Formulas Rewritten and Normalized Computationally, and Intuitionistically Simplified

Hugo Férée, Sam V Gool, Yago Iglesias Vázquez

Formulas Rewritten and Normalized Computationally, and Intuitionistically Simplified

Hugo Férée[1], Sam v. Gool[1], and Yago Iglesias Vázquez[1]

[1]IRIF, Université Paris Cité, Paris, 75013, France

Abstract: We give a verified implementation of a procedure for simplifying formulas of intuitionistic propositional logic, relying on a particular contraction-free sequent calculus for this logic. We apply this procedure to simplify the output given by a verified program for computing propositional quantifiers of such formulas, developed by the the first- and second-named authors in earlier work. This also allows us to simplify and improve the efficiency of that program, as well as the proof of its correctness.

1 Introduction

This paper is a report on work in progress on the following question:

$$\begin{array}{l}\text{Given a propositional formula } \varphi\text{, how to compute}\\ \text{a 'simpler' formula } \varphi' \text{ that is intuitionistically equivalent to } \varphi?\end{array} \tag{1}$$

Here, by a propositional *formula* we mean any term built from variables, here denoted by letters $p, q, \ldots$, using binary operations $\vee, \wedge, \to$, and a nullary operation $\bot$. Alternatively, one may think of such a formula as a type in a simply typed λ-calculus with zero, sum, and product types. Throughout the paper, by *equivalence* of two formulas φ and φ' we mean that the formula $(\varphi \to \varphi') \wedge (\varphi' \to \varphi)$ is provable, or, on the other side of the Curry-Howard isomorphism, inhabited. The precise definition of the adjective 'simpler' in (1) that we use here depends on a function associating to each formula a natural number, its *weight*. We give the function in Definition 1 below; for now, it suffices to say that it satisfies the intuitive property that shorter formulas have lower weight.

Since intuitionistic equivalence is decidable, there evidently exists a simplest formula in any equivalence class, and such a formula is computable by a brute-force enumeration. However, we here seek a practically feasible, and provably correct, answer to the question (1). The immediate origin of the question, for us, lies in a verified Coq/Rocq implementation by the first- and second-named authors [FG23] of Pitts' procedure [Pit92] for computing propositional quantifiers in intuitionistic logic. Given a propositional formula φ and a propositional variable p, the formulas $\exists p.\varphi$ and $\forall p.\varphi$ are abstractly characterized by the usual rules of quantification, see Definition 2. The surprising result of [Pit92] is that for any φ, there actually exist *propositional* formulas $\mathsf{E}_p(\varphi)$ and $\mathsf{A}_p(\varphi)$ that are equivalent to $\exists p.\varphi$ and $\forall p.\varphi$, respectively. This implies in particular that second-order propositional quantifiers may in fact be encoded, up to equivalence, in the zero-order fragment. Pitts' theorem has many consequences for higher-order intuitionistic logic, including a uniform interpolation

theorem and the existence of a model completion for the class of Heyting algebras, further see [GZ02]. Since the implementation of [FG23], researchers in intuitionistic logic have also become interested in computing the formulas $\mathsf{E}_p(\varphi)$ and $\mathsf{A}_p(\varphi)$ on specific examples; see for example [Koc23] and also our remarks in Section 5 below.

A main issue that we already identified in [FG23] was that, even on relatively small inputs, the formulas that were output by our program were very large; we will quantify this in Table 4 below. In all specific cases of interest, one could show manually that the output would admit substantial simplifications, but developing an algorithm for applying such simplifications was both a conceptual and technical challenge. Our main contribution here is to give such a simplification algorithm for intuitionistic formulas, accompanied by a verified implementation in Coq/Rocq. Further, by interleaving Pitts' original recursive procedure with our simplification procedure, we obtain a program with much improved efficiency. This moreover allows us to simplify Pitts' procedure itself both on paper and in the verified implementation, building also on recent insights from work in [FGGS24].

Our approach, which we explain in more detail in the rest of this paper, relies on the sequent calculus G4iP, which was also used by Pitts in [Pit92] and implemented in [FG23]. The first step is to use this calculus to obtain a decision procedure for the logic, for which we provide a verified implementation, as we explain in Section 3. We subsequently use the decision procedure to algorithmically simplify both sides of a sequent. Both the decision and simplification procedures are guided by the invertible rules of the sequent calculus. To be able to use the decision procedure inside the simplification procedure, we use admissibility of cut and weakening. For this, we give a verified proof of cut admissibility for the sequent calculus G4iP, relying on the formally verified proof of [SvdGGI23], by integrating the ideas developed there into our existing Coq/Rocq development. We give more details in Section 4.

