Investigating the Use of Lemmas

Vivek Nigam

INRIA & LIX/École Polytechnique, Palaiseau, France
`nigam at lix.inria.fr`

Abstract. When using an interactive theorem prover, it is necessary to inform precisely where lemmas are used. Some recent advances in proof theory, namely on focused proofs for intuitionistic logic [9], allow us to annotate lemmas in such a way that these lemmas must be (re)used. We showed, elsewhere [12], that if lemmas are restricted to literals and if programs are restricted to horn clauses, it is possible to manipulate the *polarity* of atoms in such a way that these lemmas are not reproved. We extend some of the results obtained before by allowing, both, lemmas and program clauses to be hereditary Harrop formulas [11]. We show that by manipulating the polarity of atoms, simpler proofs can be obtained when reproving a lemma. Later, we discuss how a set of lemmas can be seen as a proof object, investigate how lemmas can be extracted from a goal directed proof, and discuss some possible advantages of using such representation for proofs based on lemmas.

1 Introduction

When proving a theorem in an interactive theorem prover, such as NQTHM or Coq [3, 5, 2], a user often makes use of a set of intermediary results called lemmas. But stating the lemmas themselves is not enough for the theorem prover to complete the proof, and the user must also provide the exact position where the lemmas must be used.

Some recent advances in proof theory, namely on focused proofs for intuitionistic logic [9], allow us to annotate lemmas in such a way that these lemmas must be used and reused. We showed, elsewhere [12], that if lemmas are restricted to literals and if programs are restricted to horn clauses, it is possible to manipulate the *polarity* of atoms in such a way that these lemmas are not reproved. As a consequence, it is not necessary for a user to inform exactly where such annotated lemmas are used, but only inform which lemmas are necessary in the proof.

In this paper, we generalize some of the results obtained before, by allowing, both, lemmas and program clauses to be hereditary Harrop formulas [11]. We consider a *table* as a partially ordered set of lemmas, where the partial order represents the proof dependency between the lemmas, that is, if $A \prec B$ then the lemma A is used in the proof of B. We introduce a lemma from a table to the context of a sequent in a sequent calculus derivation [7] by using a cut rule. This allows the lemma to be available for use at any moment in the proof. And by manipulating the polarity of atoms, we can enforce that a lemma is used only by performing a forward chaining on it. This guarantees the existence of simpler proofs whenever a lemma needs to be used.

Later, our attention is turned towards the possibilities of using tables and multicut derivations as proof objects, that is a data structure that represents a proof. We present another representation of proofs, where lemmas not only can be introduced into the context of a sequent in the beginning of the proof, but can also be introduced in the middle of the proof. We discuss how focusing plays an important role on identifying where to introduce lemmas, and we propose a way to extract a table from a goal directed proof.

In the next section, we introduce the main concepts of a focused system, and later in Section 3 we introduce the focused system LJF^t. In Section 4, we describe how a table can be translated to a multicut derivation, and in Section 5, we investigate the complexity of a proof where a lemma is used. In Section 6, we investigate how tables and multicut derivations can be used as proof objects. Finally, in Section 7, we make some final comments and discuss some future work.

2 Focusing

A focused proof system provides a way to organize cut-free proofs by classifying the connectives $\wedge$, $\exists$, *true* and $\perp$ as *synchronous* (their right introduction is not necessarily invertible) and the connectives $\supset$, and $\forall$ as *asynchronous* (their right introduction rules are invertible). This dichotomy must also be extended to atomic formulas: some atoms are considered asynchronous and the rest are considered synchronous. Since the terms "asynchronous" and "synchronous" do not apply well to atomic formulas, we shall instead use the slightly more general notions of *polarity* for a formula. In particular, a formula is positive if its main connective is synchronous or it is a *positive atom* and is negative if its main connective is asynchronous or it is a *negative atom.*

A *focused* proof is composed of two alternating phases: the *asynchronous phase* applies asynchronous rules until exhaustion, that is, until no more invertible rules can be applied, and the *synchronous phase* or *focus phase*, which is composed only of non-invertible rules applied exhaustively to a given formula and any of its positive subformulas.

This special structure of focused proofs provides us with a clear distinction of "parts" in focused proofs which have a "don't know" non-determinism (synchronous phases), from "parts" in focused proofs that have a "don't care" non-determinism (asynchronous phases). This identification is not new and was already used, for example, to construct automated theorem provers [4]. We investigate in Section 6, how to capitalize on this structure and construct proof objects based only on the synchronous phases of the proof, where backtracking can potentially occur.

