Structural sharing and efficient proof-search in propositional intuitionistic logic

D. Galmiche and D. Larchey-Wendling

LORIA - Université Henri Poincaré
Campus Scientifique, BP 239
Vandœuvre-lès-Nancy, France

Abstract. In this paper, we present a new system for proof-search in propositional intuitionistic logic from which an efficient implementation based on structural sharing is naturally derived. The way to solve the problem of formula duplication is not based on logical solutions but on an appropriate representation of sequents with a direct impact on sharing and therefore on the implementation. Then, the proof-search is based on a finer control of the resources and has a $\mathcal{O}(n \log n)$-space complexity. The system has the subformula property and leads to an algorithm that, for a given sequent, constructs a proof or generates a counter-model.

1 Introduction

In recent years there was a renewed interest in proof-search for constructive logics like intuitionistic logic (IL), mainly because of research in intuitionistic type theories and their relationships with programming through proof-search [7]. Nowadays, theorem provers are more often used into the formal development of software and systems and have to be integrated into various environments and large applications. Then, to provide a simple implementation of a logical calculus (and connected proof-search methods) in imperative programming languages like C or Java, we have to consider new techniques leading to the clarity of the implementation [8] but also to a good and efficient explicit management of formulae, considered here as resources. The potential support for user interfaces or for a natural parallelisation of the proof-search process could be expected.
A recent work on efficient implementation of a theorem prover for Intuitionistic Propositional Logic (IPL) has emphasised the interest of finer control and management of the formulae involved in the proof-search, with an encoding of the sequents at the implementation step [2]. But to solve the problem of duplication of some formulae we need to reconsider the global problem of proof-search in this setting, knowing that a usable proposal is necessary for a good space complexity and an efficient implementation based on resources sharing. A logical solution based on the introduction of new variables and of some adequate strategies is given in [3]. Our goal in this paper is to propose a solution at a structural level (no new formulae and only modifications of the sequents structure) including a finer management of formulae (as resources). Therefore, we define, from an

analysis of previous works, a new logical system, that has some similarities with the one of [3]. With such a system we can naturally derive, from the logical rules and a particular representation of sequents (with specific tree structures), some new rules that act on trees to perform proof-search. Moreover, these ideas can also be applied to a refutability system for IPL to generate counter-models in case of unprovability [5]. With such an approach the depth of the search tree is in both systems linearly bounded and the control on resources leads to an efficient implementation in imperative languages.

2 Proof-search in intuitionistic logic

We focus on IPL for which the LJ sequent calculus is a simple formulation having the subformula property. On the other hand, it lacks termination and thus implementing this calculus involves some kind of loop-checking mechanism to ensure the termination of computation. A solution is the use of the contraction-free sequent calculus LJT [1] that has the nice property that backward search does not loop. There exists other proposals for proof-search in IL based for instance on resolution [7], constraints [9] or skolemization [6].
Let us start to recall the main characteristics of the LJT system. It is is sound and complete w.r.t. provability in LJ and a backward search does not loop and always terminates [1]. For that, the left-implication rule of LJ is replaced by four rules depending on the form of the premiss of the principal formula. Then, the treatment of the implication is becoming finer and the termination is obtained from a well-founded ordering on sequents. Moreover the contraction rule is no longer a primitive rule but is nevertheless admissible. The corresponding multi-conclusion calculus is also a good basis for a non-looping method to construct Kripke trees as counter-models for non-provable formulae of IPL [5]. It provides a calculus, called CRIP, in which refutations are inductively defined and Kripke counter-models can be extracted from these refutations. The rules of LJT are given in figure 1, X representing atomic formulae, A, B, G representing formulae and Γ being a multiset of formulae. This system provides a decision procedure in which the deductions (proofs) may be of exponential depth in the size of their endsequent. Moreover in rules like $\rightarrow$-L_3, some formulae of the conclusion appear twice (duplication problem). We also observe that, for instance in the $\rightarrow$-L_3 rule, the premises are not smaller (in the usual meaning) than the conclusion, even if they are with a multiset ordering (see [1]). The LJT prover, implemented in Prolog, is based on an analytic approach where the size of the formulae decreases but a potential big number of subformulae might appear. One one has no explicit knowledge on the formulae used during the proof-search or on the depth of proofs. Moreover, LJT does not satisfy the subformula property: for instance, in rule $\rightarrow$-L_4, the formula $B \rightarrow C$ is introduced in the left premise even if it is not a subformula of $(A \rightarrow B) \rightarrow C$. Some rules duplicate formulae as in rule $\rightarrow$-L_3 where C is duplicated and in rule $\rightarrow$-L_4 where B is duplicated. Therefore, the idea to share formulae naturally arises so that the duplication would be harmless in terms of space. But we have to know which are the formulae that might appear

$$\frac{}{\Gamma, X \vdash X}\ \mathrm{Id} \qquad \frac{}{\Gamma \vdash \mathsf{T}}\ \mathsf{T} \qquad \frac{}{\Gamma, \mathsf{F} \vdash G}\ \mathsf{F}$$