In order to make our simplification algorithm usable for calculating propositional quantifiers, we first give a formal proof in Coq/Rocq that these simplification functions are correct, in the sense that they always compute equivalent formulas and contexts of lower weight. Given a correct simplification function, we interleave it with Pitts' calculation at each recursive call, and then show, with a relatively low amount of effort, that the newly calculated propositional quantifiers are still correct. This methodology means that one may still improve the simplification function later, without touching the rest of the correctness proof for the propositional quantifiers.

This paper should be read as a report on work in progress, which we believe opens up a number of interesting new questions: What is the theoretical complexity of the simplification problem for a given weight function? To what extent is the simplification function that we develop here optimal? And is it even practically worthwhile to find an optimal such function, if it makes the algorithm more complex to verify?

The Coq/Rocq development is available online at `https://github.com/hferee/UIML/`. A web demo of the calculator is available at `https://hferee.github.io/UIML/`, which also links to online documentation of the Coq/Rocq code. Throughout this paper, we provide links to relevant Coq/Rocq declarations under a clickable symbol 🐓. The reference commit for the code discussed in this paper is `44d4c8e`.

2 Sequents, proofs, and propositional quantifiers

A *formula* is a term built from variables and $\bot$ using $\to$, $\vee$ and $\wedge$. The symbol $\top$ is defined as $\bot \to \bot$ and, for any formula φ, $\neg\varphi$ is defined as $\varphi \to \bot$. A *context* is a finite list of formulas, and is typically denoted with a capital Greek letter Γ or Δ. We denote the concatenation of contexts Γ and Δ by Γ, Δ, or occasionally $\Gamma \bullet \Delta$, and we abuse notation to write expressions like Γ, φ and $\Gamma \bullet \varphi$ for the concatenation of Γ with the one-element context containing only φ. A *sequent* is a pair of a context and a formula, denoted $\Gamma \Rightarrow \varphi$. We now recall a notion of *weight* on formulas and sequents [Pit92].

Definition 1 (🐓)**.** The *weight of a formula* (🐓) is inductively defined as:

$$\begin{cases} \text{weight}(\bot) & = 1 \\ \text{weight}(q) & = 1 \\ \text{weight}(\varphi \vee \psi) & = 1 + \text{weight}(\varphi) + \text{weight}(\psi) \\ \text{weight}(\varphi \wedge \psi) & = 2 + \text{weight}(\varphi) + \text{weight}(\psi) \\ \text{weight}(\varphi \to \psi) & = 1 + \text{weight}(\varphi) + \text{weight}(\psi) \end{cases}$$

The *weight of a context* Γ (🐓) is then defined as $\text{weight}(\Gamma) \stackrel{\text{def}}{=} \sum_{\varphi \in \Gamma} 5^{\text{weight}(\varphi)}$. For a sequent $\Gamma \Rightarrow \varphi$, its weight is defined as $\text{weight}(\Gamma \bullet \varphi \bullet \varphi)$. The doubling of the right hand side formula φ is a detail that is only needed for treating rules in intuitionistic modal logic.

The interest of this notion of weight is that it yields a well-founded ordering on formulas and on sequents, where $\Gamma \Rightarrow \varphi$ is considered smaller than $\Gamma' \Rightarrow \varphi'$ if it has lower weight.

Throughout the paper, we make use of the sequent calculus G4iP for intuitionistic propositional logic [Vor70, Hud88, Dyc92]. We recall the rules of the calculus in Fig. 1. The calculus was extended to intuitionistic strong Löb logic in [SvdGGI23], giving a sequent calculus G4iSL for this modal logic. Certain portions of our formalization work, notably on cut admissibility and decidability, are also done for that calculus, but in this paper we focus on our work for G4iP, the non-modal part.

The calculus G4iP has the property that each sequent in the antecedent of a rule is strictly smaller than the conclusion of the rule. Also, to any given sequent, only finitely many instances of the rules can be applied. It follows from this that the calculus is terminating.

$$\frac{}{\bot, \Gamma \Rightarrow \chi}\ (\bot\text{L}) \qquad \frac{}{\Gamma, p \Rightarrow p}\ (\text{IdP}) \qquad \frac{\Gamma, \varphi, \psi \Rightarrow \chi}{\Gamma, \varphi \wedge \psi \Rightarrow \chi}\ (\wedge\text{L}) \qquad \frac{\Gamma, \varphi \Rightarrow \psi}{\Gamma \Rightarrow \varphi \to \psi}\ (\to\text{R})$$