The notion of *focused proof* was originally given by Andreoli for linear logic [1], but several focused intuitionistic systems have been proposed [6, 8, 9]. We use

$$\frac{[N,\Gamma] \xrightarrow{N} [R]}{[N,\Gamma] \longrightarrow [R]}\ D_l \qquad \frac{[\Gamma]-_{P}\!\!\rightarrow}{[\Gamma] \longrightarrow [P]}\ D_r \qquad \frac{[\Gamma], P \longrightarrow [R]}{[\Gamma] \xrightarrow{P} [R]}\ R_l \quad \frac{[\Gamma] \longrightarrow N}{[\Gamma]-_{N}\!\!\rightarrow}\ R_r$$

$$\frac{[\Gamma, N_a], \Theta \longrightarrow \mathcal{R}}{[\Gamma], \Theta, N_a \longrightarrow \mathcal{R}}\ []_l \quad \frac{[\Gamma], \Theta \longrightarrow [P_a]}{[\Gamma], \Theta \longrightarrow P_a}\ []_r$$

$$\frac{}{[\Gamma] \xrightarrow{A_n} [A_n]}\ I_l \quad \frac{}{[\Gamma, A_p]-_{A_p}\!\!\rightarrow}\ I_r$$

$$\frac{}{[\Gamma], \Theta, \bot \longrightarrow \mathcal{R}}\ \mathit{false}_l \quad \frac{[\Gamma], \Theta \longrightarrow \mathcal{R}}{[\Gamma], \Theta, \mathit{true} \longrightarrow \mathcal{R}}\ \mathit{true}_l \quad \frac{}{[\Gamma]-_{\mathit{true}}\!\!\rightarrow}\ \mathit{true}_r$$

$$\frac{[\Gamma], \Theta, A, B \longrightarrow \mathcal{R}}{[\Gamma], \Theta, A \wedge B \longrightarrow \mathcal{R}}\ \wedge_l \quad \frac{[\Gamma]-_{A}\!\!\rightarrow \quad [\Gamma]-_{B}\!\!\rightarrow}{[\Gamma]-_{A \wedge B}\!\!\rightarrow}\ \wedge_r$$

$$\frac{[\Gamma]-_{A}\!\!\rightarrow \quad [\Gamma] \xrightarrow{B} [R]}{[\Gamma] \xrightarrow{A \supset B} [R]}\ \supset_l \quad \frac{[\Gamma], \Theta, A \longrightarrow B}{[\Gamma], \Theta \longrightarrow A \supset B}\ \supset_r$$

$$\frac{[\Gamma], \Theta, A \longrightarrow \mathcal{R}}{[\Gamma], \Theta, \exists y A \longrightarrow \mathcal{R}}\ \exists_l \quad \frac{[\Gamma]-_{A[t/x]}\!\!\rightarrow}{[\Gamma]-_{\exists x A}\!\!\rightarrow}\ \exists_r \quad \frac{[\Gamma] \xrightarrow{A[t/x]} [R]}{[\Gamma] \xrightarrow{\forall x A} [R]}\ \forall_l \quad \frac{[\Gamma], \Theta \longrightarrow A}{[\Gamma], \Theta \longrightarrow \forall y A}\ \forall_r$$

Fig. 1. The LJF system [9] originally has two conjunctions, $\wedge^+, \wedge^-$. In this paper, we only use $\wedge^+$ so we have dropped the rules for $\wedge^-$ and simply write the remaining conjunction as $\wedge$. Here A_n denotes a negative atom, A_p a positive atom, and P a positive formula, N a negative formula, N_a a negative formula or an atom, and P_a a positive formula or an atom. All other formulas are arbitrary and y is not free in Γ, Θ or R.

a recent proposal by Liang *et al.*, called LJF [9]. This system differs from the other proposals by allowing atoms to have arbitrary polarity. This flexibility is fundamental to the results in this paper. It has been observed [1, 9] that changing the polarity of atoms doesn't affect the provability of a judgment, but can affect deeply the shape of its proof. For instance in Horn theory, if all atoms are assigned with positive polarity (resp. negative polarity) only a forward chaining proof (resp. back-chaining or goal directed proof [11]) can be obtained.

In the next section we introduce the LJF system, and its extension LJF^t which includes a polarized multicut rule.

3 LJF^t

The LJF focused proof system for intuitionistic logic [9] is depicted in Figure 1. This system has four types of sequents.

1. The sequent $[\Gamma]-_{A}\!\!\rightarrow$ is a *right-focusing* sequent (the focus is A).
2. The sequent $[\Gamma] \xrightarrow{A} [R]$: is a *left-focusing* sequent (with focus on A).

