Optimization Techniques For Propositional Intuitionistic Logic And Their Implementation

Alessandro Avellone [a] Guido Fiorino [a,*] Ugo Moscato [a]

[a] *Dipartimento di Metodi Quantitativi per le Scienze Economiche ed Aziendali, Università di Milano-Bicocca, Piazza dell'Ateneo Nuovo, 1, 20126 Milano, Italy,*

Abstract

This paper presents some techniques which bound the proof search space in propositional intuitionistic logic. These techniques are justified by Kripke semantics and are the backbone of a tableau based theorem prover (PITP) implemented in C++. PITP and some known theorem provers are compared using the formulas of ILTP benchmark library. It turns out that PITP is, at the moment, the propositional prover that solves most formulas of the library.

Key words: Tableaux, intuitionistic logic, automated theorem proving

1 Introduction

The development of effective theorem provers for intuitionistic and constructive logics is of interest both for the investigation and application of such logics to formal software/hardware verification and to program synthesis (see i.e. [19,8,12,21,13,6,1]).

In this paper we present a proof strategy and an implementation based on a tableau calculus for propositional intuitionistic logic. Our decision procedure implements the tableau calculus of [2] (this calculus is an enhancement of the calculus given in [15] and is related to the tableau and sequent calculi of [25,18,11]).

* Corresponding author.
Email addresses: alessandro.avellone@unimib.it (Alessandro Avellone), guido.fiorino@unimib.it (Guido Fiorino), ugo.moscato@unimib.it (Ugo Moscato).

We introduce some new techniques utilized by our decision procedure to narrow the search space and the width of the proofs (some of these techniques have already been described in our previous work [4]). The PSPACE-completeness of intuitionistic validity [30] suggests that backtracking and branching cannot be eliminated. In order to improve the time efficiency of the implementations and make them usable, strategies have to be developed to bound backtracking and branching as much as possible.

The optimizations we present are justified by the Kripke semantics for intuitionistic logic. Such semantical techniques are related to the fact that tableau calculi are strictly joined to the semantics of the logic at hand. Loosely speaking, a tableau proof for a formula is the attempt to build a model satisfying the formula. The construction of such a model proceeds by increasing, step by step, the information necessary to define such a model (thus, step by step the accuracy of the model increases). If the proof ends in a contradiction, then there is no model for the formula. Otherwise, a model satisfying the formula is immediately derived from the proof. With this machinery at hand, we first adapt Simplification, a technique introduced in [20] for classical and modal logics, to intuitionistic logic. Then we present a technique to deduce the satisfiability of a set of formulas S, when the satisfiability of a set S' and a substitution τ such that $S = \tau(S')$ are known (where with $\tau(S')$ we mean the application of τ to S' in the obvious way). Such a technique allows us to bound backtracking. Next we discuss a technique suitable to bound branching on the formulas which only contain conjunctions and disjunctions. Such a technique is an adaptation of *regularity*, a technique well known to people engaged in classical theorem proving (see [17] for further details). Finally we discuss a criterion which allows us to establish that a set of formulas is satisfiable. In this way it is not necessary to search for a proof of contradiction.

Besides the strategy and its completeness, in the final part of the paper we present some experimental results on the implementation of PITP (we mention that a system description of PITP can be found in [5], where no account of the theoretical results related to the optimizations is given). PITP is written in C++ and it is tested on the propositional part of ILTP v1.1.2 benchmark library [28]. Of 274 propositional benchmarks contained in ILTP v1.1.2, PITP decides 234 formulas within the time limit of ten minutes (we point out that this version of PITP outperforms the previous one described in [4], where, since Simplification was not implemented, only 215 formulas were decided). To give the reader more elements to evaluate PITP optimizations, comparisons with different versions of PITP are provided.

2 Notation and Preliminaries

We consider the propositional language $\mathcal{L}$ based on a denumerable set of propositional variables $\mathcal{PV}$, the boolean constants $\top$ and $\bot$ and the logical connectives $\neg, \wedge, \vee, \rightarrow$. An atom is either a propositional variable or a boolean constant. (Propositional) Kripke models are the main tool to semantically characterize (propositional) intuitionistic logic **Int** (see [7] and [15] for the details). A Kripke model for $\mathcal{L}$ is a structure $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$, where P is a set of worlds, $\langle P, \leq, \rho \rangle$ is a poset with minimum ρ and $\Vdash$ is the *forcing relation*, a binary relation on $P \times (\mathcal{PV} \cup \{\top, \bot\})$ such that: (i) if $\alpha \Vdash p$ and $\alpha \leq \beta$, then $\beta \Vdash p$; (ii) for every $\alpha \in P$, $\alpha \Vdash \top$ holds and $\alpha \Vdash \bot$ does not hold. The forcing relation is extended in a standard way to arbitrary formulas of $\mathcal{L}$ as follows:

(1) $\alpha \Vdash A \wedge B$ iff $\alpha \Vdash A$ and $\alpha \Vdash B$;
(2) $\alpha \Vdash A \vee B$ iff $\alpha \Vdash A$ or $\alpha \Vdash B$;
(3) $\alpha \Vdash A \rightarrow B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \Vdash A$ implies $\beta \Vdash B$;
(4) $\alpha \Vdash \neg A$ iff for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \Vdash A$ does not hold.

We write $\alpha \nVdash A$ when $\alpha \Vdash A$ does not hold. It is easy to prove that for every formula A, if $\alpha \Vdash A$ and $\alpha \leq \beta$, then $\beta \Vdash A$. A formula A *is valid in a Kripke model* $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ if and only if $\rho \Vdash A$. It is well-known that **Int** coincides with the set of formulas valid in all Kripke models.

If we consider Kripke models $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $|P| = 1$ we get a classical interpretation for (propositional) classical logic **Cl**. Classical interpretations are usually seen as functions σ from $\mathcal{PV}$ to $\{true, false\}$. Given a formula A and a (classical) interpretation σ, we use $\sigma \models A$ with the usual meaning of satisfiability of a formula in a classical interpretation. Given a set S, the set $\mathcal{PV}(S)$ denotes the elements of $\mathcal{PV}$ occurring in S.

In the following presentation, we give a brief overview of the tableau calculus **Tab** of [2] which is implemented by our decision procedure. The rules of the calculus are given in Fig. 1. The calculus operates on signed well-formed formulas (swff for short), where a swff is a (propositional) formula prefixed with a sign **T**, **F** or $\mathbf{F_c}$.

The sign $\mathbf{F_c}$ was first introduced in [27] and later published in [22,23]. This sign, which intuitively represents *certain falsehood*, partially solves a problem mentioned in [15], where the tableau calculus for intuitionistic propositional logic contains an implicit rule, which allows the duplication of formulas following the application of a rule. Without this implicit rule, the calculus in [15] would not be complete; technically, the calculus uses multisets S of signed formulas, where the same signed formula $\mathcal{S}A$ can be repeated in S disregarding the order of elements of S. The rules in relation to sign $\mathbf{F_c}$ guarantee that rule

$$\frac{S, \mathbf{T}(A \wedge B)}{S, \mathbf{T}A, \mathbf{T}B}\mathbf{T}\wedge \quad \frac{S, \mathbf{F}(A \wedge B)}{S, \mathbf{F}A | S, \mathbf{F}B}\mathbf{F}\wedge \quad \frac{S, \mathbf{F_c}(A \wedge B)}{S_c, \mathbf{F_c}A | S_c, \mathbf{F_c}B}\mathbf{F_c}\wedge$$

$$\frac{S, \mathbf{T}(A \vee B)}{S, \mathbf{T}A | S, \mathbf{T}B}\mathbf{T}\vee \quad \frac{S, \mathbf{F}(A \vee B)}{S, \mathbf{F}A, \mathbf{F}B}\mathbf{F}\vee \quad \frac{S, \mathbf{F_c}(A \vee B)}{S, \mathbf{F_c}A, \mathbf{F_c}B}\mathbf{F_c}\vee$$

$$\frac{S, \mathbf{T}q, \mathbf{T}(q \to B)}{S, \mathbf{T}q, \mathbf{T}B}\mathbf{T}\to \textit{Atom}, \text{ with } q \text{ an atom}$$

$$\frac{S, \mathbf{F}(A \to B)}{S_c, \mathbf{T}A, \mathbf{F}B}\mathbf{F}\to \quad \frac{S, \mathbf{F_c}(A \to B)}{S_c, \mathbf{T}A, \mathbf{F_c}B}\mathbf{F_c}\to$$

$$\frac{S, \mathbf{T}(\neg A)}{S, \mathbf{F_c}A}\mathbf{T}\neg \qquad \frac{S, \mathbf{F}(\neg A)}{S_c, \mathbf{T}A}\mathbf{F}\neg \qquad \frac{S, \mathbf{F_c}(\neg A)}{S_c, \mathbf{T}A}\mathbf{F_c}\neg$$

$$\frac{S, \mathbf{T}((A \wedge B) \to C)}{S, \mathbf{T}(A \to (B \to C))}\mathbf{T}\to\wedge \quad \frac{S, \mathbf{T}(\neg A \to B)}{S_c, \mathbf{T}A | S, \mathbf{T}B}\mathbf{T}\to\neg$$

$$\frac{S, \mathbf{T}((A \vee B) \to C)}{S, \mathbf{T}(A \to p), \mathbf{T}(B \to p), \mathbf{T}(p \to C)}\mathbf{T}\to\vee$$

$$\frac{S, \mathbf{T}((A \to B) \to C)}{S_c, \mathbf{T}A, \mathbf{F}p, \mathbf{T}(p \to C), \mathbf{T}(B \to p) | S, \mathbf{T}C}\mathbf{T}\to\to$$

where $S_c = \{\mathbf{T}A | \mathbf{T}A \in S\} \cup \{\mathbf{F_c}A | \mathbf{F_c}A \in S\}$ and

p is a new atom

Fig. 1. The **Tab** calculus

$\mathbf{T}\neg$, which introduces sign $\mathbf{F_c}$, does not require the duplication of the negated formula to which it applies and therefore, for instance, all proofs of doubly negated classical propositional tautologies, which do not contain implications, can be proved without duplications. As we are dealing with semantic tableaux, we can say that sign $\mathbf{F_c}$ captures within the syntax the semantics of intuitionistic negation in terms of Kripke models in so far as the truth of a negation $\neg A$ in a node Γ implies that A is always false in every Δ node connected

with Γ and sign $\mathbf{F_c}$ is the *syntactical witness* of this. In [16] Fitting points out that sign $\mathbf{F_c}$ is optimal w.r.t. the treatment of negation in tableau calculi for intuitionism and still in [16] Waaler and Wallen present sign $\mathbf{F_c}$ in the framework of sequent calculi for the same logic. Since in rule $\mathbf{T}\rightarrow$ duplication on its left-hand side was still necessary, the problem of obtaining a propositional calculus without duplication remained unsolved. The same problem, mutatis mutandis, arose in sequent calculi (contraction rule) and natural deduction calculi (reuse of assumptions). In [29] (pp. 624-625), where **ft**, one of the first sequent based ATP for intuitionistic predicate logic is presented, we can find a lengthy and detailed discussion about the computational and applicative restrictions related to the necessity of duplication. Finally, in [11] a correct and complete contraction-free sequent calculus for intuitionistic propositional logic is presented, while in [25] the ideas outlined in [11] are successfully applied to obtain a correct and complete tableaux calculus for intuitionistic first-order logic, which does not require any duplication in its propositional part [1]. The rule $\mathbf{T}\rightarrow$ is subdivided into five rules according to the logical structure of the antecedent of the implication. This treatment of implication is more syntactical than semantical. The implications are rewritten in formula(s) intuitionistically equivalent, which are less complex as regards implications, and the subformula property is lost, but in any case a duplication-free calculus, which is optimal w.r.t. computations, is obtained. The calculus of Fig. 1 (first introduced by Fiorino in his PhD thesis [14] and published in [2]), which is the logical basis of the prover PITP and of its variants described here, is optimized by including in rule $\mathbf{T}\rightarrow\rightarrow$ some ideas introduced by Hudelmaier in [18] in order to obtain an $O(n \log n)$-space decision procedure for intuitionistic propositional logic by using a particular sequent calculus. The calculus of Fig. 1 is of the same complexity as Hudelmaier's. The logical complexity of the formula(s) obtained by applying the rules of Fig. 1 is in a sense less than the logical complexity of premises, and this guarantees a computational bound of proofs containing branching and backtracking. This property will be explained in detail at a later stage. To conclude, we reaffirm that the calculus of Fig. 1 does not allow the duplication of formulas and therefore the sets S of signed formulas cannot be considered multisets.