$$\frac{\Gamma, p, \varphi \Rightarrow \chi}{\Gamma, p, p \to \varphi \Rightarrow \chi}\ (p \to\text{L}) \qquad \frac{\Gamma, \varphi \to (\psi \to \chi) \Rightarrow \delta}{\Gamma, (\varphi \wedge \psi) \to \chi \Rightarrow \delta}\ (\wedge \to\text{L}) \qquad \frac{\Gamma, \varphi \to \chi, \psi \to \chi \Rightarrow \delta}{\Gamma, (\varphi \vee \psi) \to \chi \Rightarrow \delta}\ (\vee \to\text{L})$$

$$\frac{\Gamma, \varphi \Rightarrow \chi \qquad \Gamma, \psi \Rightarrow \chi}{\Gamma, \varphi \vee \psi \Rightarrow \chi}\ (\vee\text{L}) \qquad \frac{\Gamma \Rightarrow \varphi \qquad \Gamma \Rightarrow \psi}{\Gamma \Rightarrow \varphi \wedge \psi}\ (\wedge\text{R})$$

$$\frac{\Gamma \Rightarrow \varphi_i}{\Gamma \Rightarrow \varphi_1 \vee \varphi_2}\ (\vee\text{R}_i), i \in \{1,2\} \qquad \frac{\Gamma, \psi \to \chi \Rightarrow \varphi \to \psi \qquad \Gamma, \chi \Rightarrow \delta}{\Gamma, (\varphi \to \psi) \to \chi \Rightarrow \delta}\ (\to \to\text{L})$$

Figure 1. The sequent calculus G4iP. 🐓

A *proof* in G4iP of a sequent $\Gamma \Rightarrow \varphi$ is a finite rooted tree whose nodes are labeled by sequents, so that every node together with its (upward) children is an instance of one of the rules in Fig. 1, and the root is labeled $\Gamma \Rightarrow \varphi$. We write $\Gamma \vdash \varphi$ if there exists a proof of the sequent $\Gamma \Rightarrow \varphi$ in G4iP, and we call φ *provable* if $\emptyset \vdash \varphi$. Since weakening is admissible (🐓), it follows that, if φ is provable, then $\Gamma \vdash \varphi$ for any Γ. An important observation about the weight function given in Definition 1 is that the weight of sequents decreases strictly as one moves upward in G4iP.

For the purpose of the decision procedure that we will discuss below, it is useful to classify the rules of G4iP according to their number of premises, and their invertibility. Here, recall that a rule is *invertible* if, whenever its conclusion is provable, all of its premises must be provable. We also call a rule *linear* if it has only one premise. With this classification, we observe:

- 0 premises: $\bot$L and IdP.
- 1 premise and invertible: $\wedge$L, $\to$R, $p \to$L, $\wedge \to$L, and $\vee \to$L.

- 2 premises and invertible: $\vee$L and $\wedge$R.
- not invertible: $\vee R_1, \vee R_2$ and $\to\to$L.

Pitts' theorem [Pit92] says that, for any formula φ and propositional variable p, the formulas $\exists p.\varphi$ and $\forall p.\varphi$ can be expressed by quantifier-free propositional formulas. We now recall a formal definition of the meaning of these propositional quantifications.

Definition 2 (Uniform interpolants)**.** Let φ be a formula and p a propositional variable. A formula $\exists p.\varphi$ is a *right uniform interpolant* for φ with respect to p if it is a p-free formula such that $\varphi \vdash \exists p.\varphi$, and for any formula ψ, if $\varphi \vdash \psi$ then $\exists p.\varphi \vdash \psi$, i.e. $\exists p.\varphi$ is the strongest p-free formula that is entailed by φ.

Dually, $\forall p.\varphi$ is a *left uniform interpolant* for φ with respect to p if it is a p-free formula such that $\forall p.\varphi \vdash \varphi$, and for any formula θ, if $\theta \vdash \varphi$ then $\theta \vdash \forall p.\varphi$, i.e. $\forall p.\varphi$ is the weakest p-free formula that entails φ.

Note that a right uniform interpolant is unique up to equivalence, and so is a left uniform interpolant. In previous work [FG23] the first- and second-named authors developed a Coq/Rocq formalization of the construction of uniform interpolants for intuitionistic propositional logic, which was extended in [FGGS24] to modal logics **K** and **GL**, and to intuitionistic strong Löb logic **iSL**.

3 Decision procedure

Given a list of formulas Γ and a formula φ, we give a procedure that decides if there exists a proof of $\Gamma \Rightarrow \varphi$ (🐓). Indeed, the inductive definition given by Figure 1 does not guarantee decidability by construction. To explain the intuition for this decision procedure, one may start from the idea of a naive proof search, which would explore all possible proof trees : Given an input sequent, for each applicable instance of a rule, one would recursively search for a proof for each of its premises. However, such a naive proof search is very inefficient.