$$\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ \wedge\text{-L} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-R} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-R}_1$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-R}_2 \qquad \frac{\Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ \vee\text{-L} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{-R}$$

$$\frac{\Gamma, X, B \vdash G}{\Gamma, X, X \to B \vdash G}\ \to\text{-L}_1 \qquad \frac{\Gamma, A \to (B \to C) \vdash G}{\Gamma, (A \wedge B) \to C \vdash G}\ \to\text{-L}_2$$

$$\frac{\Gamma, A \to C, B \to C \vdash G}{\Gamma, (A \vee B) \to C \vdash G}\ \to\text{-L}_3 \qquad \frac{\Gamma, B \to C \vdash A \to B \qquad \Gamma, C \vdash G}{\Gamma, (A \to B) \to C \vdash G}\ \to\text{-L}_4$$

Fig. 1. The LJT calculus

during such backward proof-search. Then we have recently proposed an explicit management of the formulae (considered as resources) and a new proof-search method for LJT based on new concepts and results such as a new notion of subformula [2].

In the LG system, independently proposed in [3], the length of the deductions is linear bounded in terms of the size of the endsequent. A backward application of the rules gives rise to an $\mathcal{O}(n \log n)$-space decision algorithm for IPL. In this work, the strategy used to avoid the duplication of subformulae is different for the $\to$-L$_3$ and $\to$-L$_4$ rules: for the first one, it corresponds to add new propositional variables and to replace complex formulae by such new variables. For the second one, it forces the sequents to keep a convenient form in order to apply some transformations or reductions. In fact, in this system, the duplication problem is solved at a logical level by the introduction of new variables and the use of particular sequents [3]. But it is complicated and several implementation problems arise. Our aim, in this paper, is to propose an alternative (not logical but structural) solution based on a fine management of new structures appropriate for an efficient implementation.

Let us mention the work of Weich [10] that is based on constructive proofs of the decidability of IPL by simultaneously constructing either a proof, taking up the approach of [3] or a counter-model refining the idea of [4]. His algorithm is faster than the refutability algorithm of [5]. When restricted to provability it coincides with the one of [3] including the same logical solution for the duplication problem.

Our proposal is based on a new logical system that has some similarities with the formulations of LG dedicated to the implementation [3]. The duplication problem is in our case treated by a structural approach in which no new variable or formula is introduced and such that we only modify the structure of the sequents. It will lead to a more direct and clear implementation in imperative languages. The main characteristics coming from the use of appropriate sharing techniques are: no dynamic memory allocation, a finer control on the resources and a $\mathcal{O}(n \log n)$-space algorithm for provability. A similar approach can be applied to define and then implement an algorithm for refutability with the generation of counter-models in a same space complexity, i.e., with the depth of search tree being linearly bounded.

$$\frac{\Gamma^\star, A, B \vdash \blacksquare}{\Gamma^\star, A \wedge B \vdash \blacksquare}\ [\wedge_L] \qquad \frac{\Gamma^\star, A \vdash \blacksquare \qquad \Gamma^\star, B \vdash \blacksquare}{\Gamma^\star, A \vee B \vdash \blacksquare}\ [\vee_L]$$

$$\frac{\Gamma^\star, X, C \vdash \blacksquare}{\Gamma^\star, X, X \to C \vdash \blacksquare}\ [X\to] \qquad \frac{}{\Gamma, X, X^\star \to C \vdash \blacksquare}\ [X^\star\to]$$

$$\frac{\Gamma, A, B^\star \to C \vdash \blacksquare \qquad \Gamma^\star, C \vdash \blacksquare}{\Gamma^\star, (A \to B) \to C \vdash \blacksquare}\ [\to\to] \qquad \frac{\Gamma, A, B^\star \to C \vdash \blacksquare}{\Gamma, (A \to B)^\star \to C \vdash \blacksquare}\ [\to^\star\to]$$

$$\frac{\Gamma^\star, A \to (B \to C) \vdash \blacksquare}{\Gamma^\star, (A \wedge B) \to C \vdash \blacksquare}\ [\wedge\to] \qquad \frac{\Gamma, A^\star \to C \vdash \blacksquare \qquad \Gamma, B^\star \to C \vdash \blacksquare}{\Gamma, (A \wedge B)^\star \to C \vdash \blacksquare}\ [\wedge^\star\to]$$

$$\frac{\Gamma^\star, A \to C, B \to C \vdash \blacksquare}{\Gamma^\star, (A \vee B) \to C \vdash \blacksquare}\ [\vee\to] \qquad \frac{\Gamma, A^\star \to C, B \to C \vdash \blacksquare}{\Gamma, (A \vee B)^\star \to C \vdash \blacksquare}\ [\vee_l^\star\to]$$

$$\frac{}{\Gamma, X \vdash X}\ [\mathrm{Ax}] \qquad \frac{\Gamma, A \to C, B^\star \to C \vdash \blacksquare}{\Gamma, (A \vee B)^\star \to C \vdash \blacksquare}\ [\vee_r^\star\to]$$