Given a Kripke model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$, a world $\alpha \in P$, a formula A and a set of swffs S, the definition of $\rhd$ is as follows:

- $\alpha \rhd \mathbf{T}A$ (α realizes $\mathbf{T}A$) iff $\alpha \Vdash A$;
- $\alpha \rhd \mathbf{F}A$ iff $\alpha \nVdash A$;
- $\alpha \rhd \mathbf{F_c}A$ iff $\alpha \Vdash \neg A$;
- $\alpha \rhd S$ iff α realizes every swff in S;
- $\underline{K} \rhd S$ iff $\rho \rhd S$.

[1] The present version of **ft** [29] uses in its propositional part a contraction-free sequent calculus derived from [11].

A proof table (or proof tree) for S is a tree, rooted in S and obtained by the subsequent application of the rules of the calculus. As an example, let $S = \{\mathbf{T}(B \wedge C), \mathbf{F}(A \vee B)\}$ and $H = \mathbf{T}(A \wedge B)$. With "rule $\mathbf{T}\wedge$ applies to $S \cup \{H\}$ taking H as main swff" we mean that $\mathbf{T}\wedge$ applies to $S \cup \{H\}$ as $\frac{S,H}{S,\mathbf{T}A,\mathbf{T}B}\mathbf{T}\wedge$. If no confusion arises, we say that *a rule applies to* $S \cup \{H\}$ or, equivalently, *a rule applies to* H. Finally, with $Rule(H)$ we mean the rule related to H (in our example $Rule(\mathbf{T}(A \wedge B))$ is $\mathbf{T}\wedge$). We emphasize that in the premise of the rules the notation S, H means $S \cup \{H\}$ and H does not belong to S.

For every proof table for S, the depth of the tree and the number of symbols occurring in nodes of a branch is linearly bounded in the number of symbols occurring in S (see [2] for further details). This is the key feature to implement a depth-first decision procedure whose space complexity is $O(n \log n)$; as a matter of fact, it is well-known that to generate all the proof tables in the search space and to visit them with a depth-first strategy, it is sufficient to have a stack containing, for every node of the visited branch, the index of the main swff and a bit to store if the left branch has been visited [18].

Now, we give the tableau closure conditions. A set S of swffs is closed if at least one of the following conditions holds:

(1) S contains a conjugate pair, that is $\{\mathbf{T}A, \mathbf{F}A\} \subseteq S$ or $\{\mathbf{T}A, \mathbf{F_c}A\} \subseteq S$;
(2) S contains $\mathbf{F}\top$, $\mathbf{F_c}\top$ or $\mathbf{T}\bot$.

Proposition 1 *If a set of swffs S is closed, then for every Kripke model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$, $\rho \not\Vdash S$.*

The proposition is proved in Appendix A

A *closed proof table* is a proof table whose leaves are all closed sets. A closed proof table is a proof of the calculus and a formula A is provable iff there exists a closed proof table for $\{\mathbf{F}A\}$.

For every rule of the calculus it is easy to prove that if there exists a Kripke model $\underline{K} = \langle P,\leq,\rho,\Vdash\rangle$ and $\alpha \in P$ such that α realizes the premise of the rule, then there exists (a possibly different) Kripke model $\underline{K}' = \langle P', \leq', \rho', \Vdash'\rangle$ and $\beta \in P'$ such that β realizes the conclusion. This is the main step to prove the soundness of the calculus:

Theorem 2 (Soundness) *Let A be a wff. If there exists a closed proof table starting from $\{\mathbf{F}A\}$, then A is valid.*

To prove that a given calculus is complete one can show that if a formula is valid, then by generating all the possible proofs of the calculus, sooner or later a proof is found. This naive way to proceed, although complete and finite,

has a huge search space. The calculus has several invertible rules, that is, the realizability of one of the sets in the conclusion implies the realizability of the premise. This suggests the standard way to apply the rules that reduces the search space and preserves the completeness: *build a proof giving precedence to invertible rules and when no invertible rule is applicable, then apply the non-invertible rules in all possible ways.* This procedure is present in the papers proving the completeness by using model theoretic techniques (see e.g. [15,25]). This basic strategy can be further refined by introducing new optimization criteria which reduce the search space while still preserving the completeness. These criteria are described in the following section.

3 From the Basic to the Optimized Algorithm

In the following we sketch the recursive procedure Basic Tab(S), which is the backbone of PITP. Given a set S of swffs, Basic Tab(S) returns either a closed proof table for S or `NULL` (if there exists a Kripke model realizing S). To describe Basic Tab we use the following notation. Let S be a set of swffs, let $H \in S$ and let S_1 or $S_1 \mid S_2$ be the nodes of the proof tree obtained by applying to S the rule $Rule(H)$ corresponding to H. If Tab_1 and Tab_2 are closed proof tables for S_1 and S_2 respectively, then $\frac{S}{\mathrm{Tab}_1} Rule(H)$ or $\frac{S}{\mathrm{Tab}_1 \mid \mathrm{Tab}_2} Rule(H)$ denote the closed proof table for S defined in the obvious way. Moreover, $\mathcal{R}_i(H)$ ($i \in \{1,2\}$) denotes the set containing the swffs of S_i which replaces H. For instance:
$\mathcal{R}_1(\mathbf{T}(A \wedge B)) = \{\, \mathbf{T}A, \mathbf{T}B \,\}$,
$\mathcal{R}_1(\mathbf{T}(A \vee B)) = \{\mathbf{T}A\}$, $\mathcal{R}_2(\mathbf{T}(A \vee B)) = \{\mathbf{T}B\}$,
$\mathcal{R}_1(\mathbf{T}((A \to B) \to C)) = \{\, \mathbf{T}A, \mathbf{F}p, \mathbf{T}(B \to p), \mathbf{T}(p \to C) \,\}$,
$\mathcal{R}_2(\mathbf{T}((A \to B) \to C)) = \{\mathbf{T}C\}$.

The procedure Basic Tab divides the formulas into six groups according to their behavior with respect to branching and backtracking:
$\mathcal{C}_1 = \{\mathbf{T}(A \wedge B)$, $\mathbf{F}(A \vee B)$, $\mathbf{F_c}(A \vee B)$, $\mathbf{T}(\neg A)$, $\mathbf{T}(p \to A)$ with p an atom, $\mathbf{T}((A \wedge B) \to C)$, $\mathbf{T}((A \vee B) \to C)\}$;
$\mathcal{C}_2 = \{\mathbf{T}(A \vee B)$, $\mathbf{F}(A \wedge B);\}$
$\mathcal{C}_3 = \{\mathbf{F}(\neg A)$, $\mathbf{F}(A \to B)\}$;
$\mathcal{C}_4 = \{\mathbf{T}((A \to B) \to C)$, $\mathbf{T}(\neg A \to B)\}$;
$\mathcal{C}_5 = \{\mathbf{F_c}(A \to B)$, $\mathbf{F_c}(\neg A)\}$;
$\mathcal{C}_6 = \{\mathbf{F_c}(A \wedge B)\}$.
We call $\mathcal{C}_i$-swffs ($i = 1, \ldots, 6$) the swffs of the group $\mathcal{C}_i$ and $\mathcal{C}_i$-rules the rules related to $\mathcal{C}_i$-swffs. The intuition behind these groups is related to the fact that, in order to reduce search space, it is convenient to apply invertible rules before the non-invertible ones. Moreover, to reduce the number of nodes in the tableau proofs, the rules with one set of formulas in the conclusion are applied first. Thus, the rules of group $\mathcal{C}_1$ are invertible and contain exactly one set of

swffs in the conclusion, the rules of group $\mathcal{C}_2$ are invertible and contain exactly two sets of swffs in the conclusion, the rules of group $\mathcal{C}_3$ are non-invertible and contain exactly one set of swffs in the conclusion and the rules of group $\mathcal{C}_4$ are non-invertible and contain exactly two sets of swffs in the conclusion. The rules of groups $\mathcal{C}_5$ and $\mathcal{C}_6$ are non-invertible but they are not in $\mathcal{C}_3$ and $\mathcal{C}_4$ because, as we prove in the Completeness Lemma (Lemma 7, page 22), we do not lose completeness if $\mathcal{C}_5$ and $\mathcal{C}_6$-rules are applied only when no other rule applies. Under this condition the application of $\mathcal{C}_5$ and $\mathcal{C}_6$-rules is invertible and no backtracking is required. On the other hand, to get completeness, backtracking is in general unavoidable when $\mathcal{C}_3$ and $\mathcal{C}_4$-rules are applied. Backtracking is avoidable when $\mathcal{C}_3$ or $\mathcal{C}_4$-rules apply to $S \cup \{H\}$ taking H main swff and S does not contain **F**-swffs. By inspecting the conclusion of $\mathcal{C}_3$ and $\mathcal{C}_4$-rules it is easy to check that in this case S_c coincides with S. This strategy can be seen as a basic optimization bounding both branching and backtracking.

FUNCTION BASIC TAB (S)
1. If S is a closed set, then, by definition, S is a closed proof table and BASIC TAB returns S;
2. If S contains a $\mathcal{C}_1$-swff H, then apply $Rule(H)$ to S and let S' be the result. If $\pi =$ BASIC TAB(S') is a proof, then $\frac{S}{\pi}Rule(H)$ is a proof for S, otherwise there is no proof for S and BASIC TAB returns `NULL`;
3. If S contains a $\mathcal{C}_2$-swff H, then apply $Rule(H)$ to S and let S_1 and S_2 be the result. If both $\pi_1 =$ BASIC TAB(S_1) and $\pi_2 =$ BASIC TAB(S_2) are proofs, then $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$ is a proof for S, otherwise BASIC TAB returns `NULL`;
4. If S contains a $\mathcal{C}_3$ or $\mathcal{C}_4$-swff, then:
4.1 if S contains exactly one **F**-swff H and H is a $\mathcal{C}_3$-swff, then apply $Rule(H)$ to S and let S' be the result. If $\pi =$ BASIC TAB(S') is a proof, then $\frac{S}{\pi}Rule(H)$ is a proof for S, otherwise there is no proof for S and BASIC TAB returns `NULL`.
4.2 If S does not contain any **F**-swff, then let H be any $\mathcal{C}_4$-swff, apply $Rule(H)$ to S and let S_1 and S_2 be the result. If both $\pi_1 =$ BASIC TAB(S_1) and $\pi_2 =$ BASIC TAB(S_2) are proofs, then $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$ is a proof for S, otherwise BASIC TAB returns `NULL`.
4.3 If S contains a $\mathcal{C}_3$-swff H such that S' is the result of applying $Rule(H)$ to S and $\pi =$ BASIC TAB(S') is a proof, then $\frac{S}{\pi}Rule(H)$ is a proof for S;
4.4 If S contains a $\mathcal{C}_4$-swff such that the sets S_1 and S_2 are the result of applying $Rule(H)$ to S and both $\pi_1 =$ BASIC TAB(S_1) and $\pi_2 =$ BASIC TAB(S_2) are proofs, then $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$ is a proof for S;
If in Points **4.3** and **4.4** no proof for S is found, then BASIC TAB returns `NULL`;
5. If S contains a $\mathcal{C}_5$-swff H, then apply $Rule(H)$ to S and let S' be the result. If $\pi =$ BASIC TAB(S') is a proof, then $\frac{S}{\pi}Rule(H)$ is a proof for S, otherwise BASIC TAB returns `NULL`;
6. If S contains a $\mathcal{C}_6$-swff H, then apply $Rule(H)$ to S and let S_1 and S_2 be the result. If both $\pi_1 =$ BASIC TAB(S_1) and $\pi_2 =$ BASIC TAB(S_2) are proofs, then $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$ is a proof for S, otherwise BASIC TAB returns `NULL`;

7. If none of the previous points apply, then Basic Tab returns `NULL`.

end function Basic Tab.