Our implementation, for which we give the pseudocode in Fig. 2, improves on this by applying the rules in a specific order, so as to avoid unnecessary branching as much as possible: (1) we first try to apply axioms to end the proof search; (2) if this is not possible, then we try to apply linear invertible rules, so as to obtain a single equivalent subsequent, and iterate; (3) if none of these apply, then we try to apply a duplicating invertible proof, leading to two successive proof searches; (4) finally, if no other rule is applicable, try each possible instance of a non-invertible rule and continue the search, in a depth-first-search manner. This leads to the following function, expressed in pseudocode as a pattern match in Fig. 2. For the reader's convenience, we annotate each case with the name of the corresponding G4iP-rule. When defining this function in Coq/Rocq, one needs to simultaneously prove termination, which we obtain from the fact that applying G4iP-rules upwards decreases the weight of a sequent.

We found that the performance of the function in Fig. 2 is good enough to allow for repeated calls inside the normalization procedure that we describe below, without significant slow-down. On the other hand, it is likely that the above function does not have optimal theoretical complexity; more involved algorithms decide the same problem in $O(n \log n)$-space [Hud93]. However, formally verifying that such an optimized algorithm is correct would be a more difficult task, while this remains straightforward for our implementation given above.

4 Simplification procedure

In this section, we will explain how we use the above decision procedure to simplify formulas and contexts, first in general, and then specifically for the computation of Pitts' propositional quantifiers.

function $\Gamma \vdash_? \varphi$
  match Γ, φ **with**
  | $\Delta_1 \bullet \bot \bullet \Delta_2, \_ \Rightarrow$ true $(\bot L)$
  | $\_, p$ **when** $p \in \Gamma \Rightarrow$ true (IdP)
  | $\Delta_1 \bullet \delta_1 \wedge \delta_2 \bullet \Delta_2 \Rightarrow \delta_1 \bullet \delta_2 \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi$ $(\wedge \mathrm{L})$
  | $\_, \varphi_1 \to \varphi_2 \Rightarrow \varphi_1 \bullet \Gamma \vdash_? \varphi_2$ $(\to \mathrm{R})$
  | $\Delta_1 \bullet p \bullet \Delta_2 \bullet p \to \delta \bullet \Delta_3, \_ \Rightarrow \delta \bullet \Delta_1 \bullet \Delta_2 \bullet \Delta_3 \bullet p \vdash_? \varphi$ $(p \to \mathrm{L})$
  | $\Delta_1 \bullet (\delta_1 \wedge \delta_2) \to \delta_3 \bullet \Delta_2, \_ \Rightarrow (\delta_1 \to (\delta_2 \to \delta_3)) \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi$ $(\wedge \to \mathrm{L})$
  | $\Delta_1 \bullet (\delta_1 \vee \delta_2) \to \delta_3 \bullet \Delta_2, \_ \Rightarrow (\delta_1 \to \delta_3) \bullet (\delta_2 \to \delta_3) \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi$ $(\vee \to \mathrm{L})$
  | $\_, \varphi_1 \wedge \varphi_2 \Rightarrow (\Gamma \vdash_? \varphi_1)$ and $(\Gamma \vdash_? \varphi_2)$ $(\wedge \mathrm{R})$
  | $\Delta_1 \bullet \delta_1 \vee \delta_2 \bullet \Delta_2, \_ \Rightarrow (\delta_1 \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi)$ and $(\delta_2 \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi)$ $(\vee \mathrm{L})$
  | $\_, \varphi_1 \vee \varphi_2$ **when** $(\Gamma \vdash_? \varphi_1)$ or $(\Gamma \vdash_? \varphi_2) \Rightarrow$ true $(\vee \mathrm{R})$
  | $\Delta_1 \bullet (\delta_1 \to \delta_2) \to \delta_3 \bullet \Delta_2, \_$ **when** ...
    ... $(\delta_2 \bullet \delta_3 \bullet \Delta_1 \bullet \Delta_2 \vdash_? \delta_1 \to \delta_2)$ and $(\delta_3 \bullet \Delta_1 \bullet \Delta_2 \vdash_? \varphi) \Rightarrow$ true $(\to\to \mathrm{L})$
  | $\_, \_ \Rightarrow$ false
end function

Figure 2. Decision procedure for G4iP.