$$\frac{\Gamma, A, B \vdash G}{\Gamma, A \wedge B \vdash G}\ [\wedge_L] \qquad \frac{\Gamma, A \vdash G \qquad \Gamma, B \vdash G}{\Gamma, A \vee B \vdash G}\ [\vee_L]$$

$$\frac{\Gamma, X, C \vdash G}{\Gamma, X, X \to C \vdash G}\ [X\to] \qquad \frac{\Gamma, A, B^\star \to C \vdash \blacksquare \qquad \Gamma, C \vdash G}{\Gamma, (A \to B) \to C \vdash G}\ [\to\to]$$

$$\frac{\Gamma, A \to (B \to C) \vdash G}{\Gamma, (A \wedge B) \to C \vdash G}\ [\wedge\to] \qquad \frac{\Gamma, A \to C, B \to C \vdash G}{\Gamma, (A \vee B) \to C \vdash G}\ [\vee\to]$$

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ [\wedge_R] \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ [\vee_{R_l}]$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ [\to_R] \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ [\vee_{R_r}]$$

Fig. 2. The SLJ-system

3 A new system for intuitionistic proof-search

We present in this section a new logical system, called SLJ, that leads to a natural use of sharing techniques on structures adapted to an efficient implementation in imperative languages. It is derived from the ideas of LJT [1] and of LG [3].

3.1 The SLJ system

The SLJ-system, that is given in figure 2, includes two kinds of sequents, like in [3]. We have the usual intuitionistic sequent $\Gamma \vdash A$ which is a multiset Γ of formulae forming the hypothesis together with a unique formula A called the conclusion. The intuitive meaning of a sequent is that the conclusion A logically follows from the hypothesis in Γ. Moreover, we have a second kind of sequent which we will call *boxed sequent*: $\Gamma, \alpha^\star \to C \vdash \blacksquare$ which intuitively means $\Gamma, \alpha \to C \vdash \alpha$. The special $\blacksquare$-notation indicates that the conclusion α is shared with the left side of an implication in the hypothesis: $\alpha \to C$. For example, in rule $[\to\to]$, B is shared and not duplicated. There is a mark $\star$ to determine which formula is concerned. In this second kind of sequents, there is exactly one marked formula on the left of an implication of the hypothesis. We will denote $\Gamma^\star$ to point out the fact that Γ contains exactly one formula of type $\alpha^\star \to C$. This will happen when this last formula is not the active one like in

rule $[X\to]$ for example. This idea of marking α was introduced in [3] to avoid one particular kind of duplication of formula in LJT and is also useful for our purpose. It is important to notice that when we try to prove a boxed sequent $\Gamma, \alpha^\star \to C \vdash \blacksquare$, the sequents encountered upper in the proof are always boxed: only boxed sequents appear in the proof of an initial boxed sequent.
The reader may also notice that the rules $[\wedge_L]$, $[\vee_L]$, $[X\to]$, $[\to\to]$, $[\wedge\to]$ and $[\vee\to]$ are duplicated. There is one version for boxed sequents and another version for standard sequents. In the case of boxed sequents, the marked (or starred) formula is not active. We have chosen not to distinguish the names because only the nature of the sequent (boxed or not) differs for each pair of rules. On the contrary, if we compare the rules $[\to\to]$ and $[\to^\star\to]$, then it appears that they are very different. In the case the marked formula is active, the rules do not have the same meaning at all.
This system is very interesting because all the duplications appearing in LJT have been removed except in the rules $[\vee\to]$, $[\vee_l^\star\to]$ and $[\vee_r^\star\to]$. The problem of duplication in the rule ($\to$-L$_4$) of LJT has then disappeared. We will see in section 4 that the last duplication cases can be addressed by changing the structure of sequent to represent sharing.

3.2 Kripke models

Kripke models are a very general model representation for logics like intuitionistic or modal logics. Here, we will adopt the same notation as in [10] for Kripke trees. A *Kripke tree* is a pair $\mathcal{K} = (\mathcal{S}_\mathcal{K}, [\mathcal{K}_1, \ldots, \mathcal{K}_p])$ where $\mathcal{S}_\mathcal{K}$ is a finite set of logical variables and $[\mathcal{K}_1, \ldots, \mathcal{K}_p]$ is a finite list of Kripke trees. Moreover, we suppose that for each i, $\mathcal{S}_\mathcal{K} \subseteq \mathcal{S}_{\mathcal{K}_i}$. This monotony condition is typical to intuitionistic logic as opposed to modal logics for example. This is an inductive definition of Kripke trees and the base case is when the list $[\mathcal{K}_1, \ldots, \mathcal{K}_p]$ is empty, i.e. $p = 0$. In fact, Kripke trees are regular (oriented) trees with each node tagged with a set of variables that grows along the paths from the root to the leaves.
We then introduce the notion of *forcing*. Let $\mathcal{K} = (\mathcal{S}_\mathcal{K}, [\mathcal{K}_1, \ldots, \mathcal{K}_p])$ be a Kripke tree and F be a logical formula. We say that $\mathcal{K}$ forces F and write $\mathcal{K} \vDash F$. The inductive definition is the following:

$$\mathcal{K} \vDash X \text{ iff } X \in \mathcal{S}_\mathcal{K} \qquad \mathcal{K} \vDash A \to B \text{ iff } \mathcal{K} \vDash A \text{ implies } \mathcal{K} \vDash B \text{ and } \forall i, \mathcal{K}_i \vDash A \to B$$
$$\mathcal{K} \vDash A \wedge B \text{ iff } \mathcal{K} \vDash A \text{ and } \mathcal{K} \vDash B \qquad \mathcal{K} \vDash A \vee B \text{ iff } \mathcal{K} \vDash A \text{ or } \mathcal{K} \vDash B$$

The only difference with classical logic here is that the forcing of logical implication $A \to B$ is inherited in the sons $\mathcal{K}_i$ of $\mathcal{K}$ and so in all its subtrees. In fact, this inheritance is extended to all formulae with the following result, a proof of which can be found in [10].

Lemma 1. *If F is a logical formula and $\mathcal{K} = (\mathcal{S}_\mathcal{K}, [\mathcal{K}_1, \ldots, \mathcal{K}_p])$ is a Kripke tree forcing F, i.e. $\mathcal{K} \vDash F$, then for all i, $\mathcal{K}_i \vDash F$.*

Kripke trees are models of intuitionistic logic. Let $A_1, \ldots, A_n \vdash B$ be a sequent, we say that a Kripke tree $\mathcal{K}$ is a model of $A_1, \ldots, A_n \vdash B$ if $\mathcal{K} \vDash A_1$ and $\ldots$ and $\mathcal{K} \vDash A_n$ implies $\mathcal{K} \vDash B$. We will often write $\mathcal{K} \vDash A_1, \ldots, A_n \vdash B$.

Theorem 2 (Soundness). *If $\Gamma \vdash A$ is provable in intuitionistic logic and $\mathcal{K}$ is a Kripke tree then $\mathcal{K} \vDash \Gamma \vdash A$.*

Models are very often used in a negative way to provide a synthetic argument to the unprovability of some formula or sequent. Therefore, we say that $\mathcal{K}$ is a counter-model of the sequent $A_1, \ldots, A_n \vdash B$ if $\mathcal{K} \vDash A_1$ and ... and $\mathcal{K} \vDash A_n$ and $\mathcal{K} \nvDash B$. We write $\mathcal{K} \nvDash A_1, \ldots, A_n \vdash B$. There is an constructive version of the completeness theorem, see [10] for example.

Theorem 3 (Completeness). *Let $\Gamma \vdash A$ be a sequent, then either it is intuitionistically provable, or there exists a Kripke tree $\mathcal{K}$ such that $\mathcal{K} \nvDash \Gamma \vdash A$.*

3.3 Completeness of SLJ

Before proving the theorem we have to precise how Kripke trees can model boxed sequents. $\mathcal{K}$ is a counter-model to $\Gamma, \alpha^\star \to C \vdash \blacksquare$ if $\mathcal{K}$ is a counter-model to $\Gamma, \alpha \to C \vdash \alpha$, so $\mathcal{K} \vDash \Gamma, \alpha \to C$ and $\mathcal{K} \nvDash \alpha$. But we do not know whether $\mathcal{K}$ forces C or not at the level of $\mathcal{K}$. However, if one of its sons $\mathcal{K}_i$ forces α then $\mathcal{K}_i$ also forces C. The completeness result for SLJ can be decomposed into two parts: one for boxed sequents and one for standard sequents. Here we only prove the part for boxed sequents. The other part is very similar. The reader is reminded that the following proofs are inspired from [5,10].

Theorem 4 (Soundness). *If $\Gamma, \alpha^\star \to C \vdash \blacksquare$ is a provable sequent in SLJ, then $\Gamma, \alpha \to C \vdash \alpha$ is intuitionistically provable (or provable in LJT).*

To obtain this result, it is sufficient to show that all the boxed rules[1] of SLJ are admissible. For example, let us explore the case of $[\wedge^\star\to]$. We have to prove that the following rule is admissible in IPL:

$$\frac{\Gamma, A \to C \vdash A \qquad \Gamma, B \to C \vdash B}{\Gamma, (A \wedge B) \to C \vdash A \wedge B}$$

From the premisses, we obtain $\Gamma \vdash (A \to C) \to A$ and $\Gamma \vdash (B \to C) \to B$. The sequent $(A \to C) \to A, (B \to C) \to B, (A \wedge B) \to C \vdash A \wedge B$ is also provable. By two applications of the (Cut) rule, we obtain $\Gamma, \Gamma, (A \wedge B) \to C \vdash A \wedge B$. We finally get the conclusion by application of the contraction rule. More details can be found in [3].