Starting from Basic Tab we have developed some optimizations to reduce both branching and backtracking. In the following, each optimization is described and the necessary proofs are provided. At the end of the section we describe Tab, the algorithm obtained by inserting in Basic Tab all the optimizations.

3.1 Adapting Simplification to Intuitionistic Logic

Simplification is an optimization technique, which allows us to bound both branching and backtracking. Simplification is described in [20], where it is applied to classical and modal logics. Here we discuss Simplification for intuitionistic logic. Following [20] we describe Simplification as a set of new rules to be added to those of the calculus in Fig. 1. In the calculus we introduce rules to replace formulas with $\top$ and $\bot$ and some suitable replacements to simplify $\top$ and $\bot$ in the context of intuitionistic logic. Formally, we add the following rules to those given in Fig. 1:

$$\frac{S, \mathbf{T}A}{S[A/\top], \mathbf{T}A}\,\mathrm{Repl}\,\mathbf{T} \qquad \frac{S, \mathbf{F_c}A}{S[A/\bot], \mathbf{F_c}A}\,\mathrm{Repl}\,\mathbf{F_c}$$

where $S[A/\top]$ (respectively $S[A/\bot]$) is the set of swffs resulting from the replacement in S of every occurrence of the wff A with $\top$ (respectively with $\bot$).

Here, the meaning of the rules is the same as in the case of classical logic. Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be a Kripke model, $\alpha \in P$ and A a formula. If $\alpha \Vdash A$, then in every subsequent world $\beta \geq \alpha$, $\beta \Vdash A$. Thus starting from α the forcing of A coincides with the forcing of $\top$. Similarly for $\mathbf{F_c}$, where we recall $\mathbf{F_c}A$ is a synonym of (or can be restated as) $\mathbf{T}\neg A$. Now, what can we say about a formula $\mathbf{F}A$? From a semantical point of view, if $\alpha \rhd \mathbf{F}A$, we know that $\alpha \nVdash A$. We cannot say anything about its equivalence with $\top$ or $\bot$. However, by analyzing the semantical meaning of the connectives, we discover that $\wedge$ and $\vee$ are "classical connectives", that is, the forcing in α of a conjunction $X \wedge Y$ (resp. of a disjunction $X \vee Y$) only depends on the forcing in α of the subformulas X and Y. This means that by knowing that $\alpha \nVdash A$ holds, we can conclude $\alpha \nVdash A \wedge B$ holds and $\alpha \Vdash A \vee B$ holds iff $\alpha \Vdash B$ holds. To summarize, the forcing of conjunctive or disjunctive formulas in a world α depends on the forcing in α of their subformulas. If $\alpha \nVdash A$ holds, then given a formula X, we can replace an occurrence of A in X with $\bot$, provided that the occurrence

of A we replace is not under the scope of a negation or an implication. As a result of these considerations we can introduce the following rule

$$\frac{S, \mathbf{F}A}{S\{A/\bot\}, \mathbf{F}A}\,\mathrm{Repl}\,\mathbf{F}$$

where in this case, the replacement $S\{A/\bot\}$ applies to every formula B occurring in S, provided that its sign is $\mathbf{T}$ or $\mathbf{F}$, as follows:

- $\bot$, if B is A;
- B if B is of the kind $\neg C$ or $C \to D$ or B is a propositional variable different from A;
- $C\{A/\bot\} \wedge D\{A/\bot\}$, if B is of the kind $C \wedge D$;
- $C\{A/\bot\} \vee D\{A/\bot\}$, if B is of the kind $C \vee D$;

This means that the replacement in curly brackets is different from the replacement in square brackets: replacement in curly brackets proceeds recursively and stops when a connective $\to$ or $\neg$ is found. For instance, $X \to Y[Y/\bot]$ produces the wff $X \to \bot$, whereas $X \to Y\{Y/\bot\}$ produces the wff $X \to Y$. We recall that $\mathbf{F_c}$ is a synonym of $\mathbf{T}\neg$, thus every subformula of a $\mathbf{F_c}$-formula is under the scope of the outer negation and the replacement $\mathbf{F_c}B\{A/\bot\}$ leaves $\mathbf{F_c}B$ unchanged. This explains why the rule Repl $\mathbf{F}$ does not apply the replacement $S\{A/\bot\}$ to the $\mathbf{F_c}$-formulas occurring in S.

The simplification rules for the connectives $\wedge$ and $\vee$ are the boolean ones given in Appendix C. For the connectives $\to$ and $\neg$, whose intuitionistic semantics differs from classical semantics, the simplification rules are given below as replacement rules:

$$\frac{S}{S[\top \to A/A]}\,\mathrm{Simp}\,\top\to \qquad \frac{S}{S[A \to \top/\top]}\,\mathrm{Simp}\,\to\top$$

$$\frac{S}{S[\neg\top/\bot]}\,\mathrm{Simp}\,\neg\top \qquad \frac{S}{S[\neg\bot/\top]}\,\mathrm{Simp}\,\neg\bot$$

$$\frac{S}{S[A \to \bot/\neg A]}\,\mathrm{Simp}\,\to\bot \qquad \frac{S}{S[\bot \to A/\top]}\,\mathrm{Simp}\,\bot\to$$

where the conclusion $S[a/b]$ of a rule of the kind $\frac{S}{S[a/b]}$ applied to the premise S is obtained by replacing (in S) every occurrence of the (sub)formula a with b. By using the semantical meaning of $\top$ and $\bot$ it is easy to prove that the replacements affect neither the correctness nor the completeness, thus these replacements rules can be applied in any step of a tableau proof:

Proposition 3 *Let S be a set of swffs. For every substitution $S[a/b]$ given above, the formula $(a \to b) \wedge (b \to a)$ is intuitionistically valid.*

The proposition is proved in Appendix B.

In the following, we give an example of proof exploiting Simplification. We prove that the formula

$$((\neg\neg A \to A) \to (\neg A \vee A)) \to (\neg A \vee \neg\neg A)$$

is not valid. In the following proof tables, when some confusion could arise, beside the rule name we indicate the main swff.

$$\dfrac{\dfrac{\mathbf{F}(((\neg\neg A \to A) \to (\neg A \vee A)) \to (\neg A \vee \neg\neg A))}{\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A)), \mathbf{F}(\neg A \vee \neg\neg A)}\mathbf{F}\to}{\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A)), \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)}\mathbf{F}\vee$$

Let S be $\{\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A)), \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)\}$. Now, by applying $\mathbf{F}\neg$ to S taking $\mathbf{F}(\neg A)$ as main swff we get the following open tableau:

$$\dfrac{\dfrac{\dfrac{\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A)), \mathbf{T}A}{\mathbf{T}((\neg\neg\top \to \top) \to (\neg\top \vee \top)), \mathbf{T}A}\text{Repl}\,\mathbf{T},\quad \mathbf{T}A}{\mathbf{T}((\neg\neg\top \to \top) \to \top), \mathbf{T}A}\text{Bool Simplif}}{\mathbf{T}\top, \mathbf{T}A}\text{Simp}\to\top$$

where with "Bool Simplif" we mean the well known simplifications performed on the connectives $\vee$ and $\wedge$ given in Appendix C. By applying $\mathbf{F}\neg$ to S taking $\mathbf{F}(\neg\neg A)$ as main swff we get the following open tableau:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A)), \mathbf{T}(\neg A)}{\mathbf{T}((\neg\top \to A) \to (\top \vee A)), \mathbf{T}(\neg A)}\text{Repl}\,\mathbf{T},\quad \mathbf{T}(\neg A)}{\mathbf{T}((\neg\top \to A) \to \top), \mathbf{T}(\neg A)}\text{Bool Simplif}}{\mathbf{T}\top, \mathbf{T}(\neg A)}\text{Simp}\to\top}{\mathbf{T}\top, \mathbf{F_c}A}\mathbf{T}\neg$$

Finally, by applying $\mathbf{T}\to\to$ to S taking $\mathbf{T}((\neg\neg A \to A) \to (\neg A \vee A))$ as main swff we get the sets

$$\{\mathbf{T}(\neg\neg A), \mathbf{F}p, \mathbf{T}(A \to p), \mathbf{T}(p \to (\neg A \vee A))\}$$

and

$$\{\mathbf{T}(\neg A \vee A), \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)\}$$

respectively the left and the right conclusion of $\mathbf{T}\to\to$. The right conclusion of $\mathbf{T}\to\to$ has a closed proof table:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{T}(\neg A \vee A), \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)}{\mathbf{T}(\bot \vee A), \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)}\text{Repl}\,\mathbf{F},\quad \mathbf{F}(\neg A)}{\mathbf{T}A, \mathbf{F}(\neg A), \mathbf{F}(\neg\neg A)}\text{Bool Simplif}}{\mathbf{T}A, \mathbf{F}(\neg\top), \mathbf{F}(\neg\neg\top),}\text{Repl}\,\mathbf{T},\quad \mathbf{T}A}{\mathbf{T}A, \mathbf{F}\bot, \mathbf{F}(\neg\bot)}\text{Simp}\,\neg\top}{\mathbf{T}A, \mathbf{F}\bot, \mathbf{F}\top}\text{Simp}\,\neg\bot$$

On the other hand, the left conclusion of $\mathbf{T}\rightarrow\rightarrow$ has no closed proof table:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{T}(\neg\neg A),\mathbf{F}p,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow(\neg A\vee A))}{\mathbf{F_c}(\neg A),\mathbf{F}p,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow(\neg A\vee A))}\mathbf{T}\neg}{\mathbf{F_c}(\neg A),\mathbf{F}p,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow(\neg A\vee A))}\text{Repl}\,\mathbf{F_c},\quad \mathbf{F_c}(\neg A)}{\mathbf{F_c}(\neg A),\mathbf{F}p,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow(\bot\vee A))}\text{Bool Simplif}}{\mathbf{F_c}(\neg A),\mathbf{F}p,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow A)}\mathbf{F_c}\neg}{\mathbf{T}A,\mathbf{T}(A\rightarrow p),\mathbf{T}(p\rightarrow A)}\text{Repl}\,\mathbf{T},\quad \mathbf{T}A}{\mathbf{T}A,\mathbf{T}(\top\rightarrow p),\mathbf{T}(p\rightarrow\top)}\text{Simp}\rightarrow\top}{\mathbf{T}A,\mathbf{T}(\top\rightarrow p),\mathbf{T}\top}\text{Simp}\top\rightarrow}{\mathbf{T}A,\mathbf{T}p,\mathbf{T}\top}$$

We emphasize that Simplification applied to the example above reduces both the number of branching in tableau proofs and avoids backtracking. The proofs where Simplification is applied do not contain branches and backtracking is not necessary. Without this optimization branching and backtracking would be necessary.