4.1 Simplification of formulas in context

Our simplification procedure for formulas is a recursion on the shape of the formula to be simplified. In order to explain the idea, consider how one might simplify the formula

$$p \wedge ((q \to r) \wedge (p \to q)) \ . \tag{2}$$

After two recursive calls, we will try to simplify $p \to q$. At this point, we should remember that we are simplifying the formula $p \to q$ in a context that contains p. Thus, our recursive procedure will have to keep track of the *context* in which we are simplifying as an additional parameter. This leads us to define a *contextual simplification* function, called `contextual_simp_form` in our code (), and `csimp` for short here. We give the scheme of the definition in Fig. 3 below. The simplification on formulas, `simp_form`, will then be defined as `csimp` in an empty context.

```
csimp : formula list -> formula -> bool

csimp Δ (φ1 ∧ φ2) = let φ2' := (csimp (φ1 :: Δ) φ2) in
                    choose_conj (csimp (φ2' :: Δ) φ1) φ2'
csimp Δ (φ1 ∨ φ2) = choose_disj (csimp Δ φ1) (csimp Δ φ2)
csimp Δ (φ1 → φ2) = choose_impl (csimp Δ φ1) (csimp (φ1 :: Δ) φ2)
csimp Δ φ = ⊤ if (Δ ⊢? φ), φ otherwise
```

Figure 3. Simplification of a formula in a context.

It is in the definition of this function `csimp` that we make repeated calls to the decision procedure $\vdash_?$ for G4iP: One such call, in the final case, is explicit in the definition of `csimp`, while the other calls occur in the auxiliary functions named `choose_conj` , `choose_disj` , and `choose_impl` .

The idea of the functions named `choose_*` is as follows: A conjunction $\varphi \wedge \psi$ can be simplified to φ if it happens to be the case that $\varphi \vdash \psi$, and vice versa; similarly for disjunction. For `choose_impl` we use a number of substitution instances of intuitionistic tautologies about implications: For an implication $\varphi \to \psi$ where $\varphi \vdash \bot$, since $\bot \to \psi$ is a tautology, we may simplify $\varphi \to \psi$ to $\top$. Similarly, if $\psi \vdash \bot$, then $\varphi \to \psi$ may be replaced by $\neg\varphi$. Also, if $\vdash \psi$, then $\varphi \to \psi$ is a tautology, so it simplifies to $\top$, and if $\vdash \varphi$, then $\varphi \to \psi$ simplifies to ψ.

Example 3. As an instructive example, let us run through a call `csimp` [] $p \wedge (p \to q)$. The conjunction case defines `φ2'` with a recursive call to `csimp`, with $\Delta = [p]$ and $\varphi = p \to q$. In this recursive call, we are in the implication case, and we call `csimp` Δ p and `csimp` Δ q, which both fall in the last case: The first returns $\top$, since $\Delta \vdash p$, and the second returns q, since $\Delta \nvdash q$. According to the rules of `choose_impl`, $\top \to q$ simplifies to q, which is propagated back to the original recursive call, and becomes the value of `φ2'`. We then call `csimp` $[q]$ p, but this immediately returns p. Finally, in this simple case, `choose_conj p q` just returns $p \wedge q$. For a slightly more complex example, the reader may verify that the simplification of the formula in Eq. (2) will lead to $p \wedge r \wedge q$, as expected.

We establish in Coq/Rocq the following correctness properties of the function `csimp`.

Proposition 4. *For any context* Δ *and formula* φ*, we have either* `csimp` Δ $\varphi = \top$ *or* *weight*(`csimp` Δ φ) $\leq$ *weight*(φ)*.*

Definition 5. Two contexts Δ and Δ' are *equivalent* if, for any formula φ, we have $\Delta \vdash \varphi$, if, and only if, $\Delta' \vdash \varphi$.

As an ingredient for the correctness proof, we implemented a formal proof of the admissibility of cut for the intuitionistic modal sequent calculus G4iSL using the methods from [SvdGGI23], which also gives admissibility of cut in G4iP. Given this, one may usefully observe () that, if Δ and Δ' are equivalent, then, for any context Γ and formula φ, we have $\Gamma, \Delta \vdash \varphi$ if, and only if, $\Gamma, \Delta' \vdash \varphi$. In practice, to verify that Δ and Δ' are equivalent, it suffices to prove that $\Delta \vdash \bigwedge \Delta'$ and $\Delta' \vdash \bigwedge \Delta$. We tacitly use these observations in a number of proofs discussed below.

Proposition 6. *For any context* Δ *and formula* φ*, we have* $\Delta \vdash \varphi \leftrightarrow$ `csimp` Δ φ*. In particular, the contexts* $\varphi :: \Delta$ *and* (`csimp` Δ φ) $:: \Delta$ *are equivalent.*

For use below, if Δ is a context and $\varphi \in \Delta$, we say that there is an *applicable* contextual simplification to φ in context Δ if `csimp` $(\Delta \setminus \varphi)$ φ has strictly smaller weight than φ.

