  mu k → Inl (lazy (mu k1 →
         (Inr (lazy (mu k2 → split_aux c s k1 k2)))  k))

and split_aux (c : int ⇒ ('a,'b) sum) s k1 k2 =
  let (n, s) = s () in
  match c n with
  | Inl x → ((n,x), lazy (mu k1 → split_aux c s k1 k2))  k1
  | Inr x → ((n,x), lazy (mu k2 → split_aux c s k1 k2))  k2
```

Ramsey Ultrafilter

```
open Ultrafilter

let classical omega n () = (n, omega (val(n+1)))

let classical color1 (c : int => int => (unit, unit) sum) n =
  Ultrafilter.split (c n) (omega (val(n+1)))

let classical aux c = Ultrafilter.split (color1 c) (omega (val 0))

let classical extract (s : (int * (int * unit) stream) stream) () =
  let ((n, s1), s2) = s () in (n, extract (Ultrafilter.inter s1 s2))

let classical ramsey c = match aux c with
  | Inl s → Inl (extract s) | Inr s → Inr (extract s)
```

Ramsey Ultrafilter, testing !

```
let classical extract_list s n =
  if val(n = 0) then [] else let (p,s') = s () in
                           p :: extract_list s' (val (n - 1))

let classical finite_ramsey c n = match ramsey c with
  | Inl s → Inl (extract_list s n) | Inr s → Inr (extract_list s n)

let rec pr ch = function [] → () | [x] → Printf.fprintf ch "%d" x
                       | x::l → Printf.fprintf ch "%d,%a" x pr l
let print = function Inl l → Printf.printf "Inl(%a)\n%!" pr l
                   | Inr l → Printf.printf "Inr(%a)\n%!" pr l

let test color n = print (?finite_ramsey color n?)
```

Ramsey Ultrafilter, testing !

```
open Ultrafilter
open RamseyU

let classical color n m =
  val (if (n + m) mod 2 = 0
       then Inl () else Inr ())

let _ = test color 5




>>  Inl(0,2,4,6,8)
>>
```

Conclusion

- Direct proof of Ramsey: faster program / larger integers
- Search among all possilities: slower program / least integers
- Can we establich a link ?
- The program that look for all possibilities corresponds to what proof ?
- Probably the selective ultrafilter axiom.