Remark 4 *To reduce branching we could adopt the KE-style tableau rules ([9,10]), where, for instance, the* $\mathbf{T}\vee$ *rule of Fig. 1 would be replaced by the rules*

$$\frac{S,\mathbf{T}(A\vee B),\mathbf{F}A}{S,\mathbf{T}B,\mathbf{F}A}KE\text{-}\mathbf{T}\vee_1 \qquad \frac{S,\mathbf{T}(A\vee B),\mathbf{F}B}{S,\mathbf{T}A,\mathbf{F}B}KE\text{-}\mathbf{T}\vee_2$$

As remarked in [20] these rules are a restricted form of Simplification. For instance, if we apply the KE-$\mathbf{T}\vee_1$ *to the set* $\{\mathbf{T}(A\vee B),\mathbf{F}A\}$ *we derive the set* $\{\mathbf{T}B,\mathbf{F}A\}$*. On the other hand, by applying the Simplification technique to the set* $\{\mathbf{T}(A\vee B),\mathbf{F}A\}$ *we obtain the set* $\{\mathbf{T}(B\{A/\bot\}),\mathbf{F}A\}$*. Thus, while KE-rule derives* $\mathbf{T}B$*, Simplification derives* $\mathbf{T}(B\{A/\bot\})$ *whose length is shorter than* $\mathbf{T}B$ *if* A *occurs in* B *and this, in the general case, should reduce the length of the proof. Thus, an alternative presentation of our calculus with the simplification rules consists in inserting the bivalence rule*

$$\frac{S}{S,\mathbf{T}A|S,\mathbf{F}A}$$

and removing the rules $\mathbf{T}\vee$ *and* $\mathbf{F}\wedge$*. We cannot replace* $\mathbf{T}\rightarrow$ *and* $\mathbf{T}\neg$ *since classical and intuitionistic logic give different meaning to implication and negation. Finally, we observe that the rule* $\mathbf{T}\rightarrow$ *Atom is derivable by the Simplification rules, thus it could be removed from the tableau rules without losing the completeness.*

3.2 Exploiting the Symmetry of the Formulas to Avoid Backtracking

We present a technique which allows us to reduce backtracking. Such a technique is based on the fact that sometimes in a set two (or more) formulas

H and H' occur having the same occurrence of the same connective in the same place. As an example the formulas $(A \rightarrow (B \vee C)) \rightarrow (B \vee C)$ and $(C \rightarrow (B \vee A)) \rightarrow (B \vee A)$ have this property. Roughly speaking, if we can turn the former formula into the latter by a permutation τ on propositional variables, then from the information about the satisfiability of the latter we also get information about the satisfiability of the former without performing any further proof step. In our example by using the permutation τ defined as $\tau(A) = C$, $\tau(B) = B$, $\tau(C) = A$, we can turn the former formula into the second. Since the aim of our technique is to bound backtracking, before going into detail we describe the ideas to bound backtracking presented in [31].

To bound the search space, [31] describes a decision procedure in which backtracking is bounded by a semantical technique inspired by the completeness theorem. The completeness theorem proves the satisfiability (realizability in our context) of a set S under the hypothesis that S does not have any closed proof table. As an example, let

$$S = \{\mathbf{F}(A \rightarrow B), \mathbf{T}((A \rightarrow B) \rightarrow C), \mathbf{F}C, \mathbf{T}D\}.$$

¿From S we can define the Kripke model $\underline{K}(S) = \langle P,\leq,\rho,\Vdash\rangle$ such that $P = \{\rho\}$ and $\rho \Vdash D$ (see Fig. 2(a)). Note that $\underline{K}(S)$ realizes $\mathbf{T}D$ but $\underline{K}(S)$ does not realize S. To prove the realizability of S, the realizability of

$$S_1 = \{\mathbf{T}A, \mathbf{F}B, \mathbf{T}((A \rightarrow B) \rightarrow C), \mathbf{T}D\}$$

and one between

$$S_2 = \{\mathbf{T}A, \mathbf{F}p, \mathbf{T}(B \rightarrow p), \mathbf{T}(p \rightarrow C), \mathbf{T}D\}$$

and

$$S_3 = \{\mathbf{F}(A \rightarrow B), \mathbf{T}C, \mathbf{F}C, \mathbf{T}D\}$$

have to be proved. Since S_3 is not realizable because $\mathbf{T}C, \mathbf{F}C \in S_3$, the realizability of S_1 and S_2 must be proved. From S_1 we define the Kripke model $\underline{K}(S_1) = \langle\{\alpha\},\leq,\alpha,\Vdash\rangle$, where $\alpha \leq \alpha$, $\alpha \Vdash D$ and $\alpha \Vdash A$ such that $\underline{K}(S_1) \rhd S_1$. If we glue $\underline{K}(S_1)$ above $\underline{K}(S)$ we get a new Kripke model $\underline{K} = \langle\{\rho, \alpha\},\leq,\rho,\Vdash\rangle$ where $\rho \leq \rho$, $\rho \leq \alpha$, $\alpha \leq \alpha$, $\rho \Vdash D$,$\alpha \Vdash D$ and $\alpha \Vdash A$ (see Fig. 2(b)). Since $\underline{K} \rhd S$, we do not need to apply $\mathbf{T} \rightarrow\!\!\!\!\rightarrow$ to S in order to obtain S_2 and the Kripke model of Fig. 2(c). In this case the work on S_2 is spared.

In the general case, that is when S contains a set of the kind

$$\mathbf{F}(A_1 \rightarrow B_1), \ldots, \mathbf{F}(A_n \rightarrow B_n), \mathbf{T}((C_1 \rightarrow D_1) \rightarrow E_1), \ldots, \mathbf{T}((C_m \rightarrow D_m) \rightarrow E_m)$$

with $m + n > 1$, the information collected from non closed proof tables built from a set S is used to build a Kripke model $\underline{K}$. The procedure described in [31] prunes the search space, since in S not all the swffs requiring backtracking

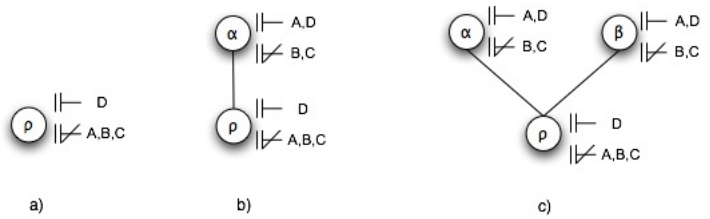

Fig. 2. Counter model for $S = \{\mathbf{F}(A \to B), \mathbf{T}((A \to B) \to C), \mathbf{F}C, \mathbf{T}D\}$.

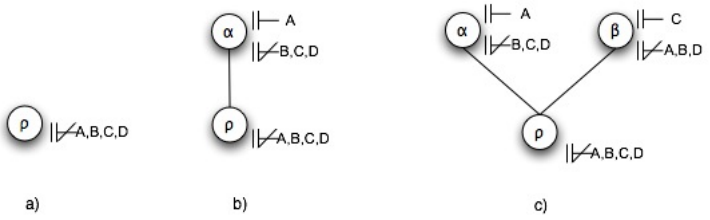

Fig. 3. Counter model for $S = \{\mathbf{F}(A \to B), \mathbf{F}(C \to D)\}$.

are considered, but only the swffs which, when checked, are not realized from the Kripke model at hand.

Now consider an example where the technique of [31] does not work. Let

$$S = \{\mathbf{F}(A \to B), \mathbf{F}(C \to D)\}.$$

¿From S we can define the Kripke model $\underline{K}(S) = \langle P,\leq,\rho,\Vdash\rangle$ such that $P = \{\rho\}$ and $\Vdash$ is the empty set (see Fig. 3 (a)). $\underline{K}(S)$ does not realize S. By applying $\mathbf{F} \to$ to S with $\mathbf{F}(A \to B)$ as the main formula we get

$$S_1 = \{\mathbf{T}A, \mathbf{F}B\}.$$

The underlying model is $\underline{K}(S_1) = \langle\{\alpha\},\leq,\alpha,\Vdash\rangle$ with $\alpha \Vdash A$. $\underline{K}(S_1)$ glued above $\underline{K}(S)$ gives rise to a model that does not realize $\mathbf{F}(C \to D)$ (see Fig. 3 (b)). Thus we must backtrack. We apply $\mathbf{F} \to$ to S with $\mathbf{F}(C \to D)$ as the main formula. We get

$$S_2 = \{\mathbf{T}C, \mathbf{F}D\}.$$

The underlying model is $\underline{K}(S_2) = \langle\{\beta\},\leq,\beta,\Vdash\rangle$ such that $\beta \leq \beta$ and $\beta \Vdash C$ realizes S_2. By gluing $\underline{K}(S_1)$ and $\underline{K}(S_2)$ above $\underline{K}(S)$ the resulting model $\underline{K} = \langle\{\rho, \alpha, \beta\},\leq,\rho,\Vdash\rangle$ such that

$$\rho \leq \alpha, \rho \leq \beta, \rho \leq \rho, \alpha \leq \alpha, \beta \leq \beta, \alpha \Vdash A \text{ and } \beta \Vdash C \tag{1}$$

realizes S (see Fig. 3 (c)). To avoid backtracking we propose a technique based on the observation that there exists a symmetry between S_1 and S_2. As a matter of fact, we can define a permutation $\tau : \mathcal{PV} \to \mathcal{PV}$ such that $\tau(C) = A$ and $\tau(D) = B$. By the definition of τ, it is easy to check that

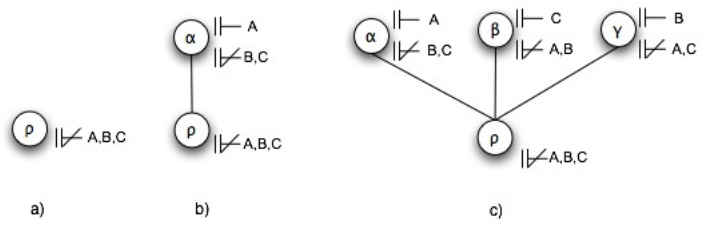

Fig. 4. Counter model for the axiom schema characterizing the logic of binary trees.

$\tau(S_2) = S_1$ holds[2]. Moreover, the Kripke model $\underline{K}(S_2) = \langle\{\alpha\}, \leq, \alpha, \Vdash'\rangle$ such that $\alpha \Vdash' p$ iff $\alpha \Vdash \tau(p)$, realizes S_2. As a matter of fact it is easy to prove the following

Proposition 5 *Let τ be a permutation on $\mathcal{PV}$ and let $\underline{K}_1 = \langle P, \leq, \rho, \Vdash\rangle$ and $\underline{K}_2 = \langle P, \leq, \rho, \Vdash'\rangle$ be Kripke models. If, for every $\alpha \in P$ and for every $p \in \mathcal{PV}$, $\alpha \Vdash p$ iff $\alpha \Vdash' \tau(p)$, then, for every $\alpha \in P$ and for every wff A, $\alpha \Vdash A$ iff $\alpha \Vdash' \tau(A)$.*

Now continuing with the example we get the resulting Kripke model $\underline{K}$ by means of the following two steps:

(1) We rename the worlds of $\underline{K}(S_2)$ in such a way that these new worlds are disjointed from those of $\underline{K}(S_1)$. We denote with $\underline{K}'(S_2)$ the Kripke model obtained from $\underline{K}(S_2)$ by renaming its worlds and defining the forcing relation accordingly. In our example $\underline{K}'(S_2) = \langle\{\beta\}, \leq, \beta, \Vdash''\rangle$ where for every $p \in \mathcal{PV}$, $\beta \Vdash'' p$ iff $\alpha \Vdash' p$.
(2) The Kripke model $\underline{K} = \langle\{\rho, \alpha, \beta\}, \leq, \rho, \Vdash\rangle$ is obtained by gluing $\underline{K}(S_1)$ and $\underline{K}'(S_2)$ above $\underline{K}(S)$ and is defined as (1) (see Fig. 3 (c)).