4.2 Simplification of contexts

In order to use the above simplification function on formulas in the calculation of propositional quantifiers, it will be necessary to simplify *contexts*. We thus define a function `simp_env` (), which simplifies a context Δ to Δ', by iteratively applying the following rules in order:

1. First, apply a linear invertible left rule if possible. More precisely, whenever $\Delta \vdash \cdot$ is an instance of a conclusion of an invertible left rule that has assumption $\Delta' \vdash \cdot$, we replace Δ with Δ'.
2. Check whether there is $\varphi \in \Delta$ such that $\Delta \setminus \{\varphi\} \vdash \varphi$; if so, continue with $\Delta' := \Delta \setminus \{\varphi\}$.
3. Finally, check whether there is a formula φ in Δ to which a contextual simplification applies, and if so, obtain Δ' by replacing φ with `csimp` $(\Delta \setminus \{\varphi\})$ φ.

The second rule consists in weakening a redundant hypothesis, whereas the third one more generally simplifies each formula using the remainder of the context. The latter is however computationally more costly, which is why redundant formulas are removed first.

We then prove the following two properties of the function `simp_env`.

Proposition 7. *For any context* Δ*,* *weight*(`simp_env`(Δ)) $\leq$ *weight*(Δ)*.*

Proof. One verifies that each of the rules defining `simp_env` decreases the weight. For (1), this is a property of the sequent calculus. For (2), this is clear. For (3), it follows from the fact that the `choose` functions decrease the weight. □

Proposition 8. *For any context* Δ*, the context* `simp_env`(Δ) *is equivalent to* Δ*.*

4.3 Simplifications specific to Pitts' algorithm

In earlier work [FG23] we gave a direct implementation of Pitts' algorithm [Pit92] for computing propositional quantifiers. We briefly recall the general scheme of this algorithm. The goal is to compute, for any formula φ, formulas $\exists p.\varphi$ and $\forall p.\varphi$ that satisfy the properties of Definition 2. The algorithm more generally computes, for any context Γ and formula φ, a formula $\mathsf{E}_p(\Gamma)$ and a formula $\mathsf{A}_p(\Gamma;\varphi)$; the quantifiers are then defined as $\exists p.\varphi := \mathsf{E}_p([\varphi])$ and $\forall p.\varphi := \mathsf{A}_p([];\varphi)$.

The formulas $\mathsf{E}_p(\Gamma)$ and $\mathsf{A}_p(\Gamma;\varphi)$, in turn, are defined as the conjunction of a set of formulas $\mathcal{E}_\mathsf{p}(\Gamma)$ and the disjunction of a set of formulas $\mathcal{A}_\mathsf{p}(\Gamma;\varphi)$, respectively. We do not recall in detail the definition of the sets of formulas $\mathcal{E}_\mathsf{p}(\Gamma)$ and $\mathcal{A}_\mathsf{p}(\Gamma;\varphi)$ here; see [Pit92, Table 5], or the Coq/Rocq declarations `e_rule` (🐓), `a_rule_env` (🐓), and `a_rule_form` (🐓). For the following discussion, it suffices to know that $\mathcal{A}_\mathsf{p}(\Gamma;\varphi)$ and $\mathcal{E}_\mathsf{p}(\Gamma)$ are built by recursively computing E_p and A_p for the assumptions of any G4iP-rule that may be applied to the input; more precisely, at each stage of the computation, there are 8 different kinds of recursive calls for E and 13 different kinds of recursive calls for A.

The construction outlined above is practical for proving that the output formula is correct with respect to the specification. However, its behaviour is very similar to the naive proof search algorithm, with additional cost incurred by the fact that non-linear rules induce many recursive calls. Our main improvement on the algorithm is to follow a better proof strategy, similar to the one discussed in Section 3: We simplify the output during every recursive call of $\mathcal{A}_p$ or $\mathcal{E}_p$, using the function `simp_env` described above. This will still result in a correct output, essentially because, whenever φ and φ' are equivalent, the specifications of Definition 2 imply that $\exists p.\varphi$ and $\exists p.\varphi'$ are equivalent, and that $\forall p.\varphi$ and $\forall p.\varphi'$ are equivalent; this invariance property lifts to E_p and A_p. In short, we redefine the functions as:

$$\mathsf{A}_p(\Gamma;\varphi) := \bigvee \mathcal{A}_\mathsf{p}(\texttt{simp\_env}(\Gamma);\varphi) \tag{3}$$

$$\text{and} \quad \mathsf{E}_p(\Gamma) := \bigwedge \mathcal{E}_\mathsf{p}(\texttt{simp\_env}(\Gamma)), \tag{4}$$

where $\mathcal{A}_p$ and $\mathcal{E}_p$ are defined in the same way as in Pitts' original algorithm, but use the results of the new functions E_p and A_p in their recursive calls.

