Preliminary Report on the Yalla Library

Yet Another deep embedding of Linear Logic in Coq

Coq Workshop 2018

Olivier LAURENT

CNRS – Lyon – France

Olivier.Laurent@ens-lyon.fr

July 8, 2018

Linear Logic Proofs

```
Inductive ll P : list formula → Type :=
| ax_r : ∀ X, ll P (covar X :: var X :: nil)
| ex_r : ∀ l1 l2, ll P l1 → PCperm_Type (pperm P) l1 l2 → ll P l2
| mix0_r {f : pmix0 P = true} : ll P nil
| mix2_r {f : pmix2 P = true} : ∀ l1 l2, ll P l1 → ll P l2 → ll P (l2 ++ l1)
| one_r : ll P (one :: nil)
| bot_r : ∀ l, ll P l → ll P (bot :: l)
| tens_r : ∀ A B l1 l2, ll P (A :: l1) → ll P (B :: l2) → ll P (tens A B :: l2 ++ l1)
| parr_r : ∀ A B l, ll P (A :: B :: l) → ll P (parr A B :: l)
| top_r : ∀ l, ll P (top :: l)
| plus_r1 : ∀ A B l, ll P (A :: l) → ll P (aplus A B :: l)
| plus_r2 : ∀ A B l, ll P (A :: l) → ll P (aplus B A :: l)
| with_r : ∀ A B l, ll P (A :: l) → ll P (B :: l) → ll P (awith A B :: l)
| oc_r : ∀ A l, ll P (A :: map wn l) → ll P (oc A :: map wn l)
| de_r : ∀ A l, ll P (A :: l) → ll P (wn A :: l)
| wk_r : ∀ A l, ll P l → ll P (wn A :: l)
| co_r : ∀ A lw l, ll P (wn A :: map wn lw ++ wn A :: l) → ll P (wn A :: map wn lw ++ l)
| cut_r {f : pcut P = true} : ∀ A l1 l2, ll P (dual A :: l1) → ll P (A :: l2) → ll P (l2 ++ l1)
| gax_r : ∀ a, ll P (projT2 (pgax P) a).
```

Linear Logic Proofs

Parameters

```
Inductive ll P : list formula → Type :=
| ax_r : ∀ X, ll P (covar X :: var X :: nil)
| ex_r : ∀ l1 l2, ll P l1 → PCperm_Type (pperm P) l1 l2 → ll P l2
| mix0_r {f : pmix0 P = true} : ll P nil
| mix2_r {f : pmix2 P = true} : ∀ l1 l2, ll P l1 → ll P l2 → ll P (l2 ++ l1)
| one_r : ll P (one :: nil)
| bot_r : ∀ l, ll P l → ll P (bot :: l)
| tens_r : ∀ A B l1 l2, ll P (A :: l1) → ll P (B :: l2) → ll P (tens A B :: l2 ++ l1)
| parr_r : ∀ A B l, ll P (A :: B :: l) → ll P (parr A B :: l)
| top_r : ∀ l, ll P (top :: l)
| plus_r1 : ∀ A B l, ll P (A :: l) → ll P (aplus A B :: l)
| plus_r2 : ∀ A B l, ll P (A :: l) → ll P (aplus B A :: l)
| with_r : ∀ A B l, ll P (A :: l) → ll P (B :: l) → ll P (awith A B :: l)
| oc_r : ∀ A l, ll P (A :: map wn l) → ll P (oc A :: map wn l)
| de_r : ∀ A l, ll P (A :: l) → ll P (wn A :: l)
| wk_r : ∀ A l, ll P l → ll P (wn A :: l)
| co_r : ∀ A lw l, ll P (wn A :: map wn lw ++ wn A :: l) → ll P (wn A :: map wn lw ++ l)
| cut_r {f : pcut P = true} : ∀ A l1 l2, ll P (dual A :: l1) → ll P (A :: l2) → ll P (l2 ++ l1)
| gax_r : ∀ a, ll P (projT2 (pgax P) a).
```