As another example, consider

$$S = \{\ \mathbf{T}((A \to (B \vee C)) \to (B \vee C)), \mathbf{T}((C \to (B \vee A)) \to (B \vee A)),$$
$$\mathbf{T}((B \to (C \vee A)) \to (C \vee A)), \mathbf{F}(B \vee (C \vee A))\ \},$$

where only a few steps are needed to obtain S starting from $\{\mathbf{F}H\}$, where H is the axiom schema

$$\bigwedge_{i=0}^{2} \left((P_i \to \bigvee_{j \neq i} P_j) \to \bigvee_{j \neq i} P_j \right) \to \bigvee_{i=0}^{2} P_i$$

characterizing the logic of binary trees (a logic in the family of k-ary trees logics, [7], also known as Gabbay-de Jongh logics). ¿From S we can define

[2] Given a swff H, a set of swffs S and a proof T, $\tau(H)$, $\tau(S)$ and $\tau(T)$ mean, respectively, the swff, the set of swffs and the proof obtained by applying τ in the obvious way. For instance, if $H = \mathbf{T}(q \to p)$ and the permutation τ is such that $\tau(q) = a$ and $\tau(p) = b$, then $\tau(H) = \mathbf{T}(a \to b)$.

the model $\underline{K}(S) = \langle P,\leq,\rho,\Vdash\rangle$ such that $P = \{\rho\}$ and $\Vdash$ is the empty set (see Fig. 4 (a)). $\underline{K}(S)$ does not realize S. By applying $\mathbf{T}\,{\rightarrow\!\!\rightarrow}$ to S with $H = \mathbf{T}(((A \rightarrow (B \vee C)) \rightarrow (B \vee C)))$ we get

$$\begin{aligned} S_1 = \{&\mathbf{T}(B \vee C), \mathbf{T}((C \rightarrow (B \vee A)) \rightarrow (B \vee A)),\\ &\mathbf{T}((B \rightarrow (C \vee A)) \rightarrow (C \vee A)), \mathbf{F}(B \vee (C \vee A))\} \end{aligned}$$

and

$$\begin{aligned} S_2 = \{&\mathbf{T}((C \rightarrow (B \vee A)) \rightarrow (B \vee A)), \mathbf{T}((B \rightarrow (C \vee A)) \rightarrow (C \vee A)),\\ &\mathbf{T}A, \mathbf{F}p, \mathbf{T}((B \vee C) \rightarrow p), \mathbf{T}(p \rightarrow (B \vee C))\}. \end{aligned}$$

Since S_1 is not realizable, that is, S_1 has a closed proof table, to prove the realizability of S we have to prove the realizability of S_2. The set S_2 defines the Kripke model $\underline{K}(S_2) = \langle\{\alpha\},\leq,\alpha,\Vdash\rangle$, where $\alpha \leq \alpha$ and $\alpha \Vdash A$. Thus $\underline{K}(S_2) \rhd S_2$ holds. Now, if we glue $\underline{K}(S_2)$ above $\underline{K}(S)$ we get a new model $\underline{K}'(S) = \langle\{\rho,\alpha\},\leq,\rho,\Vdash\rangle$, where $\rho \leq \rho, \rho \leq \alpha, \alpha \leq \alpha$ and $\alpha \Vdash A$ (see Fig. 4 (b)). $\underline{K}'(S)$ does not realize S. Thus we must backtrack twice:

(i) By applying $\mathbf{T}\,{\rightarrow\!\!\rightarrow}$ to S with $H = \mathbf{T}((C \rightarrow (B \vee A)) \rightarrow (B \vee A))$ we get,

$$\begin{aligned} S_3 = \{&\mathbf{T}((A \rightarrow (B \vee C)) \rightarrow (B \vee C)),\\ &\mathbf{T}((B \rightarrow (C \vee A)) \rightarrow (C \vee A)),\\ &\mathbf{T}C, \mathbf{F}q, \mathbf{T}((B \vee A) \rightarrow q), \mathbf{T}(q \rightarrow (B \vee A))\} \end{aligned}$$

and

$$\begin{aligned} S_4 = \{&\mathbf{T}((A \rightarrow (B \vee C)) \rightarrow (B \vee C)), \mathbf{T}(B \vee A),\\ &\mathbf{T}((B \rightarrow (C \vee A)) \rightarrow (C \vee A)), \mathbf{F}(B \vee (C \vee A))\}. \end{aligned}$$

(ii) By applying $\mathbf{T}\,{\rightarrow\!\!\rightarrow}$ to S with $H = \mathbf{T}((B \rightarrow (C \vee A)) \rightarrow (C \vee A))$ we get

$$\begin{aligned} S_5 = \{&\mathbf{T}((A \rightarrow (B \vee C)) \rightarrow (B \vee C)),\\ &\mathbf{T}((C \rightarrow (B \vee A)) \rightarrow (B \vee A)),\\ &\mathbf{T}B, \mathbf{F}r, \mathbf{T}((C \vee A) \rightarrow r), \mathbf{T}(r \rightarrow (C \vee A))\} \end{aligned}$$

and

$$\begin{aligned} S_6 = \{&\mathbf{T}((A \rightarrow (B \vee C)) \rightarrow (B \vee C)),\\ &\mathbf{T}((C \rightarrow (B \vee A)) \rightarrow (B \vee A)),\\ &\mathbf{T}(C \vee A), \mathbf{F}(B \vee (C \vee A))\} \end{aligned}$$

In a few steps we find that S_4 and S_5 are not realizable, hence they have closed tableau. From S_3 we define the Kripke model $\underline{K}(S_3) = \langle\{\beta\},\leq,\beta,\Vdash\rangle$ where $\beta \leq \beta$ and $\beta \Vdash C$. $\underline{K}(S_3) \rhd S_3$. ¿From S_6 we define the Kripke model $\underline{K}(S_6) = \langle\{\gamma\},\leq,\gamma,\Vdash\rangle$ where $\gamma \leq \gamma$ and $\gamma \Vdash B$. $\underline{K}(S_6) \rhd S_6$. Thus by gluing $\underline{K}(S_2), \underline{K}(S_3)$ and $\underline{K}(S_6)$ above $\underline{K}(S)$ we get a model $\underline{K}$ realizing S (see Fig. 4 (c)). Since we can define the permutations τ_1 and τ_2 such that $\tau_1(S_3) = S_2$ and $\tau_2(S_6) = S_2$ we can avoid backtracking. Thus no proof of realizability of S_3 or S_6 is needed and the Kripke models realizing S_3 and S_6 can be obtained by applying the permutations on the forcing relation of the Kripke model for S_2.

The optimization we are discussing is based on the fact that the realizability of a swff H can be established by knowing that a swff H' is realizable and there exists a function τ on propositional variables that applied to H transforms H into H', that is $\tau(H)$ is syntactically equal to H'. Such a function τ also allows us to transform the Kripke model K' for H' in a Kripke model K for H. Thus, the problem is to find a permutation τ on propositional variables such that, given a realizable formula H' and a formula H, $\tau(H) = H'$. This implies that H is realizable too and given a Kripke model realizing H' we have a way, via τ, to build the Kripke model realizing H. The problem of finding such a τ is linear in the length of H and H' (as one can easily prove analyzing the function Build Perm described below). The constraint that $\tau(H)$ is syntactically equal to H' could be relaxed by considering the commutative properties of the disjunctions and conjunctions. But in this case, the search for such a τ is exponential in the length of the formulas. To summarize, we build a permutation τ giving syntactical identity between H and H' where τ is defined on the propositional variables occurring in H and H'. Thus, in our search for a permutation τ we disregard the commutativity of the disjunctions and conjunctions.

To avoid backtracking, Tab builds a permutation τ and uses it as follows. Let S be a set of swffs, let H and H' be $\mathcal{C}_3$-swffs of S and let us suppose that $U' = (S \setminus \{H'\})_c \cup \mathcal{R}_1(H')$ is realized by a Kripke model K'. Before applying $Rule(H)$, Tab checks if there exists a permutation τ from $\mathcal{PV}(S)$ to $\mathcal{PV}(S)$ such that for every $V \in U$, $\tau(V) \in U'$, where $U = (S \setminus \{H\})_c \cup \mathcal{R}_1(H)$. We already know that the set U' is realizable by the Kripke model $\underline{K}'$. Since we have a permutation τ, then the set U is realized by the Kripke model $\underline{K}$ having the same poset as $\underline{K}'$ and such that for every world α and every propositional variable p, $\alpha \Vdash_{\underline{K}'} p$ iff $\alpha \Vdash_{\underline{K}} \tau(p)$. Note that, although $\underline{K}' \rhd U'$ and there exists a permutation τ such that for every $V \in U$, $\tau(V) \in U'$, this does not imply that $\underline{K}' \rhd U$. Thus, we have taken a different route with respect to [31], where the realizability of U is checked on $\underline{K}'$ that realizes U' and the two methods work in different situations.

The construction of such a τ is described below. When Build Perm is called

the actual parameter τ is the empty function. BUILD PERM returns a permutation or FAIL (if BUILD PERM does not build the permutation):

FUNCTION BUILD PERM (H,H',τ)
1. If H and H' are propositional variables then if $\tau(H)$ and $\tau(H')$ are not defined, then BUILD PERM sets $\tau(H) = H'$, $\tau(H') = H$ and returns τ; otherwise BUILD PERM returns FAIL;
2. If H and H' are respectively of the kind $\neg U$ and $\neg U'$, then BUILD PERM returns the result of the call BUILD PERM (U,U',τ);
3. If H and H' are respectively of the kind $U_L \odot U_R$ and $U'_L \odot U'_R$, with $\odot \in \{\rightarrow, \wedge, \vee\}$, then let $\tau =$ BUILD PERM(U_L, U_R, τ). If τ is not FAIL, then BUILD PERM returns the result of the call BUILD PERM(U_R, U'_R, τ). In the other cases BUILD PERM returns FAIL.
END FUNCTION BUILD PERM

If BUILD PERM returns a permutation, then if for every $V \in U$, $\tau(V) \in U'$, then τ is used. Otherwise if BUILD PERM does not return a permutation or the previous check fails, then PITP considers that no permutation exists and solves U. Analogously for $\mathcal{C}_4$-swffs. Since our search for a permutation is narrow, many of them are disregarded. Thus at present this optimization gives advantages only in relation to the SYJ209 family formulas of ILTP v1.1.2. library (see Fig. 8 and Fig. 9, where this optimization is referred to **Perm**). BUILD PERM treats every component (connective or variable) of H and H' at most once, it is easy to see that BUILD PERM executes in linear time in the length of the input formulas.

Remark 6 *We could also define a permutation to prove that a set is not realizable. As a matter of fact, if S is not realizable and there exists a permutation τ such that $\tau(S) = S'$, then S' is not realizable. Thus, given a set S and a $\mathcal{C}_2$ or $\mathcal{C}_6$-swff $H \in S$, if $(S \setminus \{H\}) \cup \mathcal{R}_1(H)$ is closed and there exists a permutation τ such that $\tau((S \setminus \{H\}) \cup \mathcal{R}_1(H)) = (S \setminus \{H\}) \cup \mathcal{R}_2(H)$, then $(S \setminus \{H\}) \cup \mathcal{R}_2(H)$ is not realizable and the tableau proof for $(S \setminus \{H\}) \cup \mathcal{R}_1(H)$ can be translated via τ into a tableau proof for $(S \setminus \{H\}) \cup \mathcal{R}_2(H)$ (see Points 4 and 7 of* TAB*). As a trivial application, consider a valid wff $H(\underline{p})$, where $\underline{p} = \{p_1, \ldots, p_n\}$ are all the propositional variables occurring in H. To prove that $\{\mathbf{F}(H(\underline{p}) \wedge H(\underline{q}))\}$ is closed, it is sufficient to prove, by an application of $\mathbf{F}\wedge$, that $\{\mathbf{F}H(\underline{p})\}$ is closed and that there exists a permutation for which $\{\mathbf{F}H(\underline{p})\} = \tau(\{\mathbf{F}H(\underline{q})\})$.*

3.3 Further Optimizations

The notions given in the following are used to formalize the two optimizations described in this section. The idea behind both the optimizations is that if the information in a set of swffs S allows us to define a classical interpretation

fulfilling the formulas in S, then the set S is also intuitionistically satisfiable.

Now, our aim is to define a classical interpretation based on the swffs of a set S. Given a set of swffs S, the signed atoms of S are the elements of the set $\delta_S = \{H | H \in S \text{ and } H \text{ is a signed atom}\}$. Given δ_S, we denote with σ_{δ_S} the (classical) interpretation defined as follows: if $\mathbf{T}p \in \delta_S$, then $\sigma_{\delta_S}(p) = true$, $\sigma_{\delta_S}(p) = false$ otherwise. Given a classical interpretation σ, (I) $\sigma \rhd \mathbf{T}A$ iff $\sigma \models A$; (II) $\sigma \rhd \mathbf{F}A$ and $\sigma \rhd \mathbf{F_c}A$ iff $\sigma \not\models A$. Moreover, given a set of swffs S, $\sigma \rhd S$ iff for every $H \in S$, $\sigma \rhd H$.