Example 9. To give an example of how these simplifications improve the original algorithm, consider the formula

$$\varphi_n := (a_0 \wedge p) \wedge \cdots \wedge (a_n \wedge p)\ .$$

The formula $\exists p.\varphi_n$ is, up to equivalence, $\bigwedge_{i=0}^{n} a_i$, and this is indeed what is computed by our simplified algorithm. However, the naive implementation computes a much larger formula.

In the non-optimized implementation, the function E_p will first find all possible instances of the rule $\wedge$L that have a conclusion of the form $\varphi_n \vdash \cdot$. There are $n+1$ such possible applications, corresponding to the $n+1$ subformulas of the form $(a_i \wedge p)$ in φ_n. This means that E_p will be defined as

$$\bigwedge_{0\le i\le n} \mathsf{E}_p([a_0 \wedge p, \ldots, a_{i-1} \wedge p, a_i, p, a_{i+1} \wedge p, \ldots, a_n \wedge p])$$

Now, for each of the calls to E_p in this conjunction, there are still n possible applications of $\wedge$L. Continuing in this way, there will be approximately $(n+1)!$ recursive calls, all of which result in identical final leaves in every branch, namely, $\mathsf{E}_p(a_0, p, a_1, p, \ldots, a_n, p)$. Without simplifications, this leads to a lot of repeated computation, and a very large output formula.

The main gain obtained by the definitions given in (3) and (4) is that the function `simp_env` applies any possible left invertible linear rules to the context. This means that $\mathcal{E}_p$ and $\mathcal{A}_p$ are only ever applied to a simplified context Γ' that does not contain any formulas of the form $p \to \varphi$, $\varphi \wedge \psi$, $(\varphi \vee \psi) \to \chi$, or $(\varphi \wedge \psi) \to \chi$. As a consequence, four of the

rules from the sequent calculus G4iP can never apply to Γ', reducing the number of different kinds of recursive calls from 8 to 4 for E and from 13 to 9 for A. Since any applicable rule to Γ' incurs a recursive call, this reduction considerably decreases the overall number of redundant recursive calls. We demonstrate this empirically in Section 5.

While one could use these observations to give a shorter correctness proof for the simplified propositional quantifiers, 'from scratch', and with fewer cases, we instead re-use the correctness proofs that we had already formalized previously, with only slight modifications. Whenever the old version of the proof used an induction hypothesis about $\mathcal{A}_p(\Gamma;\varphi)$ or $\mathcal{E}_p(\Gamma)$, in the new proof we are able to use the same induction hypothesis for $\mathcal{A}_p(\texttt{simp\_env}(\Gamma);\varphi)$ or $\mathcal{E}_p(\texttt{simp\_env}(\Gamma))$. To see why this is possible, observe that (i) $\texttt{simp\_env}(\Gamma)$ has lower weight than Γ, so the induction hypothesis used for Γ *a fortiori* applies to $\texttt{simp\_env}(\Gamma)$; and (ii) the context $\texttt{simp\_env}(\Gamma)$ is equivalent to the context Γ, and may therefore be replaced by it in any hypothesis.

A technical modification in the formalization of the proof that we had to make in order for this to go through was a change in the induction principle. Our original implementation used an induction on the last rule in a hypothetical proof of a sequent of the form $\Gamma \vdash \cdot$ (when proving correctness for $\mathsf{E}_p(\Gamma)$) or of a sequent of the form $\Gamma \bullet \cdot \vdash \varphi$ (when proving correctness for $\mathsf{A}_p(\Gamma;\varphi)$). In our new proof, we use an induction on the weight of the sequent.

5 Benchmarks

To better quantify the impact of our simplifications, we developed a set of benchmarks that analyze both the computation time and the reduction in the size of the output formulas (measured in weight) generated by Pitts' construction. These benchmarks allowed us to clearly evaluate the effectiveness of each improvement. Note however that this measure is relatively arbitrary, and that there is no guarantee that we obtain the smallest interpolant with respect to this measure.

The example formulas used for our benchmark were of two kinds: First, we have two sequences of formulas φ_{imp} and φ_{conj} of increasing complexity, which we knew were treated inefficiently by our initial implementation; Second, our initial implementation drew the interest of researchers in intuitionistic logic, notably M. Jibladze (personal communication) and Z. Kocsis [Koc23], who provided us with a number of specific formulas of interest to them, for which the output of our initial implementation was highly complex, but for which one could prove 'by hand' that the quantified formulas could be represented in a simple way. We include some of these examples as the formulas ψ_1, ψ_2, ψ_3 (communicated to us by M. Jibladze) and ψ_4 [Koc23, 3.13] below.