Linear Logic Proofs

Non Commutativity

```
Inductive ll P : list formula → Type :=
| ax_r : ∀ X, ll P (covar X :: var X :: nil)
| ex_r : ∀ l1 l2, ll P l1 → PCperm_Type (pperm P) l1 l2 → ll P l2
| mix0_r {f : pmix0 P = true} : ll P nil
| mix2_r {f : pmix2 P = true} : ∀ l1 l2, ll P l1 → ll P l2 → ll P (l2 ++ l1)
| one_r : ll P (one :: nil)
| bot_r : ∀ l, ll P l → ll P (bot :: l)
| tens_r : ∀ A B l1 l2, ll P (A :: l1) → ll P (B :: l2) → ll P (tens A B :: l2 ++ l1)
| parr_r : ∀ A B l, ll P (A :: B :: l) → ll P (parr A B :: l)
| top_r : ∀ l, ll P (top :: l)
| plus_r1 : ∀ A B l, ll P (A :: l) → ll P (aplus A B :: l)
| plus_r2 : ∀ A B l, ll P (A :: l) → ll P (aplus B A :: l)
| with_r : ∀ A B l, ll P (A :: l) → ll P (B :: l) → ll P (awith A B :: l)
| oc_r : ∀ A l, ll P (A :: map wn l) → ll P (oc A :: map wn l)
| de_r : ∀ A l, ll P (A :: l) → ll P (wn A :: l)
| wk_r : ∀ A l, ll P l → ll P (wn A :: l)
| co_r : ∀ A lw l, ll P (wn A :: map wn lw ++ wn A :: l) → ll P (wn A :: map wn lw ++ l)
| cut_r {f : pcut P = true} : ∀ A l1 l2, ll P (dual A :: l1) → ll P (A :: l2) → ll P (l2 ++ l1)
| gax_r : ∀ a, ll P (projT2 (pgax P) a).
```

Hiding Parameters

Recommendation

- define your own inductive
- inject it in an instance of **ll**
- import / use results from the library

Various Templates Provided

```
Inductive mell : list formula → Type :=
| ax_r : ∀ X, mell (covar X :: var X :: nil)
| ex_r : ∀ l1 l2, mell l1 → Permutation_Type l1 l2 → mell l2
| mix_r : ∀ l1 l2, mell l1 → mell l2 → mell (l1 ++ l2)
| tens_r : ∀ A B l1 l2, mell (A :: l1) → mell (B :: l2) → mell (tens A B :: l1 ++ l2)
| parr_r : ∀ A B l, mell (A :: B :: l) → mell (parr A B :: l)
| oc_r : ∀ A l, mell (A :: map wn l) → mell (oc A :: map wn l)
| de_r : ∀ A l, mell (A :: l) → mell (wn A :: l)
| wk_r : ∀ A l, mell l → mell (wn A :: l)
| co_r : ∀ A l, mell (wn A :: wn A :: l) → mell (wn A :: l)
```

Linear Logic Proofs

Computational Content

```
Inductive ll P : list formula → Type :=
| ax_r : ∀ X, ll P (covar X :: var X :: nil)
| ex_r : ∀ l1 l2, ll P l1 → PCperm_Type (pperm P) l1 l2 → ll P l2
| mix0_r {f : pmix0 P = true} : ll P nil
| mix2_r {f : pmix2 P = true} : ∀ l1 l2, ll P l1 → ll P l2 → ll P (l2 ++ l1)
| one_r : ll P (one :: nil)
| bot_r : ∀ l, ll P l → ll P (bot :: l)
| tens_r : ∀ A B l1 l2, ll P (A :: l1) → ll P (B :: l2) → ll P (tens A B :: l2 ++ l1)
| parr_r : ∀ A B l, ll P (A :: B :: l) → ll P (parr A B :: l)
| top_r : ∀ l, ll P (top :: l)
| plus_r1 : ∀ A B l, ll P (A :: l) → ll P (aplus A B :: l)
| plus_r2 : ∀ A B l, ll P (A :: l) → ll P (aplus B A :: l)
| with_r : ∀ A B l, ll P (A :: l) → ll P (B :: l) → ll P (awith A B :: l)
| oc_r : ∀ A l, ll P (A :: map wn l) → ll P (oc A :: map wn l)
| de_r : ∀ A l, ll P (A :: l) → ll P (wn A :: l)
| wk_r : ∀ A l, ll P l → ll P (wn A :: l)
| co_r : ∀ A lw l, ll P (wn A :: map wn lw ++ wn A :: l) → ll P (wn A :: map wn lw ++ l)
| cut_r {f : pcut P = true} : ∀ A l1 l2, ll P (dual A :: l1) → ll P (A :: l2) → ll P (l2 ++ l1)
| gax_r : ∀ a, ll P (projT2 (pgax P) a).
```