In order to bound branching, TAB treats in a particular way the non atomic formulas where intuitionistic and classical interpretation coincide (by the Kripke semantics, these formulas are those containing only conjunctions and disjunctions). We say that a wff or a swff can be classically evaluated (*cle*-wff and *cle*-swff for short) iff conjunctions and disjunctions are the only connectives occurring in it.

Let S be a set of swffs such that every possible Simplification rule has been applied to it. In this case the only propositional variables occurring in a cle-swff are those that do not occur as swffs in S. Thus we can conclude that the **F**-cle-swffs occurring in S are realized by σ_{δ_S}. On the other hand, σ_{δ_S} does not realize the **T**-cle-swffs and when a related rule is applied to a **T**-cle-swff H, the non atomic swffs in $\mathcal{R}_i(H)$ ($i \in \{1, 2\}$) are not realized by the new underlying model.

The above considerations about cle-swffs imply that completeness is not lost if we disregard the cle-swffs that are realized by the model underlying S (see Point 4 of the function TAB and the Completeness Lemma for the details). As an example, consider the set of swffs

$$\begin{aligned} S = \{\ & \mathbf{T}((B \to A) \to (A \wedge (B \wedge C))), \\ & \mathbf{T}((B \to C) \to ((C \to B) \to (A \wedge (B \wedge C)))), \\ & \mathbf{T}((C \to A) \to ((A \to C) \to (A \wedge (B \wedge C)))), \\ & \mathbf{F}(A \wedge (B \wedge C))\ \}. \end{aligned}$$

The swff $\mathbf{F}(A \wedge (B \wedge C))$ is realized by σ_{δ_S}, thus TAB disregards such a swff and chooses one of other swffs in S. Note that BASIC TAB would consider $\mathbf{F}(A \wedge (B \wedge C))$, thus giving rise to a bigger proof table. This technique is our adaptation of *regularity* [17], a technique well known to people engaged in classical tableaux theorem proving.

Now we come to the last optimization. When neither a $\mathcal{C}_1$ nor a $\mathcal{C}_2$-rule apply to a set of formulas S, TAB checks if the Kripke model coinciding with the classical interpretation σ_{δ_S} realizes S (formally, TAB checks if $\sigma_{\delta_S} \rhd S$ holds).

In this case, since S is realizable, TAB does not need to go on with a useless search for a closed proof table (in Section 5 this optimization is referred to as **noInv**).

The following algorithm TAB is obtained from BASIC TAB by inserting all the optimizations given above. Given a set S of swffs, TAB(S) returns either a closed proof table for S or NULL (if there exists a Kripke model realizing S). Some instructions "return NULL" are labelled with **r1**,...,**r6**. In the Completeness Lemma we refer to such instructions by means of these labels.

FUNCTION TAB (S)
1. If S is a closed set, then TAB returns the proof S;

2. Let S' =SIMPLIFICATION(S). If $S' \neq S$, then let π = TAB(S'). If π is a proof, then TAB returns $\frac{S}{\pi}$SIMP, otherwise TAB returns NULL;

3. If a $\mathcal{C}_1$-rule applies to S, then let H be a $\mathcal{C}_1$-swff. If TAB($(S \setminus \{H\}) \cup \mathcal{R}_1(H)$) returns a proof π, then TAB returns the proof $\frac{S}{\pi}Rule(H)$, otherwise TAB returns NULL (**r1**);

4. If a $\mathcal{C}_2$-swff H different from a **F**-*cle*-swff belongs to S, then let π_1 = TAB($S \setminus \{H\} \cup \mathcal{R}_1(H)$). If π_1 is NULL, then TAB returns NULL. Otherwise, let π_2 = TAB($S \setminus \{H\} \cup \mathcal{R}_2(H)$). If π_2 is a proof, then TAB returns the proof $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$, otherwise ($\pi_2$ is NULL) TAB returns NULL;

5. If a $\mathcal{C}_3$ or $\mathcal{C}_4$-rule applies to S, then TAB proceeds as follows:
5.0 If $\sigma_{\delta_S} \rhd S$, then TAB returns NULL (**r2**).
5.1 If S contains exactly one **F**-swff H and H is a $\mathcal{C}_3$-swff, then if TAB($(S \setminus \{H\}) \cup \mathcal{R}_1(H)$) returns a proof π, then TAB returns the proof $\frac{S}{\pi}Rule(H)$, otherwise TAB returns NULL;

5.2 If S does not contain any **F**-swff, then let H be any $\mathcal{C}_4$-swff and let π_1 = TAB($S \setminus \{H\} \cup \mathcal{R}_1(H)$). If π_1 is NULL, then TAB returns NULL. Otherwise, let π_2 = TAB($S \setminus \{H\} \cup \mathcal{R}_2(H)$). If π_2 is a proof, then TAB returns the proof $\frac{S}{\pi_1 \mid \pi_2}Rule(H)$, otherwise ($\pi_2$ is NULL) TAB returns NULL;

5.3 Let $\{H_1, \ldots, H_n\}$ be all the $\mathcal{C}_3$-swffs in S. For $i = 1, \ldots, n$, the following instructions are iterated: if for every swff H_j ($j \in \{1, \ldots, i-1\}$) there is no permutation τ such that $(S \setminus \{H_j\})_c \cup \mathcal{R}_1(H_j) = \tau((S \setminus \{H_i\})_c \cup \mathcal{R}_1(H_i))$, then let π = TAB($(S \setminus \{H_i\})_c \cup \mathcal{R}_1(H_i)$). If π is a proof, then TAB returns the proof $\frac{S}{\pi}Rule(H_i)$;
5.4 Let $\{H_1, \ldots, H_n\}$ be all the $\mathcal{C}_4$-swffs in S. For $i = 1, \ldots, n$, the following Points (**5.4.1**) and (**5.4.2**) are iterated.
(**5.4.1**) If for every swff H_j ($j \in \{1, \ldots, i-1\}$) there is no permutation τ such that $(S \setminus \{H_j\}) \cup \mathcal{R}_2(H_j) = \tau((S \setminus \{H_i\}) \cup \mathcal{R}_2(H_i))$, then let $\pi_{2,i}$ = TAB($(S \setminus \{H_i\}) \cup \mathcal{R}_2(H_i)$). If $\pi_{2,i}$ is NULL, then TAB returns NULL (**r3**). If for every swff H_j, with $j \in \{1, \ldots, i-1\}$, there is no permutation τ such that $(S \setminus \{H_j\})_c \cup \mathcal{R}_1(H_j) = \tau((S \setminus \{H_i\})_c \cup \mathcal{R}_1(H_i))$, then if TAB($(S \setminus \{H_i\})_c \cup$

$\mathcal{R}_1(H_i)$) returns a proof π_1, TAB returns the proof $\frac{S}{\pi_1 \mid \pi_{2,i}} Rule(H_i)$.
(**5.4.2**) If Point (**5.4.1**) does not hold, then there exists a permutation τ and a swff H_j ($j \in \{1, \ldots, i-1\}$) such that $(S \setminus \{H_j\}) \cup \mathcal{R}_2(H_j) = \tau((S \setminus \{H_i\}) \cup \mathcal{R}_2(H_i))$. If for every swff H_u, with $u \in \{1, \ldots, i-1\}$, there is no permutation τ such that $(S \setminus \{H_u\})_c \cup \mathcal{R}_1(H_u) = \tau((S \setminus \{H_i\})_c \cup \mathcal{R}_1(H_i))$, then if TAB$((S \setminus \{H_i\})_c \cup \mathcal{R}_1(H_i))$ returns a proof π_1, TAB returns the proof $\frac{S}{\pi_1 \mid \tau^{-1}(\pi_{2,j})} Rule(H_i)$.
If in Points (**5.3**) and (**5.4**) TAB does not find any closed proof table, then TAB returns `NULL` (**r4**).

6. If a $\mathcal{C}_5$-rule applies to S, then let H be a $\mathcal{C}_5$-swff. If TAB$((S \setminus \{H\})_c \cup \mathcal{R}_1(H))$ returns a proof π, then TAB returns the proof $\frac{S}{\pi} Rule(H)$, otherwise TAB returns `NULL`;

7. If a $\mathcal{C}_6$-rule applies to S, then let H be a $\mathcal{C}_6$-swff. Let $\pi_1 =$ TAB$((S \setminus \{H\})_c \cup \mathcal{R}_1(H))$. If π_1 is `NULL`, then TAB returns `NULL`. Otherwise, let $\pi_2 =$ TAB$((S \setminus \{H\})_c \cup \mathcal{R}_2(H))$. If π_2 is a proof, then TAB returns $\frac{S}{\pi_1 \mid \pi_2} Rule(H)$, otherwise ($\pi_2$ is `NULL`) TAB returns `NULL` (**r5**);

8. If none of the previous points apply, then TAB returns `NULL` (**r6**).

END FUNCTION TAB.

PITP implements FUNCTION TAB. Regarding the strategy devised above one can wonder if performance improves when the order of Points 5.4.1 and 5.4.2 is swapped (this corresponds to visiting the non-invertible branch first). Since the invertible rules are applied before the non-invertible ones, there are clues that the choice made by FUNCTION TAB is wise. Besides PITP, we have developed PITPINV a variant of PITP that visits the non-invertible branch of the $\mathcal{C}_4$-rules first. Despite the fact that PITPINV outperforms the provers on the formulas of the ILTP v1.1.2 library (see Fig. 5), on randomly generated formulas its performance is poor. Thus we focus mainly on PITP.

4 Completeness

In order to prove the completeness of TAB, we prove that given a set of swffs S, if the call TAB(S) returns `NULL`, then we have enough information to build a model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd S$. To prove the proposition we need to introduce the functions $\sharp_{\rightarrow}$ and deg defined as follows:

- if p is an atom, then $\sharp_{\rightarrow}(p) = 0$;
- $\sharp_{\rightarrow}(A \circ B) = \sharp_{\rightarrow}(A) + \sharp_{\rightarrow}(B)$ with $\circ \in \{\wedge, \vee\}$;
- $\sharp_{\rightarrow}(\neg A) = \sharp_{\rightarrow}(A)$;
- $\sharp_{\rightarrow}(A \rightarrow B) = \sharp_{\rightarrow}(A) + \sharp_{\rightarrow}(B) + 7$;

- $\deg(\bot) = 1$ and $\deg(\top) = 1$;
- if p is a propositional variable, then $\deg(p) = 2$;
- $\deg(A \wedge B) = \deg(A) + \deg(B) + 2$;
- $\deg(A \vee B) = \deg(A) + \deg(B) + 9$;
- $\deg(A \rightarrow B) = \deg(A) + \deg(B) + \sharp_{\rightarrow}(A) + 1$;
- $\deg(\neg A) = \deg(A) + 1$;
- $\deg(S) = \sum_{H \in S} \deg(H)$.

It is easy to show that if S' is obtained from a set of swffs S by a recursive call of TAB (S), then $\deg(S') < \deg(S)$.

Lemma 7 (Completeness) *Let S be a set of swffs and suppose that* TAB(S) *returns the* NULL *value. Then, there is a Kripke model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd S$.*

Proof: The proof is by induction on the complexity, $\deg(S)$, of S.
Basis: if $\deg(S) = 0$, then S contains atomic swffs only. TAB(S) carries out the instruction labeled (***r6***). Moreover, S does not contain sets of the kind $\{\mathbf{T}p, \mathbf{F}p\}$, $\{\mathbf{T}p, \mathbf{F_c}p\}$, $\{\mathbf{T}\bot\}$, $\{\mathbf{F}\top\}$ and $\{\mathbf{F_c}\top\}$. Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be the Kripke model such that $P = \{\rho\}$ and $\rho \Vdash p$ iff $\mathbf{T}p \in S$. It is easy to show that $\rho \rhd S$.