Theorem 5 (Completeness). *Let $\Gamma, \alpha^\star \to C \vdash \blacksquare$ be a boxed sequent, either it admits a proof in SLJ, or it admits a Kripke tree counter-model.*

The proof is done by induction on $\Gamma, \alpha \to C$, but it is not a simple induction on the size of $\Gamma, \alpha \to C$, see [1]. First, we point out the fact that some rules have the important invertibility property: any counter-model to one of the premisses is a counter-model to the conclusion. This is the case for the $[\wedge_L]$, $[\vee_L]$, $[X\to]$,

[1] Boxed rules are the rules of figure 2 for which the conclusion is a boxed sequent.

$[X^\star\!\to]$, $[\to^\star\!\to]$, $[\wedge\!\to]$, $[\wedge^\star\!\to]$ and $[\vee\!\to]$ rules. Then if Γ has one of the following form,

$$\begin{array}{lll} \Gamma \equiv \Delta, A \wedge B & \Gamma \equiv \Delta, A \vee B & \Gamma \equiv \Delta, X, X \to C \\ \Gamma \equiv \Delta, (A \vee B) \to C & \Gamma \equiv \Delta, (A \wedge B) \to C & \end{array}$$

one of the above rule can be applied. The induction hypothesis gives us either one proof for each premiss or else one counter-model to one of the premisses, and thus we either have a proof or a counter model of $\Gamma, \alpha^\star \to C \vdash \blacksquare$. This is also the case when $\alpha \equiv A \to B$, $\alpha \equiv A \wedge B$ or $\alpha \equiv X$ and $X \in \Gamma$. Then the problem is reduced to the following case:

$$\Gamma \equiv \begin{cases} X_1, \ldots, X_n, Y_1 \to D_1, \ldots, Y_k \to D_k, \\ (A_1 \to B_1) \to C_1, \ldots, (A_p \to B_p) \to C_p \end{cases}$$

with $\{X_1, \ldots, X_n\} \cap \{Y_1, \ldots, Y_k\} = \emptyset$. Moreover, either $\alpha \equiv A \vee B$, or $\alpha \equiv X$ and $X \notin \{X_1, \ldots, X_n, Y_1, \ldots, Y_k\}$. We will suppose that $\alpha \equiv A \vee B$ for the rest of the proof — the other case is simpler. We introduce $\Delta_i = \Gamma - \big\{(A_i \to B_i) \to C_i\big\}$. Then, by induction hypothesis, either one of the $\Delta_i, A_i, B_i^\star \to C_i, \alpha \to C \vdash \blacksquare$ is provable, or there exists a counter-model for each of them.
In the first case, suppose that $\Delta_{i_0}, A_{i_0}, B_{i_0}^\star \to C_{i_0}, \alpha \to C \vdash \blacksquare$ has a proof in SLJ. Then consider the rule

$$\frac{\Delta_{i_0}, A_{i_0}, B_{i_0}^\star \to C_{i_0}, \alpha \to C \vdash \blacksquare \qquad \Delta_{i_0}, C_{i_0}, \alpha^\star \to C \vdash \blacksquare}{\Gamma, \alpha^\star \to C \vdash \blacksquare}\ [\to\to]$$

This rule is right invertible: a counter-model of the second premiss is a counter-model of the conclusion. On the other hand, a proof of the second premiss would give a proof of $\Gamma, \alpha^\star \to C \vdash \blacksquare$. Thus, we obtain a proof or a counter-model of the conclusion.
In the second case, for each $i \in [1,p]$ let $\mathcal{K}_i$ be a counter-model of $\Delta_i, A_i, B_i^\star \to C_i, \alpha \to C \vdash \blacksquare$. Let us consider $\alpha \equiv A \vee B$.[2] By induction hypothesis, either one sequent among $\Gamma, A^\star \to C, B \to C \vdash \blacksquare$ and $\Gamma, A \to C, B^\star \to C \vdash \blacksquare$ is provable, or we have two counter-models $\mathcal{K}_A$ and $\mathcal{K}_B$. In the first subcase, we obtain a proof of $\Gamma, (A \vee B)^\star \to C \vdash \blacksquare$ applying $[\vee_l^\star\!\to]$ or $[\vee_r^\star\!\to]$. In the second subcase, we define $\mathcal{K} = \big(\{X_1, \ldots, X_n\}, [\mathcal{K}_1, \ldots, \mathcal{K}_p, \mathcal{K}_A, \mathcal{K}_B]\big)$. It is only routine to check that $\mathcal{K}$ is a counter-model to $\Gamma, \alpha^\star \to C \vdash \blacksquare$. We now give some details of the proof. As $X_j \in \Delta_i$, we get $\mathcal{K}_i \vDash X_j$ and so $X_j \in \mathcal{S}_{\mathcal{K}_i}$ for any $j \in [1,n]$ and $i \in \{1, \ldots, p, A, B\}$. Thus, $\mathcal{K}$ is a Kripke tree and it is simple to check that $\mathcal{K} \vDash X_1, \ldots, X_n, Y_1 \to D_1, \ldots, Y_k \to D_k, \alpha \to C$.
As $\mathcal{K}_A \nvDash A$ and $\mathcal{K}_B \nvDash B$, we obtain $\mathcal{K} \nvDash \alpha$ and then it remains to prove that $\mathcal{K} \vDash (A_j \to B_j) \to C_j$ hold. For any $i \in \{1, \ldots, p, A, B\} - \{j\}$, $(A_j \to B_j) \to C_j$ is in the context (Δ_i or Γ) and so $\mathcal{K}_i \vDash (A_j \to B_j) \to C_j$. Why does $\mathcal{K}_j \vDash (A_j \to B_j) \to C_j$ hold? Because we have $\mathcal{K}_j \vDash A_j$ and $\mathcal{K}_j \vDash B_j \to C_j$. Moreover $\mathcal{K}_j \nvDash B_j$ and also $\mathcal{K}_j \nvDash A_j \to B_j$ and thus $\mathcal{K} \nvDash A_j \to B_j$. Finally, if $\mathcal{K} \vDash A_j \to B_j$ then $\mathcal{K} \vDash C_j$.