3. The sequent $[\Gamma], \Theta \longrightarrow \mathcal{R}$ is an *unfocused sequent.* Here, Γ contains negative formulas and positive atoms, and $\mathcal{R}$ is either in brackets, written as $[R]$, or without brackets.
4. The sequent $[\Gamma] \longrightarrow [R]$ is an instance of the previous sequent where Θ is empty.

As an inspection of the inference rules of LJF reveals, the search for a *focused* proof is composed of two alternating phases and these phases are governed by polarities. The *asynchronous phase* applies invertible (asynchronous) rules until exhaustion: no backtracking during this phase of search is needed. The asynchronous phase uses the third type of sequent above (the unfocused sequents): in that case, Θ contains positive or negative formulas. If Θ contains positive formulas, then an introduction rule (either $\wedge_l, \exists_l$ or false_l) is used to decompose it; if it is negative, then the formula is moved to the Γ context (by using the $[]_l$ rule). The end of the asynchronous phase is represented by the fourth type of sequent. Such a sequent is then established by using one of the decide rules, D_r or D_l. The application of one of these decide rules then selects a formula for focusing and switches proof search to the *synchronous phase* or *focused phase.* This focused phase then proceeds by applying sequences of inference rules on focused formulas: in general, backtracking may be necessary in this phase of search.

We use an extension of LJF, called LJF^t. The sequents in LJF^t are the same four kinds of sequents except that we add a polarity declaration, $\mathcal{P}$, to all of them: if an atom is an instance of an atom in the set $\mathcal{P}$ then it is considered positive; otherwise it is considered negative. The multicut rule is the only rule that can change the declaration $\mathcal{P}$. In particular, the *polarized version* of the multicut rule is given as

$$\frac{\mathcal{P}; [\Gamma] \longrightarrow [L_1] \qquad \cdots \qquad \mathcal{P}; [\Gamma] \longrightarrow [L_n] \qquad \mathcal{P} \cup \Delta_P; [\Gamma \cup \Delta_L] \longrightarrow [R]}{\mathcal{P}; [\Gamma] \longrightarrow [R]}\ mc.$$

Where $\Delta_L = \{L_1, \ldots, L_n\}$ is a set of formulas, and Δ_P is the set of atoms appearing anywhere in Δ_L.

Since we no longer use LJF, but use its extension LJF^t, we name a rule in LJF^t with the same name of its corresponding rule in LJF. The following proposition can be proved by a simple induction on the depth of the cut free proofs.

Proposition 1. *LJF^t is sound and complete with respect to LJF.*

4 Tables as Multicut Derivations

We consider a table as a partially ordered finite set of formulas.

Definition 1. *A* table *is a tuple $\mathcal{T} = \langle \mathcal{F}, \prec \rangle$, where $\mathcal{F}$ is some finite set of formulas, and $\prec$ is a partial order relation over the elements of $\mathcal{F}$.*

The intended meaning of a table is to be a collection of provable formulas (from some assumed context, say, Γ). The order relationship $B \prec C$ is intended to denote the requirement that the formula B is not to be proved during a proof C: that is, if an attempt to prove the formula C results in the subgoal B, then B should not be proved: since B is also in the table and it has a proof.

Tables can arise in a number of situations: in automated deduction via previously proved subgoals [16, 14], or in interactive theorem proving via a sequence of lemmas introduced by a user to direct the proof system towards its final theorem [3].

We use multicut rules to build from a table a sequent calculus derivation, and introduce the lemmas stored in the table to the assumptions (the left-hand-side of the sequent) of a program. The order in which the cuts appear in such derivation is important since a lemma, A, in the table might be required in the proof of another lemma, B, if for example $A \prec B$.

Definition 2. *Let $\mathcal{T} = \langle \mathcal{F}, \prec \rangle$ be a table. The* multicut derivation *for $\mathcal{T}$ and the sequent $\mathcal{S} = \mathcal{P}; \Gamma \longrightarrow G$, written as $mcd(\mathcal{T}, \mathcal{S})$, is defined inductively as follows: if $\mathcal{F}$ is empty then $mcd(\mathcal{T}, \mathcal{S})$ is the derivation containing just the sequent $\mathcal{P}; \Gamma \longrightarrow G$. Otherwise, if $\{F_1, \ldots, F_n\}$ is the collection of $\prec$-minimal elements in $\mathcal{F}$ and if Π is the multicut derivation for the smaller table $\langle \mathcal{F} \setminus \{F_1, \ldots, F_n\}, \prec \rangle$ and the sequent $\mathcal{P}'; \Gamma, F_1, \ldots, F_n \longrightarrow G$, then $mcd(\mathcal{T}, \mathcal{S})$ is the derivation*

$$\frac{\mathcal{P}; \Gamma \longrightarrow F_1 \quad \cdots \quad \mathcal{P}; \Gamma \longrightarrow F_n \quad \begin{matrix} \Pi \\ \mathcal{P}'; \Gamma, F_1, \ldots, F_n \longrightarrow G \end{matrix}}{\mathcal{P}; \Gamma \longrightarrow G}\; mc$$

Multicut derivations are always open *derivations (that is, they contain leafs that are not proved). A* proof of a mulitcut derivation *is any (closed) proof that extends this open derivation.*

The following proposition states when a multicut derivation of table and a initial sequent can be extended to a proof.