Step: Let us assume by induction hypothesis that the proposition holds for all sets S' such that $\deg(S') < \deg(S)$. We prove that the proposition holds for S by inspecting all the possible cases where the procedure returns the NULL value. We prove some cases.
Case 1: the instruction labeled **r1** *has been performed.* By induction hypothesis there exists a Kripke model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd (S \setminus \{H\}) \cup \mathcal{R}_1(H)$, with $H \in \mathcal{C}_1$. We prove $\rho \rhd H$ by proceeding according to the cases of H. If H is of the kind $\mathbf{T}(A \wedge B)$, then by induction hypothesis $\rho \rhd \{\mathbf{T}A, \mathbf{T}B\}$, thus $\rho \Vdash A$ and $\rho \Vdash B$, and therefore $\rho \Vdash A \wedge B$. This implies $\rho \rhd \mathbf{T}(A \wedge B)$. The other cases for $H \in \mathcal{C}_1$ are similar.
Case 2: the instruction labeled **r2** *has been performed.* Thus $\sigma_{\delta_S} \rhd S$ holds. We use σ_{δ_S} to define a Kripke model $\underline{K}$ with a single world ρ such that $\rho \Vdash p$ iff $\sigma(p) = \mathit{true}$. Since ρ behaves as a classical interpretation, $\rho \rhd S$ holds.
Case 3: the instruction labeled **r3** *has been performed.* By induction hypothesis there exists a model $\underline{K}$ such that $\rho \rhd (S \setminus \{H_i\}) \cup \mathcal{R}_2(H_i)$, where $H_i \in \mathcal{C}_4$. Let us suppose that H is of the kind $\mathbf{T}((A \rightarrow B) \rightarrow C)$, thus $\rho \rhd \mathbf{T}C$ and this implies $\rho \rhd H_i$. The proof goes similarly if H_i is of the kind $\mathbf{T}(\neg A \rightarrow B)$.
Case 4: the instruction labeled **r4** *has been performed.* This implies that for every $H \in S \cap \mathcal{C}_i$ ($i \in \{3, 4\}$), we have two cases: (a) TAB$((S \setminus \{H\})_c \cup \mathcal{R}_1(H))$ = NULL, thus by induction hypothesis there exists a Kripke model $\underline{K}_H = \langle P_H, \leq_H, \rho_H, \Vdash_H \rangle$ such that $\rho_H \rhd (S \setminus \{H\})_c \cup \mathcal{R}_1(H)$; (b) there exists a permutation τ from $\mathcal{PV}(S)$ to $\mathcal{PV}(S)$ and a swff $H' \in S \cap \mathcal{C}_i$ such that TAB$((S \setminus \{H'\})_c \cup \mathcal{R}_1(H'))$ = NULL and $(S \setminus \{H'\})_c \cup \mathcal{R}_1(H') = \tau((S \setminus \{H\})_c \cup \mathcal{R}_1(H))$. Thus by Point (a) applied to H', there exists a Kripke model

$\underline{K}_{H'} = \langle P_{H'}, \leq_{H'}, \rho_{H'}, \Vdash_{H'} \rangle$ such that $\rho_{H'} \rhd (S \setminus \{H'\})_c \cup \mathcal{R}_1(H')$. By using τ we can translate $\underline{K}_{H'}$ into a model $\underline{K}_H = \langle P_H, \leq_H, \rho_H, \Vdash_H \rangle$, where $P_H = P_{H'}$, $\leq_H = \leq_{H'}$, $\rho_H = \rho_{H'}$ and for every world $\alpha \in P_H$, if $p \in \mathcal{PV}(S)$, then $\alpha \Vdash_H \tau(p)$ iff $\alpha \Vdash_{H'} p$. By definition of $\underline{K}_H$, it follows $\underline{K}_H \rhd (S \setminus \{H\})_c \cup \mathcal{R}_1(H))$;

Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be a Kripke model defined as follows:

$$
\begin{aligned}
P &= \bigcup_{H \in S \cap (\mathcal{C}_3 \bigcup \mathcal{C}_4)} P_H \cup \{\rho\};\\
\leq &= \bigcup_{H \in S \cap (\mathcal{C}_3 \bigcup \mathcal{C}_4)} \leq_H \cup \{(\rho, \alpha) | \alpha \in P\};\\
\Vdash &= \bigcup_{H \in S \cap (\mathcal{C}_3 \bigcup \mathcal{C}_4)} \Vdash_H \cup \{(\rho, p) | \mathbf{T}p \in S\}.
\end{aligned}
$$

By construction of $\underline{K}$, $\rho \rhd S$.
Case 5: the instruction labeled **r5** *has been performed.* We point out that $S \cap \mathcal{C}_1 = S \cap \mathcal{C}_2 = S \cap \mathcal{C}_3 = S \cap \mathcal{C}_4 = S \cap \mathcal{C}_5 = \emptyset$. By induction hypothesis there exists a model $\underline{K}_H = \langle P_H, \leq_H, \rho_H, \Vdash_H \rangle$ such that $\rho_H \rhd (S \setminus \{H\})_c \cup \mathcal{R}_2(H)$, where $H = \mathbf{F_c}(A \wedge B)$. Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be a Kripke model defined as follows: $P = P_H \cup \{\rho\}$, $\leq = \leq_H \cup \{(\rho, \alpha) | \alpha \in P\}$, $\Vdash = \Vdash_H \cup \{(\rho, p) | \mathbf{T}p \in S\}$. By the construction of $\underline{K}$, $\rho \rhd S$, in particular, by induction hypothesis $\rho_H \Vdash \neg B$ and therefore $\rho_H \Vdash \neg(A \wedge B)$. This implies $\rho \rhd \mathbf{F_c}(A \wedge B)$.
Case 6: the instruction **r6** *has been carried out.* In this case S contains atomic swffs, swffs of the kind $\mathbf{T}(p \to A)$ with $\mathbf{T}p \notin S$ and $\mathbf{F}$-cle-swffs. Let $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ be the Kripke model such that $P = \{\rho\}$ and $\rho \Vdash p$ iff $\mathbf{T}p \in S$. It is easy to show that $\rho \rhd S$. As a matter of fact, if $\mathbf{T}(p \to A) \in S$, since $\mathbf{T}p \notin S$, $\rho \nVdash p$ then $\rho \Vdash p \to A$. Note that all the propositional variables occurring in the $\mathbf{F}$-cle-swffs do not occur in S as swffs. This implies that every propositional variable occurring in a $\mathbf{F}$-cle-swff is not forced in ρ. Since cle-swffs contain conjunctions and disjunctions only, it immediately follows that ρ realizes all the $\mathbf{F}$-cle-swffs occurring in S. □

By Lemma 7 we immediately get the completeness of Tab.

Theorem 8 (Completeness) *If* $A \in \mathbf{Int}$*, then* Tab$(\{\mathbf{F}A\})$ *returns a closed proof table starting from* $\mathbf{F}A$*.*

5 Implementation and Results

We have implemented Tab as an iterative procedure in C++. In the following we emphasize some issues of the implementation of PITP. The procedure

Simplification is implemented as an iterative procedure. Differently from the description given in Section 3, the procedure Simplification is called after the application of a rule. The actual parameters of Simplification are two sets of swffs: the former, we call it S, is (one of) the conclusion(s) of the rule. The latter, we call it *New*, contains the new swffs introduced in S by the application of the rule. For every swff H in *New*, Simplification applies the related replacement rule to S. If H can be replaced in S, then after the replacement the simplifications are performed. If new swffs are obtained, then they are added to the set *New*. Simplification stops when all swffs in the set *New* have been considered for replacement. We do not go into details of the data structures we have introduced in order to minimize the time required to perform replacements and simplifications. We only emphasize that our formulas are implemented as graphs, in which if a formula A occurs as subformula of B two or more times, then B has two or more pointers towards A (thus we represent more occurrences of the same (sub)formula only once). Moreover every (sub)formula A has pointers towards all the formulas B having A as subformula. This helps when a formula A has to be replaced in a set a swffs. As a matter of fact, the replacement of a $\mathbf{T}$ and $\mathbf{F_c}$-swff is immediate. This organization of the data structures implies that the simplification rules are implemented in a bottom-up fashion, visiting the graph in an upward motion.

Our main source regarding the results is the site of ILTP library. Fig. 5 is built taking the detailed results related to each prover available from `http://-www.cs.uni-potsdam.de/ti/iltp/results.html`. The results show that PITP and PITPINV outperform the known theorem provers on ILTP v1.1.2 library. Within 10 minutes PITP decides 238 out of 274 formulas and PITPINV decides 262 out of 274 formulas of ILTP v1.1.2 [3] .

The library divides the formulas into several families. Every family contains formulas sharing the same pattern of increasing complexity. In Fig. 6 and Fig. 7 for every family we report the index of the largest formula which every prover is able to decide within 600s CPU time and in parenthesis the CPU time necessary to solve such a formula (some families of ILTP v1.1.2 are missing because they are decided within 1s by all provers). PITPINV solves all the formulas in all families except the formulas in SYJ202+1 family (the pigeon formulas). PITP solves all the formulas in five families, ft-C solves all the formulas of SYJ212+1 and it is the best prover in this family, LJT solves all the formulas of SYJ205+1 and it is the best prover in this family (ex aequo with PITP); finally, STRIP solves all the formulas in two families but in no class is it the best prover. PITP and PITPINV are the best provers of ILTP v1.1.2 library. Although PITPINV outperforms PITP on the ILTP v1.1.2 formulas, some experiments on randomly generated formulas show that

[3] The line *errors* refers to run-time errors. PITP and PITPINV give one error parsing a 4.98 GB formula.

	ft Prolog	ft C	LJT	STRIP	PITP	PITPINV
solved	188	199	175	205	238	262
(%)	68.6	72.6	63.9	74.8	86.9	95.6
proved	104	106	108	120	128	128
refuted	84	93	67	85	110	134
solved after:						
0-1s	173	185	166	182	215	227
1-10s	5	6	4	13	15	20
10-100s	6	7	2	8	6	12
100-600s	4	1	3	2	2	3
(>600s)	86	75	47	64	35	11
errors	0	0	52	5	1	1

Fig. 5. ft Prolog, ft C, LJT, STRIP and PITP on ILTP v1.1.2 formulas

PITP has the best performance. This suggests that the strategy given by the completeness theorem, where the invertible branch is explored first (PITP strategy), is, in the general case, better than the one where the non-invertible branch is explored first (PITPINV strategy). For this reason, we will consider PITP as our main prover and hereafter we refer to its performance.

	202 provable	205 provable	206 provable
ft Prolog	07 (516.55)	08 (60.26)	10 (144.5)
ft C	07 (76.3)	09 (85.84)	11 (481.98)
LJT	02 (0.09)	20 (0.01)	05 (0.01)
STRIP	07 (268.59)	14 (162.67)	20 (33.24)
PITP	9 (50.25)	20 (0.01)	20 (3.92)
PITPINV	9 (42.68)	20 (33.45)	20 (3.89)

Fig. 6. Provable ILTP v1.1.2 SYJ-formulas solved by classes

	207 refutable	208 refutable	209 refutable	211 refutable	212 refutable
ft Prolog	07 (358.05)	08 (65.41)	10 (543.09)	04 (66.62)	20 (0.01)
ft C	07 (51.13)	17 (81.41)	10 (96.99)	04 (17.25)	20 (0.01)
LJT	03 (2.64)	08 (0.18)	10 (461.27)	08 (546.46)	07 (204.98)
STRIP	04 (5.91)	08 (40.62)	10 (82.33)	09 (50.19)	20 (38.94)
PITP	06 (68.50)	20 (1.76)	10 (206.65)	20 (191.57)	20 (5.51)
PITPINV	20 (305.21)	20 (1.80)	20 (60.54)	20 (139.67)	20 (5.51)

Fig. 7. Refutable ILTP v1.1.2 SYJ-formulas solved by classes

In Fig. 10 and Fig. 11 we compare the number and, in parenthesis, the percentage of randomly generated formulas decided by PITP and STRIP within ten minutes CPU time (since the performance of ft-C was much worse than STRIP and PITP, the results of ft-C have been omitted; the results of LJT have been omitted since run-time errors ranged, depending on the type of simulation,

from 7% to 33%). All the results refer to randomly generated formulas containing one thousand connectives and decided on a Intel(R) Xeon(TM) CPU 3.00GHz, 2048 KB cache size and 2 GB RAM. Fig. 10 is related to formulas containing respectively one, ten and one hundred variables. Fig. 11 is related to formulas of the implicative fragment. We point out that the performances of PITP on the formulas of the implicative fragment containing one (respectively one hundred) variable(s) are comparable with the performances on the propositional formulas containing one (respectively one hundred) variable(s).