[2] Remember we have supposed this but a complete proof would also consider the case $\alpha \equiv X$ and $X \notin \{X_1, \ldots, X_n, Y_1, \ldots, Y_k\}$.

This completeness proof for the boxed fragment of SLJ also describes an algorithm that produces a proof or a counter-model out of a given (boxed) sequent. It can be effectively used to build counter-models.

4 Sharing techniques

In this section, we will present an alternative approach, w.r.t. [3], to address the problem of duplication of formulae during proof-search. It is not logical but structural: only the structure of sequents is modified, no new formula or variable is introduced. Looking at the SLJ-system, we can observe that a duplication occurs when the active formula is of type $(A \vee B) \to C$, and only in this case, as for example in the rule $[\vee\!\to]$. Hudelmaier proposed a way to bypass this problem for complexity reasons: he wanted the linear size of sequent to decrease strictly while applying rules of his system and put a restriction on C that is to be an atomic formula, a variable for example. But doing this, he had to introduce a new rule in the system so that any formula C could be replaced by a variable without modifying the meaning of the sequent:

$$\frac{\Gamma, A \to X, X \to C \vdash G}{\Gamma, A \to C \vdash G} \; [X \text{ new variable}]$$

By this trick, the system assures that any duplication is done on a formula of size 1 and then has the property that the application of rules decreases the linear size of the sequent. That is why we say that the duplication problem is addressed on the logical side. We can see that the variable X represents the sharing of C between $A \to C$ and $B \to C$ by combining the two rules:

$$\frac{\dfrac{\Gamma, A \to X, B \to X, X \to C \vdash G}{\Gamma, (A \vee B) \to X, X \to C \vdash G} \; [\vee\!\to]}{\Gamma, (A \vee B) \to C \vdash G} \; [X \text{ new variable}]$$

But this method has the drawback of introducing some new formulae ($A \to X$ for example) during the proof-search. This could be a problem when we look for a resource-conscious implementation of proof-search. It may not be easy to allocate in advance all the formulae that could appear during the proof-search process. We propose an alternative approach that consists in encoding the sharing of C between $A \to C$ and $B \to C$ inside the sequent structure.

4.1 A structural solution

Usually, an intuitionistic sequent is a multiset (or set or list, depending on structural rules) of formulae, as hypothesis, together with a unique formula for the conclusion. Here, we will consider a new structure encoding the sharing.

Sequent structure. The left part of sequents will have the structure of a forest (i.e. list of trees). Each tree represents a compound formula where sharing is taken into account. For this, the nodes of the trees are associated to subformulae of the initial sequent which will be called *tags*. Let l be a leaf in this tree, we denote by $F_0, \ldots, F_l$ the list of formulae encountered while following the path from the root of tag F_0 to the leaf l tagged with F_l.

Each tree $\mathcal{T}$ is associated a *meaning*, which is the logical formula $\bigwedge_l F_l \to \cdots \to F_0$ where l ranges over the leaves of $\mathcal{T}$. As usual, $\to$ is right associative and $F_l \to \cdots \to F_0$ means $F_l \to (\cdots \to F_0)$. See the example in figure 3. There is an obvious map from formulae to trees $F \longmapsto \bullet F$ that associates a formula F to one-node tree whose root is tagged with F.

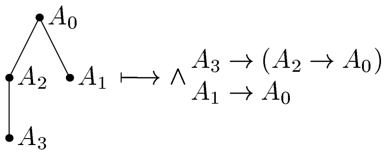

Fig. 3. Meaning function

Rules. The rules are derived from the SLJ rules and the preceding meaning maps. They act on the leaves of the trees and there are two cases: the tree is a one-node tree or else the root is not a leaf.

In the case of a one-node tree, either the formula is an implication in which case the node is rewritten as a two-nodes tree (rule $[\to_L]$ in figure 5) or we apply left conjunction or disjunction rule to obtain one or two one-node trees.

