Preliminary Report on the `Yalla` Library

Yet Another deep embedding of Linear Logic in `Coq`

Olivier Laurent

Univ. Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, F-69342, LYON Cedex 07, France

`Olivier.Laurent@ens-lyon.fr`

Abstract

We present some results and comments around the ongoing development of the `Yalla` library which provides a deep embedding of linear logic in `Coq` relying on an explicit exchange rule.

The `Yalla` library (`https://perso.ens-lyon.fr/olivier.laurent/yalla/`) provides a deep embedding of Linear Logic (`LL`) [Gir87] in `Coq` in order to allow users to prove meta-theoretical properties of (variants of) `LL`.

Various developments around deep embeddings of `LL` into theorem provers already exist (for example [PW99, CLR17, XORN17]) but either they do not really consider proving serious meta-theoretical properties (such as cut elimination), or they rely on sequents defined as multisets. This is for example the case of [XORN17] which is probably the closest to what we do here. While allowing to prove properties of the `LL` sequent calculus, the main specificity of `Yalla` is thus to rely on an explicit exchange rule to be ready for computational interpretations of proofs through the Curry-Howard correspondence. Moreover some variants of the system are taken into account such as cyclic linear logic, introduction of the *mix* rules, etc.

The presented work focuses on the propositional setting (clearly quantification should be part of future work but would depend on a choice of representation for binders).

1 Sequents as Lists

Sequents and Curry-Howard. Sequent calculus systems (in particular for `LL`) are often formalized by defining sequents as finite multisets which is not appropriate to give computational content to proofs through the Curry-Howard correspondence. Here is an intuitionistic example:

$$\begin{array}{c} \dfrac{\dfrac{\dfrac{A \in (\!|A,A|\!)}{A,A \vdash A}\,\text{ax}}{A \vdash A \to A}\,\to}{\vdash A \to A \to A}\,\to \\ \lambda x.\lambda y.? \end{array} \quad \text{vs} \quad \begin{array}{c} \dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{ax}}{A,A \vdash A}\,\text{wk}}{A \vdash A \to A}\,\to}{\vdash A \to A \to A}\,\to \\ \lambda x.\lambda y.x \end{array} \quad \text{and} \quad \begin{array}{c} \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{ax}}{A,A \vdash A}\,\text{wk}}{A,A \vdash A}\,\text{ex}\ (12)}{A \vdash A \to A}\,\to}{\vdash A \to A \to A}\,\to \\ \lambda x.\lambda y.y \end{array} \quad \text{with} \quad \begin{array}{c} \dfrac{\Gamma \vdash A}{\Gamma, B \vdash A}\,\text{wk} \\[2ex] \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\to \end{array}$$

On the left-hand side, we use a multiset $(\!|A, A|\!)$ and it is not possible to distinguish the two occurrences of A. On the right-hand side, sequents are lists and an explicit exchange rule specifies which permutation of the formulas is applied, making possible to follow formula occurrences along derivations.

Proofs with Lists. The inductive type we consider for `LL` proofs in `Yalla` is based on:

```
Inductive ll: list formula -> Type :=
| ex_r: forall l1 l2, ll l1 -> Permutation_Type l1 l2 -> ll l2
| tens_r: forall A B l1 l2, ll (A :: l1) -> ll (B :: l2) -> ll (tens A B :: l2 ++ l1)
...
```

Note the use of `Type` as output for `ll` (this allows us in particular to define the size of proofs as a function `ll l -> nat`), and the use of `Permutation_Type` in the exchange rule. `Permutation_Type` is the direct adaptation of `Permutation` from the standard library to the type `list A -> list A -> Type` instead of `list A -> list A -> Prop`. Similarly we have adapted `Equalities` to `CEqualities` and `Orders` to `COrders` by turning `Prop` into `Type` in relations (following `CRelationClasses`). This opens the question of having a more systematic use of `Type` instead of `Prop` in the standard library. The move works reasonably well but some troubles occur with `setoid_rewrite`.

Work supported by Fédération Informatique de Lyon under LLipIdO project: `https://perso.ens-lyon.fr/olivier.laurent/llipido/`

In order to simplify the use of permutations, we have developed some naive automation to solve goals of the shape `Permutation _ _, ..., Permutation _ _ |- Permutation _ _`. A more systematic approach (such as a decision procedure if possible) would be useful.

The important use of manipulations of lists led us to propose some add-ons for the standard library (mainly on lists and permutations). This includes some very natural properties which could be added to the standard library.

Multisets. Even if the `Yalla` library is developed with lists and permutations, we want the user to be able to define his own inductive type for proofs relying on multisets, and to use `Yalla` results anyway. For this we want a notion of finite multisets for which equality is `Coq` equality (without relying on proof irrelevance) and which exactly defines the quotient of lists up to permutation (with functions `list A -> FinMultiset A` and `FinMultiset A -> list A`). The material we can find in the standard library or others (`CoLoR` library[1], P. Letouzey's work[2], or [XORN17]) do not fit this specification. We have axiomatized the corresponding notion:

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

and we provide a construction of such finite multisets from any `UsualOrderedTypeFull` by using ordered lists (note that choosing between orders notions from `Orders` or `Relation_Definitions` or `Relations_1` in the standard library is not completely immediate for the beginner).

2 Linear Logic Specificities

Parametrized Rules. Following the idea of levels in D. Pous' `relation-algebra`[3], our inductive type for proofs is in fact parametrized in such a way that the user could choose: to consider full permutations or cyclic ones only (with a dedicated adaptation of the `Permutation` library for cyclic permutations), to add open hypotheses, to include the *cut* rule or not, to use *mix* rules or not. This makes the inductive type heavier to manipulate but it allows us to factorize a lot of results in the library (with, in the case of permutations in particular, a tension sometimes between trying to make one single but more involved proof, or to simply split into two cases).

It is then strongly suggested to the users to define their own inductive type for the specific proof system they want to study. In this way they do not have to manipulate the parametrized inductive type of `Yalla` in their development, but just once and for all in proving equivalence between their type and an instance of the type of the library. Such interface files are provided in `Yalla` for a few chosen natural instances, and could be used as models for others.

Cut Elimination. The cut-elimination property is proved by a rather standard induction on the size of the cut formula and on the sum of the sizes of the premises (this size is defined as a mix of the number of rules and of the height of the proof by using a sum for binary multiplicative rules and a max for binary additive rules). Various cases are similar, and we still do not have a clear opinion about what is the best approach to factorize proofs here: write `ltac` code, simply copy-paste tactics, rely on `Coq` automation, etc.

We would like to thank Damien Pous for his crucial help in the development of the `Yalla` library.

References

[CLR17] Kaustuv Chaudhuri, Leonardo Lima, and Giselle Reis. Formalized meta-theory of sequent calculi for substructural logics. *Electronic Notes in Theoretical Computer Science*, 332:57–73, June 2017.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[PW99] James Power and Caroline Webster. Working with linear logic in Coq. In *Theorem Proving in Higher Order Logics: Emerging Trends*, September 1999.

[XORN17] Bruno Xavier, Carlos Olarte, Giselle Reis, and Vivek Nigam. Mechanizing linear logic in Coq. In S. Alves and R. Wassermann, editors, *Logical and Semantic Frameworks with Applications*, pages 60–77, 2017.

[1] http://color.inria.fr/doc/CoLoR.Util.Multiset.MultisetList.html
[2] https://github.com/coq-contribs/fsets/blob/master/MultiSets.v
[3] http://perso.ens-lyon.fr/damien.pous/ra/