Proposition 2. *Let Γ be a collection of formulas, G be a formula, and $\mathcal{T}$ be a table of formulas, all of which are provable from Γ (the partial order is not restricted). The sequent $\Gamma \longrightarrow G$ is intuitionistically provable if and only if the open derivation $mcd(\mathcal{T}, \Gamma \longrightarrow G)$ can be extended to a proof in LJF^t.*

Proof The proof in the forward direction is by induction on the length of the longest path in the table's partial order. The converse is proved by forgetting the polarity information and using cut-elimination. □

5 Hereditary Harrop Lemmas

In this section and the next one, we investigate tables containing hereditary Harrop formulas [11], which are the D-formulas in the following grammar.

$$G := true \mid A \mid G_1 \wedge G_2 \mid \exists x\, G. \mid \forall x\, G \mid D \supset G$$

$$D := A \mid G \supset A \mid D_1 \wedge D_2 \mid \forall x\, D.$$

Before we start, we discuss more about focused systems, more specifically about decide rules. When the asynchronous phase of proof search ends, that is, when all the invertible rules have been applied, the decide rules, namely D_l and D_r, choose a formula on which search should focus. Since logic programs generally contain many formulas, the choice made by these decide rules is a form of *don't know* non-determinism, which is a potential source of backtracking. For example, while the sequent $[A_1, A_1 \supset A_0, A_2 \supset A_0] \longrightarrow [A_0]$ has four formulas on which to focus, a valid LJF proof can be built on by focusing on the formula $A_1 \supset A_0$ (here, A_0, A_1, A_2 are atomic formulas).

It is in our experience that searching for proofs that contains a path with many alternations between asynchronous and synchronous phases is *harder*, that is there is more potential backtracking, than proofs that contain several branches but with fewer alternations between these phases. The following definition specifies the notion of *decide depth* of a proof.

Definition 3. *The* decide-depth *of an LJF^t proof Ξ is the maximum number of occurrences of decide rules (*i.e., *D_r and D_l) on any path from the root to a leaf in Ξ.*

The program that is specified in Figure 2 is used to compute the decide depth of the (re)proof of a lemma, as states the following proposition.

$$
\begin{array}{l}
decDepth^a(A, []) = 1\\
decDepth^a(A, L) = 2 + decDepth^s_L(L)\\
decDepth^a(\forall x\, F, L) = decDepth^a(F)\\
decDepth^a(F_1 \supset F_2, L) = decDepth^a(F_2, [F_1 \mid L])\\
decDepth^a(\exists x\, F, L) = max(decDepth^s(F), decDepth^s_L(L)) + 1\\
decDepth^a(F_1 \wedge F_2, L) = max(decDepth^s_L(L), decDepth^s(F_1), decDepth^s(F_2)) + 1\\
decDepth^s(F_1 \supset F_2) = decDepth^a(F_1 \supset F_2, \emptyset)\\
decDepth^s(F_1 \wedge F_2) = max(decDepth^s(F_1), decNum^s(F_2))\\
decDepth^s(\forall x\, F) = decDepth^a(\forall x\, F, \emptyset)\\
decDepth^s(\exists x\, F) = decDepth^s(F)\\
decDepth^s(A) = 0\\
decDepth^s_L([]) = 0\\
decDepth^s_L([F \mid L]) = max(decDepth^s(F), decDepth^s_L(L))
\end{array}
$$

Fig. 2. Program that calculates the decide depth of the proof necessary to reprove a lemma. The predicate $decDepth^a$ (respectively $decDepth^s$) computes the decide depth when the proof is in the asynchronous phase (respectively synchronous phase). The predicate $decDepth^s_L$ computes the maximum decide depth of the (re)proof of the formulas in a list.