In the case the root is not a leaf, a formula $F_l \to (F_{l-1} \to \cdots \to F_0)$ corresponds to the leaf l and thus to an left implication rule of SLJ. For example, if F_l is $A \wedge B$ the leaf l is replaced by a two-nodes tree whose leaf is tagged with A and whose root is tagged with B like it appears in rule $[\wedge\!\to]$ of figure 5.

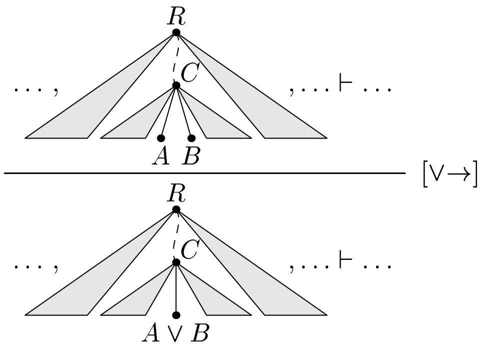

Fig. 4. Tree decomposition rule

We focus here on one of the rules that involves duplication. Considering the trees of figure 4, we see that the tree of the conclusion is associated to a formula of the shape $\cdots \wedge \big[(A \vee B) \to (C \to \cdots \to R)\big] \wedge \cdots$. Considering the rule $[\vee\!\to]$ of SLJ, and the fact that $\wedge$ can be exchanged with commas (,) on the left side of SLJ-sequents because of the rule $[\wedge_L]$, we can convert this formula into $\cdots \wedge \big[A \to (C \to \cdots \to R) \wedge B \to (C \to \cdots \to R)\big] \wedge \cdots$. The new formula exactly corresponds to the tree of the premise. It illustrates the method from which we derive the tree manipulation rules from SLJ-rules. We also notice how the duplication has been removed with the help of sharing. The node tagged with C is shared in the two paths leading to A and B. Finally, A and B which replace $A \vee B$ are subformulae. This fact is a general one, and all rules introduce only subformulae so that the tags of the trees are all subformulae of the formulae of the initial sequent. The it provides a subformula property (see section 4.2).

It is important to point out that the sharing version of SLJ is not a simple implementation in terms of trees. In particular, we have previously seen that proving a boxed sequent only involves boxed sequents. This is no longer the case

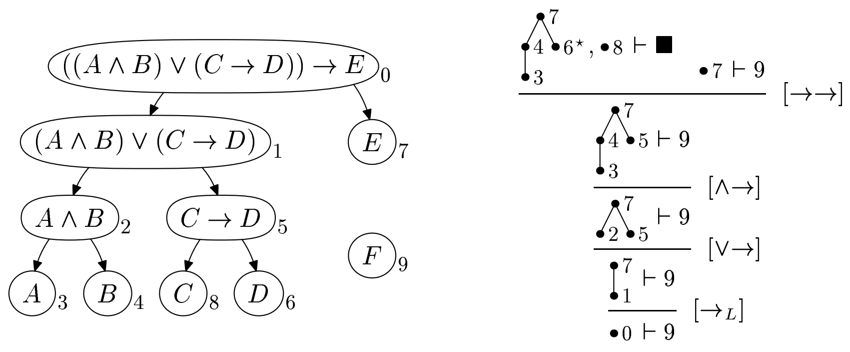

Fig. 5. An example of proof-search

in the sharing tree version. The tree version of the rule $[X\to]$ might remove a whole subtree and this one might contain the marked (or starred) formula. In this case, the premiss cannot be a boxed sequent and we must reintroduce the marked formula as the conclusion of the premiss.

An example. Considering figure 5, we aim to prove the sequent $((A \wedge B) \vee (C \to D)) \to E \vdash F$. First we build the tree of the subformulae of this sequent. The nodes are given with numbers used to label the nodes of proof trees. The proof-search tree is written bottom-up as usual. We start by translating the premiss into a forest (in this case, there is only one tree) and then we develop this forest following the rules for trees. Notice that only the leaves of the trees are concerned in those rules. Inner nodes are "asleep" until all the sons are "removed." This point has some consequences when efficient implementation is taken into account (see section 5). Moreover a marked formula $6^\star$ appears in the tree at the end of the proof-search. As for SLJ, there might be at most one marked formula in the forest, meaning that this formula is shared with the conclusion and in this case, the conclusion is boxed.

4.2 Properties of structural sharing

Structural sharing avoids the duplication of formulae. In fact, anytime a formula on a leaf is decomposed, it is always replaced by one or two leaves that are tagged with the subformulae of the tagging formula. Sometimes a part of the tree can be removed. Let us consider some results about complexity, subformula property and counter-models. If we measure the size of the forest by the sum of the size of each tagging formula, then the size of any premise of a rule is strictly smaller than the size of the conclusion. Moreover, the size of the initial forest is exactly the size of initial sequent in the usual sense.