Curry-Howard

Sequents as Multisets

(already in Intuitionistic Logic)

which Church Boolean is this?

$$\dfrac{\dfrac{\dfrac{}{[\![A, A]\!] \vdash A}}{[\![A]\!] \vdash A \to A}}{[\![\ ]\!] \vdash A \to A \to A}$$

Sequents as Lists

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\color{green}{A} \vdash \color{green}{A}}}{\color{green}{A}, \color{magenta}{A} \vdash \color{green}{A}}}{\color{green}{A} \vdash \color{magenta}{A} \to \color{green}{A}}}{\vdash \color{green}{A} \to \color{magenta}{A} \to \color{green}{A}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\color{green}{A} \vdash \color{green}{A}}}{\color{green}{A}, \color{magenta}{A} \vdash \color{green}{A}}\ \text{ex (12)}}{\color{magenta}{A}, \color{green}{A} \vdash \color{green}{A}}\ \to}{\color{magenta}{A} \vdash \color{green}{A} \to \color{green}{A}}\ \to}{\vdash \color{magenta}{A} \to \color{green}{A} \to \color{green}{A}}$$

Proofs rather than Provability

proof size is defined : $\quad$ **ll** $P\ l \to$ **nat**

Finite Multisets

Desired Properties

Finite multisets up to `Coq` equality *(without proof irrelevance)*

Axiomatic Definition

```
Class FinMultiset M A := {
  empty : M;
  add : A → M → M;
  elts : M → list A;
  elts_empty : elts empty = @nil A;
  elts_add : ∀ a m, Permutation (elts (add a m)) (a :: elts m);
  retract : ∀ m, fold_right add empty (elts m) = m;
  perm_eq : ∀ l1 l2, Permutation l1 l2 →
                         fold_right add empty l1 = fold_right add empty l2 }.
```

```
Definition list2fm l := fold_right add empty l.
Lemma elts_perm : ∀ l, Permutation (elts (list2fm l)) l.
Definition sum m1 m2 := list2fm (elts m1 ++ elts m2).

...
```

Instances

For any B : **UsualOrderedTypeFull** (in particular $B \to$ **nat** injective)
we can build **FinMultiset** (**SortedList** B) B

Standard Library

stdlib 2?

- Natural results missing on lists, permutations, etc

  `Lemma` in_elt $\{A\}$: $\forall$ $(a{:}A)$ $l1$ $l2$, In a $(l1$ ++ a :: $l2)$.

  `Lemma` Forall_app_inv $\{A\}$: $\forall$ P $(l1\ l2$: **list** $A)$,
  Forall P $(l1$ ++ $l2)$ $\leftrightarrow$ **Forall** P $l1$ $\wedge$ **Forall** P $l2$.

- From `Prop` to `Type`
  - adapt existing results
  - back to `Prop` by **inhabited**
  - correct `setoid_rewrite` `[Issue #7675]`
  - **and** and **prod** associate differently `[Issue #7676]`

- Cyclic permutations
- Finite multisets

- Notions of orders (choose one in stdlib? others outside?)

What's Next?

Ongoing for Release 2.0

- cut-elimination proof for full ILL
- deduce cut-elimination for LL from ILL

Planned

- Quantifiers in linear logic
- More automation for permutation solving
- Parametric exponential rules

Try It

`https://perso.ens-lyon.fr/olivier.laurent/yalla/`
`https://github.com/olaure01/yalla`
`https://github.com/olaure01/yalla/tree/working/`