Proposition 3. *Let Γ be a collection of D-formulas, let $F \in \Gamma$ be a formula such that all of its atoms are in $\mathcal{P}$, and let F' be an instance of F. Then there is a proof for $\mathcal{P}; [\Gamma] \longrightarrow [F']$ with decide-depth of $decDepth^a(F, [])$.*

Proof Proof by structural induction on F. The most interesting case is when $F = F_1 \supset F_2$. In the first asynchronous phase, some formulas augment the program through a $\supset_r$. When this phase ends, the formula F can be decided on (the left), and the synchronous phase begins. This phase behaves in a symmetric way as the first asynchronous phase, and the same number of $\supset_l$ rules occurs as the number of $\supset_r$ rules applied in the asynchronous phase. The left premise of these $\supset_l$ occurrences are exactly a right-focused-sequent of the formulas that augmented the program in the asynchronous phase. This corresponds exactly to the specification of $decDepth^a$, where the formulas that augment the program in the asynchronous phase are stored in a list, and later used by the $decDepth^s_L$ predicate, that represents the left premises of the $\supset_l$ of the corresponding synchronous phase. Since these two phases are symmetric, after the synchronous phase ends, induction hypotheses can be used to finish the proof. $\square$

A consequence of this proposition is that, if a table contains only Horn lemmas, reproving a lemma requires the search for a proof with only two decide rules. For example:

$$\cfrac{\cfrac{\begin{array}{c}\cfrac{}{\mathcal{P};[\Gamma']-_{B_1\bar{x}}\!\rightarrow}\,I_r \quad\cdots\quad \cfrac{}{\mathcal{P};[\Gamma']-_{B_n\bar{x}}\!\rightarrow}\,I_r\\ \vdots\\ \mathcal{P};[\Gamma']-_{B_1\bar{x}\wedge\ldots\wedge B_n\bar{x}}\!\rightarrow\end{array} \qquad \cfrac{\cfrac{\cfrac{}{\mathcal{P};[\Gamma',A\bar{x}]-_{A\bar{x}}\!\rightarrow}\,I_r}{\mathcal{P};[\Gamma',A\bar{x}]\longrightarrow[A\bar{x}]}\,D_l}{\mathcal{P};[\Gamma']\xrightarrow{A\bar{x}}[A\bar{x}]}\,R_l,[]_l}{\mathcal{P};[\Gamma,\forall\bar{x}\,(B_1\bar{x}\wedge\ldots\wedge B_n\bar{x}\supset A\bar{x}),B_1\bar{x},\ldots,B_n\bar{x}]\longrightarrow[A\bar{x}]}\,[]_r,D_l,\forall_l,\supset_l}{\mathcal{P};[\Gamma,\forall\bar{x}\,(B_1\bar{x}\wedge\ldots\wedge B_n\bar{x}\supset A\bar{x})]\longrightarrow\forall\bar{x}\,(B_1\bar{x}\wedge\ldots\wedge B_n\bar{x}\supset A\bar{x})}\,\forall_r,\supset_r,\wedge_l^*[]_l^*$$

Proposition 3 extends the results obtained in [12]. There lemmas were restricted to atoms, and reproving required only one decide rule in the asynchronous phase, and no decide rule in the synchronous phase, which are just subcases in the program depicted in Figure 2.

By allowing the use of formulas, instead of only atoms, in tables, we increase considerably the possibilities of using lemmas (when compared to the tables in [12]). For example, consider a Horn program composed of two clauses: $p\,(f\,z)$ and $\forall x\,(p\,x \supset p\,(f\,x))$; and the query $p\,(f^n\,z)$, where $(f^n\,x)$ denotes that there are n consecutive applications of f. If a logic programming interpreter is given such program and query, it would take exactly n decide rules to find a proof. However, by using a well chosen set of lemmas, one can reduce that number significantly, for example, if a table with the following lemmas is given: $\forall x\,(p\,x \supset p\,(f^2\,x))$, $\forall x\,(p\,x \supset p\,(f^4\,x))$, $\ldots$, $\forall x\,(p\,x \supset p\,(f^m\,x))$. The proof can be found with exponentially fewer occurences of decide rules. To prove each lemma it is required only two decide rules, as illustrated below:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{P}; [p\,x, \Gamma]-_{p\,x}\!\!\longrightarrow}\,I_r \qquad \dfrac{\dfrac{\dfrac{}{\mathcal{P}; [p\,x, \Gamma]-_{p\,(f^i\,x)}\!\!\longrightarrow}\,I_r \qquad \dfrac{\dfrac{}{[\mathcal{P}; p\,x, p\,(f^i\,x), p\,(f^{2i}\,x), \Gamma]-_{p\,(f^{2i}\,x)}\!\!\longrightarrow}\,I_r}{\mathcal{P}; [p\,x, p\,(f^i\,x), p\,(f^{2i}\,x), \Gamma] \longrightarrow [p\,(f^{2i}\,x)]}\,D_r}{\mathcal{P}; [p\,x, \Gamma] \xrightarrow{\forall x\,(p\,x \supset p\,(f^i\,x))} [p\,(f^{2i}\,x)]}\,\forall_l, \supset_l, R_l, []_l}{\mathcal{P}; [p\,x, p\,(f^i\,x), \Gamma] \longrightarrow [p\,(f^{2i}\,x)]}\,D_l}{\mathcal{P}; [p\,x, \Gamma] \xrightarrow{\forall x\,(p\,x \supset p\,(f^i\,x))} [p\,(f^{2i}\,x)]}\,\forall_l, \supset_l, R_l, []_l}{\mathcal{P}; [p\,x, \Gamma] \longrightarrow [p\,(f^{2i}\,x)]}\,D_l}{\mathcal{P}; [\Gamma] \longrightarrow \forall x\,(p\,x \supset p\,(f^{2i}\,x))}\,\forall_r, \supset_r, []_l, []_r$$