- $\varphi_{\text{imp}}(0) = p, \quad \varphi_{\text{imp}}(n+1) = \varphi_{\text{imp}}(n) \to p_{n+1}$
- $\varphi_{\text{conj}}(0) = p_0 \wedge p, \quad \varphi_{\text{conj}}(n+1) = \varphi_{\text{conj}}(n) \wedge (p_{n+1} \wedge p)$
- $\psi_1 : r \leftrightarrow ((p \to q) \vee ((p \to q) \to q))$
- $\psi_2 : (x \leftrightarrow (p \vee \neg p)) \wedge (y \leftrightarrow (q \vee \neg q)) \wedge \neg(p \wedge q)$
- $\psi_3 : ((q \to (p \vee r)) \to \neg(t \vee p))$
- $\psi_4 : (\neg p \to q) \wedge (\neg\neg p \to q)$

Fig. 4 contains the results of running our program on these inputs. In the optimized version (commit `44d4c8e`), all computations completed in under 1 second, except for the last one, which timed out. This marks a significant improvement over the original execution times (commit `ebb461d`). The improvement in weight is also encouraging, being able to simplify the weight by several orders of magnitude in some cases. This is thanks to the double nature of the simplifications: improving the order in which the rules are applied

Formula	Original weight	Optimized weight
$\forall p, \varphi_{\text{imp}}(6)$	2462	18
$\exists p, \varphi_{\text{imp}}(6)$	131699*	25
$\forall p, \varphi_{\text{imp}}(7)$	263402*	27
$\exists p, \varphi_{\text{imp}}(7)$	–	25
$\exists p, \varphi_{\text{imp}}(8)$	–	25
$\exists p, \varphi_{\text{imp}}(9)$	–	–
$\exists p, \varphi_{\text{conj}}(3)$	42142*	10
$\exists p, \varphi_{\text{conj}}(4)$	–	13
$\forall p, \psi_1$	84	4
$\exists p, \psi_1$	–	5
$\exists p, \psi_2$	–	44
$\exists q, \exists p, \psi_2$	–	8
$\forall p, \psi_3$	429	1
$\exists p, \psi_3$	668	13
$\forall p, \psi_4$	33	1
$\exists p, \psi_4$	160	5

Figure 4. Experimental results for computing propositional quantifiers on the set of formulas defined in Section 5. The 'Original' column shows the initial weights, while the 'Optimized' column presents the weights after applying the optimizations described in this paper. '–' indicates computations that exceeded the 5-minute time limit, and '*' that the computation time took more that 2 minutes.

in Pitts' construction (similar to the decision procedure of Section 3) and simplifying the output during the construction itself, as described in the previous section.

As the example $\exists p, \varphi_{\text{imp}}(9)$ shows, there is still room for improvement. This example, with a high degree of implication nesting on the left, makes for a large uniform interpolant (due to the rule $\to\to L$), for which the simplification procedure does not run in reasonable time.

References

[Dyc92] Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795807, 1992.

[FG23] Hugo Férée and Sam van Gool. Formalizing and computing propositional quantifiers. *Certified proofs and programs (CPP)*, 2023.

[FGGS24] Hugo Férée, Iris van der Giessen, Sam van Gool, and Ian Shillito. Mechanised uniform interpolation for K, GL, and iSL. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Automated Reasoning: 12th International Joint Conference, IJCAR 2024, Nancy, France, July 36, 2024, Proceedings, Part I*. Springer, 2024.

[GZ02] Silvio Ghilardi and Marek Zawadowski. *Sheaves, Games, and Model Completions*. Springer, 2002.

[Hud88] J. Hudelmaier. A Prolog program for intuitionistic logic. Technical report, University of Tübingen, 1988. SNS-Bericht 88-28.

[Hud93] J. Hudelmaier. An $O(n \log n)$-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.

[Koc23] Zoltan A. Kocsis. Proof-theoretic methods in quantifier-free definability, 2023. Preprint arXiv:2310.03640.

[Pit92] A. M. Pitts. On an interpretation of second-order quantification in first-order intuitionistic propositional logic. *J. Symbolic Logic*, 57(1):33–52, 1992.

[SvdGGI23] Ian Shillito, Iris van der Giessen, Rajeev Goré, and Rosalie Iemhoff. A new calculus for intuitionistic strong löb logic: Strong termination and cut-elimination, formalised. In Revantha Ramanayake and Josef Urban, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 73–93, Cham, 2023. Springer Nature Switzerland.

[Vor70] N. N. Vorobev. A new algorithm for derivability in the constructive propositional calculus. *American Mathematical Society Translations*, 94:37–71, 1970. Translated from the 1952 Russian original.