Theorem 6. *The depth of the proof-search tree is bounded by the size of the sequent (or forest) to prove.*

This result is important and already obtained in [3], providing an $\mathcal{O}(n \log n)$-space decision procedure for intuitionistic logic. Here we focus on an efficient implementation of such a procedure, based on structural sharing. Another important point is that a tagging formula is always replaced by some of its direct subformulae.

Theorem 7. *The tagging formulae occurring during the proof-search, as the forest gets decomposed, are subformulae of the initial sequent (or forest).*

The consequence for implementation is significant. In fact, an occurrence of a formula A is always decomposed by the same rule, which ever the context. This only depends on the *polarity*[3] of A as a subformula. It allows to do the allocation of space for subformulae before the proof-search starts, and not dynamically.
In section 3.3, we sketched an algorithm to effectively build a counter-model when the proof-search fails, like what was already done in [10]. The proof can be very easily adapted to the sharing-trees version of the system. In fact, one can effectively build Kripke-trees that are *strong counter models* (see [10]) in case of the failure of the proof-search algorithm on sharing trees.

Theorem 8. *The algorithm can produce proofs or counter-models and their depth is bounded by the size of the sequent to prove.*

5 Implementation issues

The sharing tree version of the SLJ system is well suited for an efficient implementation in imperative languages. A proof-search algorithm requires a bounded memory space (depending on the size of the initial sequent) and it can be implemented without using dynamic memory allocation at all. Moreover, good choices are not obvious, especially in terms of time, if we aim to have a light use of resources. To define a good proof-search strategy is not an easy task and can depend on the kind of formulae. Its implementation could be difficult because it could involve too much administrative tasks like maintaining lists of formulae or even traversing lists at each point in the proof-search. A strategy has to take into account that some rules have the invertibility property. Moreover, in case the algorithm has to return counter-models, the strategies are restricted to those respecting the order in the completeness proof of section 3.3.
For a given strategy, the program having a sequent (set of sharing trees) as input has to choose a leaf of the tree. Going through the tree to search for a particular kind of leaf is not a good solution and it is necessary to gather leaves by types. But this information has to be maintained when the proof rules are applied. To summarise, the basic resources are the sequent represented as a tagged tree of formulae and the conclusion (or starred formula if we have a boxed sequent). This information is completed with some administrative information so as to make these resources accessible in minimal time. The key-points are the following: the

[3] This is the parity of the number of times a subformula occurs on the left of $\rightarrow$.

leaves should be organised by types to have an efficient choice. It is especially important to avoid the traversal of leaves at each point of the proof-search; the application of rules should not involve too much administrative tasks otherwise the benefits of the added information would be null; the reverse application of rules is also concerned by this point because the non-invertibility of some rules implies some backtracking.

6 Conclusion

In this paper, we have presented a new system SLJ derived from ideas of LJT [1] and LG [3] which is proved sound and complete for IPL. It is an intermediate step towards an efficient implementation of IPL based on the concept of sharing trees. This system has very nice properties like the subformula property, a linear bound on the depth of proof-search and the ability to generate Kripke counter-models. In addition to the theoretical complexity results, our approach provides an effective and fine control of space and time resources, adapted to the design in imperative languages. A first implementation has been developed in C and the practical feasibility of the method has been confirmed. A possible extension of this work would be, as in [10], implementing the intuitionistic proof of the system based on sharing trees and mechanically extracting a (proved) algorithm from this proof.

References

1. R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57:795–807, 1992.
2. D. Galmiche and D. Larchey-Wendling. Formulae-as-resources management for an intuitionistic theorem prover. In *5th Workshop on Logic, Language, Information and Computation, WoLLIC'98*, Sao Paulo, Brazil, July 1998.
3. J. Hudelmaier. An $\mathcal{O}(n \log n)$-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.
4. M. Ornaghi P. Miglioli, U. Moscato. Avoiding duplications in tableaux systems for intuitionistic and Kuroda logic. *Logic Journal of the IGPL*, 5(1):145–167, 1997.
5. L. Pinto and R. Dyckhoff. Loop-free construction of counter-models for intuitionistic propositional logic. In Behara and al., editors, *Symposia Gaussiana*, pages 225–232, 1995.
6. N. Shankar. Proof search in the intuitionistic sequent calculus. In *11th Conference on Automated DEduction, LNAI 607*, pages 522–536, Saratoga Springs, June 1992.
7. T. Tammet. A resolution theorem prover for intuitionistic logic. In *13th Conference on Automated Deduction, LNAI 1104*, pages 2–16, NJ, USA, 1996.
8. C. Urban. Implementation of proof search in the imperative programming language pizza. In *Int. Conference TABLEAUX'98, LNAI 1397*, pages 313–319, Oisterwijk, The Netherlands, May 1998.
9. A. Voronkov. Proof-search in intuitionistic logic based on constraint satisfaction. In *Int. Workshop TABLEAUX'96, LNAI 1071*, pages 312–327, Italy, 1996.
10. K. Weich. Decisions procedures for intuitionistic logic by program extraction. In *Int. Conference TABLEAUX'98, LNAI 1397*, pages 292–306, Oisterwijk, The Netherlands, May 1998.