6 Tables as Proofs

In the previous sections, we used a multicut derivation, which could be represented by a partially ordered set of formulas (hereditary Harrop formulas). In these multicut derivations, all the lemmas in a set are inserted to a sequent's context already in the beginning of the proof tree. This gives rise to two problems:

First, all the lemmas must be provable from the initial context. In a proof it might happen that the context is augmented with a formula, for instance, by the $\supset_r$ rule. One could imagine that less complex lemmas could be used if this new context is considered, in particular, if the formula included in the context is part of the body of the lemma. For example, consider that the lemma $A \supset B$ is provable from the initial context Γ. In addition, consider that this lemma is only used later in the proof when the context Γ is augmented with the formula A. One could imagine that instead of including the previous lemma in the beginning of the proof, it would be better to include the more simple lemma, B, later when the context is already augmented with the formula A. As we show in the previous section, it is simpler, in the sense of number of decide rules to be made, to reprove the simpler lemma, B, than to reprove the lemma $A \supset B$.

Second, lemmas might be only necessary in some parts of the proof, but by including them in the context already in the beginning of the proof, all the lemmas are present in all branches of the tree, even in the branches where lemmas are not necessary. This is a problem because it increases the non-determinism of the decide rules, since there would be more formulas in the context to focus on.

We now propose a more complex derivation, where multicuts derivations may appear, not only in the beginning of the proof, but also in the middle of a proof. Instead of a partially ordered set of formulas, this new derivation will be represented by a partially ordered set of multicut derivations. The partial order denotes the provability dependency between the multicut derivations. The architecture of this new derivation is depicted in Figure 3. The idea is that each multicut derivation is only used when there is a change in the context, for example, by $\supset_r$ rule. Since the context can only be modified by the asynchronous

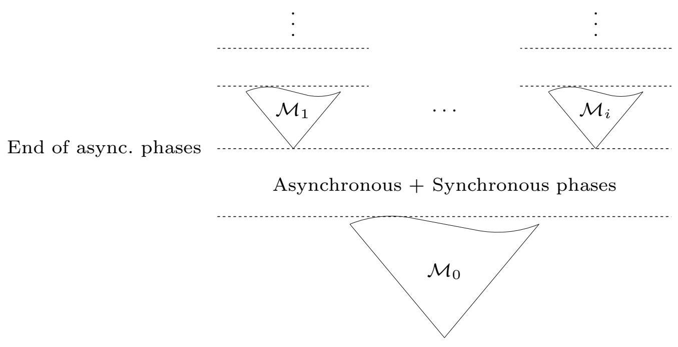
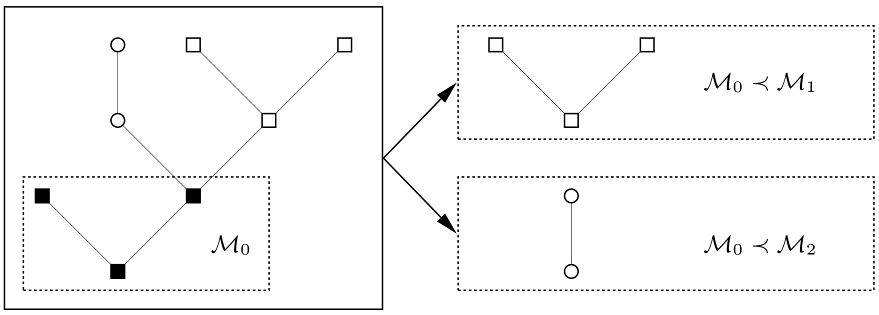


