Yalla

Yet Another deep embedding of Linear Logic in Coq

Machine Proofs of Linear Logic

Olivier LAURENT

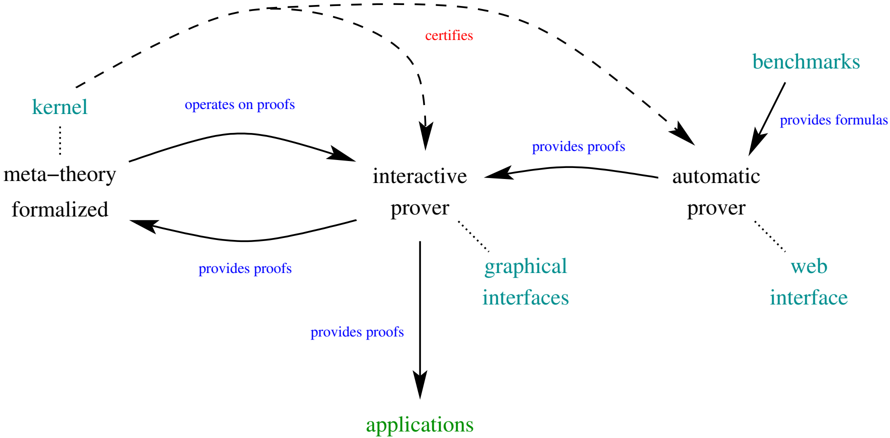
CNRS – Lyon – France

Olivier.Laurent@ens-lyon.fr

December 18, 2018

Computer-Assisted Linear Logic

An Ecosystem



Yalla

Representation of Linear Logic Proofs

The Inductive Type

```
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
| ex_wn_r : ∀ l1 lw lw' l2, ll P (l1 ++ map wn lw ++ l2) →
                    Permutation_Type lw lw' → ll P (l1 ++ map wn lw' ++ 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 l, ll P (wn A :: wn A :: l) → ll P (wn A :: 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).
```

Representation of Linear Logic Proofs

Parameters

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

Representation of 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
| ex_wn_r : ∀ l1 lw lw' l2, ll P (l1 ++ map wn lw ++ l2) →
                      Permutation_Type lw lw' → ll P (l1 ++ map wn lw' ++ 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 l, ll P (wn A :: wn A :: l) → ll P (wn A :: 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

Recommendations

- 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).

Fixpoint mell2ll : formula → formulas.formula

Definition pfrag_mell := mk_pfrag false NoAxioms false  true   true.
                                  cut   axioms   mix0  mix2  perm

Lemma mell2mellfrag : ∀ l, mell l ↔ ll pfrag_mell (map mell2ll l).
```

Representation of 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
| ex_wn_r : ∀ l1 lw lw' l2, ll P (l1 ++ map wn lw ++ l2) →
                    Permutation_Type lw lw' → ll P (l1 ++ map wn lw' ++ 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 l, ll P (wn A :: wn A :: l) → ll P (wn A :: 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

which Church Boolean is this?

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

Curry-Howard

Sequents as Multisets

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}}{A, A \vdash A}}{A \vdash A \to A}}{\vdash A \to A \to A}$$

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

Curry-Howard

Sequents as Multisets

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}}{A, A \vdash A}}{A \vdash A \to A}}{\vdash A \to A \to A}$$

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

Proofs (`Type`) rather than Provability (`Prop`)

proof size as a defined function: $\mathbf{ll}\ P\ l \to \mathbf{nat}$

Induced Coq Contributions

Standard Library

- Missing results on lists, permutations, etc

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

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

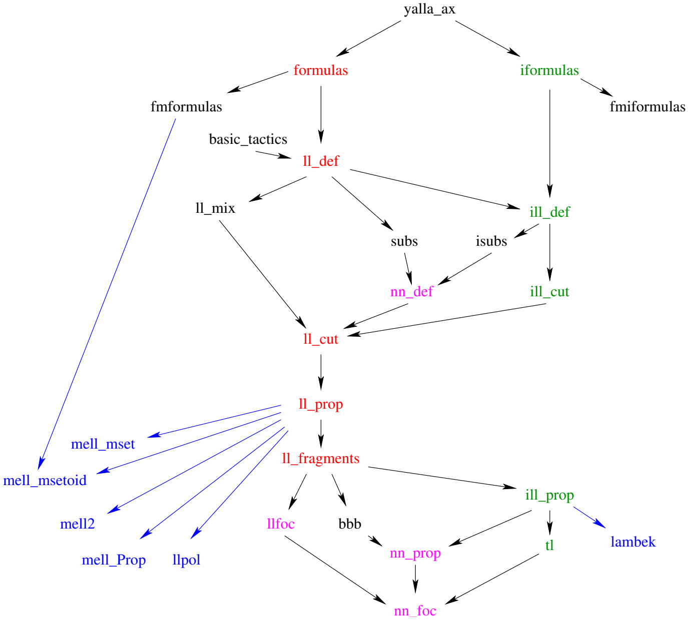
  $\vdots$

- From `Prop` to `Type`
  - bug in `setoid_rewrite`
  - **and** and **prod** associate differently

Add-Ons

- Cyclic permutations
- Parametric permutations PCperm PEperm
- Finite multisets up to Coq equality

Current State



Main Content

Results

- substitutions and freshness
- around mix rules
- translations $\mathsf{LL} \leftrightarrow \mathsf{ILL} \leftrightarrow \mathsf{TL}$
- conservativity $\mathsf{LL} \leftrightarrow \mathsf{ILL} \leftrightarrow \mathsf{TL}$
- cut elimination
- sub-formula property
- deduction theorem
- reversibility and focusing

Related Systems (some)

- Lambek calculus
- MELL in `Prop`
- MELL with multisets

What's Next?

Before Release 2.0 (ongoing)

- Cut-elimination proof for ILL
- Non-commutative cut-elimination for ILL and LL
- Some cleaning
- Move to Coq 8.9

Planned (not necessarily in 2.0)

- Quantifiers in linear logic
- More automation for permutation solving
- Parametric exponential rules (subexponentials, light, etc)
- Cut-elimination as proof rewriting
- Denotational semantics
- Intuitionistic and Classical Logics `(ongoing [C. Lucas])`
- Automating correspondence with user-defined fragments

Try It!!!

`https://perso.ens-lyon.fr/olivier.laurent/yalla/`

`https://github.com/olaure01/yalla/tree/working`

Users, comments and manpower are welcome!

Support guaranteed:

- `olivier.laurent@ens-lyon.fr`
- `https://github.com/olaure01/yalla/issues`