Fig. 3. Illustration of a derivation obtained from a partially ordered set of multicut derivations. The triangles are multicut derivations, the parallel dashed lines represents the end of an asychronous phase, and the spaces between these lines are sequences of asynchronous and synchronous phases. Here, the following relations hold: $\mathcal{M}_0 \prec \mathcal{M}_1$ and $\mathcal{M}_0 \prec \mathcal{M}_i$.

rules $\forall_r, \exists_l$, and $\supset_r$, focusing provides a natural way to identify when to use a multicut derivation, namely, whenever an asynchronous phase ends. Moreover, if for instance there are two partial orders where, say, $\mathcal{M}_0 \prec \mathcal{M}_1$ and $\mathcal{M}_0 \prec \mathcal{M}_2$, but there are no relations between $\mathcal{M}_1$ and $\mathcal{M}_2$, the multicuts $\mathcal{M}_1$ and $\mathcal{M}_2$ are used in two different branches of $\mathcal{M}_0$. The sequent associated with the multicut (recall that a multicut is obtained from a table and a root sequent) identifies which branch this multicut derivation should be used.

One could argue that using the sequent associated to the multicut derivation to identify in which branch this derivation must be used is not convincing. It would be necessary to identify if two sequents are identical, and checking if two contexts are identical may not be a trivial task. In practice, it will not be required to check if two contexts are identical, but rather check if the formula to be proved (that is the right-hand-side of the sequent) can be proved by the multicut derivation, and if all lemmas used in this derivation are provable by the context of the sequent where the derivation will be used.

Even though we are mostly concerned in using tables and multicut derivations than discovering them, we now specify how to extract from a finite (goal-directed) proof tree a collection of partially ordered set of multicut derivations. We only table sequents of the form $[\Delta] \longrightarrow [A]$, that is, a sequent where no more asynchronous rules can be applied. Given a sequence of multicut derivations $\mathcal{M}_1 \prec \ldots \prec \mathcal{M}_n$ where $n \geq 0$, and a LJF proof tree, Ξ, with root $[\Gamma] \longrightarrow [A]$ and all atoms with negative polarity (that is a goal directed proof), we procede as follows: First, extract the subtree Ξ_Γ of Ξ that contains all nodes in Ξ with context $[\Gamma]$. Second, compute the table, $\mathcal{T}$, obtained from the postorder traversal of Ξ_Γ, that is, a depth first traversal, where the children of a tree are visited first (see [12] for more details). Third, increment the inputed sequence with the new multicut derivation, $\mathcal{S} = \mathcal{M}_1 \prec \ldots \prec \mathcal{M}_n \prec mcd(\mathcal{T}, [\Gamma] \longrightarrow [A])$. Repeat this process for all the children subtrees of Ξ_Γ and sequence $\mathcal{S}$, until there are no more nodes in Ξ to be extracted. The output of this procedure is a collection of sequences of multicut derivations. We illustrate this procedure in Figure 4 with an example. The following proposition, which is proved by induction on the height of the tree, states how a derivation that is built in this way could be extended to a proof.



$\mathcal{M}_0$
$\mathcal{M}_0 \prec \mathcal{M}_1$
$\mathcal{M}_0 \prec \mathcal{M}_2$

Fig. 4. Illustration of the procedure to obtain a collection of partially ordered set of tables from an existing (goal directed) proof. The three different kinds of nodes (filled squares, ellipses, and blank squares) represent sequents with different contexts. First, the subtree with the filled squares is extracted, and, from it, the table $\mathcal{M}_0$ is constructed with the table obtained by using a postorder traversal in this tree and tree's root. Later, this procedure is repeated with the two remaining subtrees, namely the one with elliptical nodes and the one with blank squares, obtaining, respectively, the multicut derivations $\mathcal{M}_1$ and $\mathcal{M}_2$. Finally, these data structures are added to the inputed sequence, which contains only the multicut derivation $\mathcal{M}_0$, and outputs the two sequences depicted in the figure.

Proposition 4. *Let Ξ be a LJF proof, $\mathcal{M}$ be a partially ordered set of multicut derivations, obtained from Ξ by using the procedure described before, and $\Xi_{\mathcal{M}}$ be a derivation obtained from $\mathcal{M}$. There is a proof for $\Xi_{\mathcal{M}}$ where all added subproofs have an unique occurrence of the D_l rule.*

Given that it is easy to check if a partially ordered set of multicut derivations is derived from a cut-free proof, one could consider this set as a legitimate proof object.

In the proof carrying code setting [13], proof objects are transmitted together with mobile codes to assure that some (safety) properties are satisfied by these programs. The client checks that the programs are safe to be executed by checking the proof object associated with them. Some engineering characteristics are paramount for proof objects. They should not be too large, so that transmission costs are reduced, and not be hard to check, so that the consumer of these objects doesn't require many resources. Furthermore, as proof objects can be reused later with other consumers, these object need to be flexible enough to guarantee a large scope of possible consumers.

A partially ordered set of multicut derivations seems to be a good option. First, the trade-offs between the size of the proof object and the complexity for checking it are fairly understood. If all atoms are tabled, as in Proposition 4, checking if the proof object is valid is easy. However, one could consider to reduce the size of the proof object, by not tabling some atoms. Clearly, the consumer would have to prove these atoms, increasing the complexity of checking the proof object. Second, tables represent only the declarative information, that is all the atoms that are provable, and not the information about how to prove the lemmas. This allows proof objects to be flexible enough to be used by different implementations of search engines, and, therefore, by several consumers.

It is worth noticing that consumers would still require a simple proof search engine, for example, an engine that implements LJF^{t}[1]. One could argue that this inclusion would affect the consumer's trust base by including many lines of code to it. We follow the arguments given by Pollack with respect to theorem provers [15], and argue that simple proof engines have been tested and studied for decades, and there is a considerable amount of evidence that such implementations are trustworthy. Furthermore, we believe that the amount of code included in a consumer's trust base is not prohibitively high, since for checking a table, very simple proof engines (for example, without tabling mechanisms) can be used.

7 Conclusions and Future Works

The results in this paper are still not strong enough to be realistically used in theorem provers, since interesting theorems usually requires more than just computations, but also, for instance, induction. A future work is to investigate the use of focused system, more specifically the polarity of atoms, in more expressive logics [**?**,10]. The use of tables as proof objects still needs more experimental results, that should be done soon.

We showed here that by "playing" adequately with the polarity of atoms in a focused system, it is possible to reprove lemmas easily. We also discussed the use of tables as a proof object, and described a way to extract and use a sequence of multicut derivations from an existing goal directed proof.

Acknowledgments I thank Dale Miller for our fruitful discussions and for his helpful comments. This work has been supported in part by INRIA through the "Equipes Associées" Slimmer and by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project.

[1] This differs from traditional PCC proof object proposals where the proof is given entirely by its producer, and consumers are not required to perform non-deterministic search.

References

1. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.
2. D. Aspinall, S. Gilmore, M. Hofmann, D. Sannella, and I. Stark. Mobile resource guarantees for smart devices. In G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, and T. Muntean, editors, *Construction and Analysis of Safe, Secure and Interoperable Smart Devices: Proceedings of the International Workshop CASSIS 2004*, volume 3362 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 2005.
3. Robert S. Boyer and J. Strother Moore. *A Computational Logic.* Academic Press, 1979.
4. Kaustuv Chaudhuri. *The Focused Inverse Method for Linear Logic.* PhD thesis, Carnegie Mellon University, December 2006. Technical report CMU-CS-06-162.
5. Gilles Dowek, Amy Felty, Gèrard Huet Hugo Herbelin, Chet Murthy, Catherine Parent, Christine Paulin-Mohring, and Benjamin Werner. The coq proof assistant user's guide. Technical report, INRIA, Rocquencourt, France, 1993.
6. R. Dyckhoff and S. Lengrand. LJQ: a strongly focused calculus for intuitionistic logic. In A. Beckmann et al, editor, *Computability in Europe 2006*, volume 3988 of *LNCS*, pages 173–185. Springer Verlag, 2006.
7. Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.
8. Radha Jagadeesan, Gopalan Nadathur, and Vijay Saraswat. Testing concurrent systems: An interpretation of intuitionistic logic. In *Proceedings of FSTTCS*, 2005.
9. Chuck Liang and Dale Miller. Focusing and polarization in intuitionistic logic. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 451–465. Springer-Verlag, 2007.
10. Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232:91–119, 2000.
11. Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
12. Dale Miller and Vivek Nigam. Incorporating tables into proofs. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, volume 4646 of *LNCS*, pages 466–480. Springer-Verlag, 2007.
13. George C. Necula. Proof-carrying code. In *Conference Record of the 24th Symposium on Principles of Programming Languages 97*, pages 106–119, Paris, France, 1997. ACM Press.
14. Brigitte Pientka. Tabling for higher-order logic programming. In *20th International Conference on Automated Deduction, Talinn, Estonia*, pages 54 – 69. Springer-Verlag, 2005.
15. Robert Pollack. How to believe a machine-checked proof. In G. Sambin and J. Smith, editors, *Twenty Five Years of Constructive Type Theory.* Oxford University Press, 1998.
16. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, Scott A. Smolka, Terrance Swift, and David Scott Warren. Efficient model checking using tabled resolution. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV97)*, number 1254 in LNCS, pages 143–154, 1997.