In Fig. 8 and Fig. 9 we compare different optimizations and give the results of their effectiveness on some classes of ILTP v1.1.2 formulas (in Fig. 8 in brackets we give the CPU time, whereas in Fig. 9 in square brackets we give the number of sets of swffs built by our provers in order to solve the formula). We run PITP on a Xeon 3.2GHz machine with 2MB cache and 2GB Ram (note that our computer is different from the one used to obtain the results in `http://-www.cs.uni-potsdam.de/ti/iltp/results.html`, thus the results in Fig. 6 and Fig. 7 should not be compared with the ones of Fig. 8 and Fig. 9).

Among the optimizations the most effective is **Simp**. When **Simp** is not active (see the column on the far right) the performances dramatically decrease in many families. With regard to the other optimizations, there are some advantages in some classes and disadvantages in others. The optimization **Perm** gives advantages only on the SYJ209 family formulas of ILTP. We remark that **Perm** gives advantages also on formulas characterizing finite k-ary trees quoted in the previous Subsection 3.2. The optimization **noInv** does not provide benefits when applied to ILTP formulas. Experiments on random formulas show that **Simp** reduces the cases where **noInv** is applied but does not substitute it. As a matter of fact, if we consider the set $S = \{\mathbf{T}(A \rightarrow B)\}$, where A is an implicative formula which is not realized by the classical interpretation giving false value to every propositional variable and B is a classical contradiction, then if **noInv** is used we can immediately conclude that S is realizable. Without **noInv** we have to apply the rule $\mathbf{T} \rightarrow\rightarrow$ and develop both the sets in its conclusion. A simulation on five thousand formulas containing two thousand connectives and one hundred variables gave the following results: PITP with **noInv** not active did not decide 88 formulas within ten minutes CPU time, whereas PITP with **noInv** active did not decide 68 formulas. On the 4912 formulas decided in both cases, PITP with **noInv** not active took 11324 seconds, whereas PITP with **noInv** active took 9238 seconds. Finally a simulation on five thousand formulas of the implicative fragment containing two thousand connectives and one hundred variables gave the following results: PITP with **noInv** not active did not decide 120 formulas within ten minutes CPU time, whereas PITP with **noInv** active did not decide 112. On the 4480 formulas decided in both cases, PITP with **noInv** not active took 5665 seconds, whereas PITP with **noInv** active took 3115 seconds.

WFF	PITP ALL	PITP -noInv	PITP -Regl	PITP -Perm	PITP -Simp
201	20 (0)	20 (0)	20 (0)	20 (0)	20 (0)
202	09 (50.11)	09 (49.84)	09 (48.12)	09 (50.88)	06 (19.90)
203	20 (0)	20 (0)	20 (0)	20 (0)	20 (0)
204	20 (0)	20 (0)	20 (0)	20 (0)	20 (0)
205	20 (0)	20 (0)	20 (0)	20 (0)	20 (0)
207	06 (75.95)	06 (75.14)	06 (76.65)	06 (74.22)	04 (11.31)
208	20 (0.19)	20 (0.19)	20 (0.19)	20 (0.19)	08 (428.73)
209	10 (222.45)	10 (219.31)	10 (224.23)	10 (406.36)	10 (357.65)
210	20 (0)	20 (0)	20 (0)	20 (0)	20 (0)
211	20 (223.15)	20 (209.77)	20 (225.38)	20 (208.32)	20 (597.29)
212	20 (0)	20 (0)	20 (0)	20 (0)	20 (0.28)

Fig. 8. Time comparison between PITP optimizations on ILTP v1.1.2 SYJ-formulas

WFF	PITP ALL	PITP -noInv	PITP -Regl	PITP -Perm	PITP -Simp
201	20 [1105]	20 [1105]	20 [1608]	20 [1105]	20 [10369]
202	09 [544733]	09 [544733]	09 [544733]	09 [544733]	06 [1119875]
203	20 [80]	20 [80]	20 [80]	20 [80]	20 [750]
204	20 [22]	20 [22]	20 [22]	20 [22]	20 [62]
205	20 [93]	20 [93]	20 [93]	20 [93]	20 [343]
207	06 [10439389]	06 [10439389]	06 [10439432]	06 [10439389]	04 [6664039]
208	20 [4235]	20 [4235]	20 [4236]	20 [4235]	08 [11348148]
209	10 [12288168]	10 [12288168]	10 [12288168]	10 [24343423]	10 [223561169]
210	20 [24]	20 [24]	20 [24]	20 [24]	20 [64]
211	20 [36699632]	20 [36699632]	20 [36699632]	20 [36699632]	20 [116897467]
212	20 [80]	20 [80]	20 [80]	20 [80]	20 [136]

Fig. 9. Generated set of swffs comparison between PITP optimizations on ILTP v1.1.2 SYJ-formulas

The memory usage can be summarized as follows: formula SYJ208+1.020 requires 700 MB and the increasing factor between two subsequent formulas of the family is 1.3; formula SYJ212+1.020 requires 133 MB and the increasing factor between two subsequent formulas of the family is 1.98; formula SYJ206+1.020 requires 110MB and the increasing factor between two subsequent formulas of the family is two. In all other cases memory consumption is under 25MB with an increasing factor less than 1.55. Moreover, we remark that PITP never went out of memory during the experiments on random formulas.

6 Conclusions

PITP and PITPINV are the fastest among the theorem provers for propositional intuitionistic logic in the ILTP Library. The optimizations implemented

Formulas containing 1 variable					
	0-1s	1-10s	10-100s	100-600s	> 600
STRIP	3232(91.870)	105(2.985)	73(2.075)	42(1.194)	66(1.876)
PITP	3517(99.98)	1(0.02)	0 (0.0)	0 (0.0)	0
Formulas containing 10 variables					
	0-1s	1-10s	10-100s	100-600s	> 600
STRIP	915(70.385)	92(7.077)	81(6.231)	35(2.692)	177(13.615)
PITP	1284(98.78)	15(1.15)	1(0.07)	0 (0.0)	0
Formulas containing 100 variables					
	0-1s	1-10s	10-100s	100-600s	> 600
STRIP	607(60.7)	50(5.0)	50(5.0)	30(3.0)	263(26.3)
PITP	985(98.5)	5(0.5)	4(0.4)	0(0.4)	2(0.2)

Fig. 10. PITP and STRIP on random formulas

Formulas containing 1 variable					
	0-1s	1-10s	10-100s	100-600s	> 600
STRIP	3004(84.811)	88(2.484)	61(1.722)	39(1.101)	350(9.881)
PITP	3539(99.91)	3(0.09)	0(0.0)	0(0.0)	0(0.0)
Formulas containing 100 variables					
	0-1s	1-10s	10-100s	100-600s	> 600
STRIP	52(33.3)	10(6.4)	5(3.2)	1(0.6)	88(56.4)
PITP	906(98.07)	2(0.21)	7(0.75)	2(0.21)	7(0.75)

Fig. 11. PITP and STRIP on implicative random formulas

in PITP are of paramount importance in order to achieve the performances described in this paper. As a matter of fact, in an earlier version PITP only decided 202 formulas of the ILTP Library; later the version described in [4] decided 215 formulas. Besides PITP, here we describe a variant called PITPINV. Both PITP and PITPINV implement the same optimizations and differ on the strategy employed in the application of the rules. PITP decides 238 formulas and implements the search strategy that visits the invertible branch of $\mathbf{T} \rightarrow\rightarrow$ and $\mathbf{T} \rightarrow \neg$ first. PITPINV decides 262 formulas and implements a search strategy which visits the non-invertible branch of $\mathbf{T} \rightarrow\rightarrow$ and $\mathbf{T} \rightarrow \neg$ first. Despite the result on the formulas on the ILTP Library, the performances of PITPINV are worse than the performances of PITP on randomly generated formulas.

As regards the optimizations the most effective is Simplification. This optimization is an adaptation of the one described in [20] for classical and modal logics. As it is in the tradition of our group (see e.g. [24,26]), we stressed the *classical content* of intuitionistic logic in order to simplify as much as possible its proof theoretic treatment. Our adaptation to intuitionistic logic is related to the semantical meaning of the **F** signed formulas since, unlike classical logic, in the intuitionistic case **F** signed formulas are not equivalent to $\bot$.

Also, we remark that the analysis of the optimizations are developed by means of model theoretic techniques based on the Kripke semantics of propositional intuitionistic logic.

In the future we plan to apply all the optimizations of PITP to IPTP, a parallel version of PITP where we use the technique of threads to manage parallel computation on a 128 CPU machine; extensions to other intermediate propositional logics should not present particular difficulties (see e.g. [3]). Finally, first order extension will require, in our opinion, much work including suitable adaptations of well established first order classical techniques.

Acknowledgements

We thanks the anonymous referees that helped us to improve the quality of the paper and Nuala Tansey for English revision.

References

[1] A. Avellone, M. Ferrari, C. Fiorentini, G. Fiorino, and U. Moscato. Esbc: an application for computing stabilization bounds. *Electronic Notes in Theoretical Computer Science*, 153(1):23–33, 2006.

[2] A. Avellone, G. Fiorino, and U. Moscato. A new $O(n \log n)$-space decision procedure for propositional intuitionistic logic. In A. Voronkov M. Baaz and J. Makowsky, editors, *LPAR 2002: Short Contributions, CSL 2003: Extended Posters*, volume VIII of *Kurt Gödel Society, Collegium Logicum*, pages 17–33, 2004.

[3] A. Avellone, C. Fiorentini, G. Fiorino, and U. Moscato. A space efficient implementation of a tableau calculus for a logic with a constructive negation. In J. Marcinkowski and A. Tarlecki, editors, *CSL*, volume 3210 of *Lecture Notes in Computer Science*, pages 488–502. Springer, 2004.

[4] A. Avellone, G. Fiorino, and U. Moscato. A Tableau Decision Procedure for Propositional Intuitionistic Logic. In *Proceedings of the 6th International Workshop on the Implementation of Logics*, volume 212 of *CEUR Workshop Proceedings*, pages 64–79, 2006.

[5] A. Avellone, G. Fiorino, and U. Moscato. Improvements to the tableau prover pitp. In N. Olivetti, editor, *TABLEAUX*, volume 4548 of *Lecture Notes in Computer Science*, pages 233–237. Springer, 2007.

[6] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Texts in theoretical computer science. Springer-Verlag, 2004.

[7] A. Chagrov and M. Zakharyaschev. *Modal Logic*. Oxford University Press, 1997.

[8] R. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice–Hall, Englewood Cliffs, New Jersey, 1986.

[9] M. D'Agostino and M. Mondadori. The Taming of the Cut. Classical Refutations with Analytic Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[10] M. D'Agostino. Tableaux Methods for Classical Propositional Logic. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 45–123. Kluwer, Dordrecht, 1999.

[11] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.

[12] U. Egly and S. Schmitt. On intuitionistic proof transformations, their complexity, and application to constructive program synthesis. *Fundamenta Informaticae*, 39(1-2):59–83, 1999.

[13] M. Ferrari, C. Fiorentini, and M. Ornaghi. Extracting exact time bounds from logical proofs. In A. Pettorossi, editor, *Logic Based Program Synthesis and Transformation, 11th International Workshop, LOPSTR 2001, Selected Papers*, volume 2372 of *Lecture Notes in Computer Science*, pages 245–265. Springer-Verlag, 2002.

[14] G. Fiorino. *Decision procedures for propositional intermediate logics*. PhD thesis, Dipartimento di Scienze dell'Informazione, Universita' degli Studi di Milano, Italy, 2001.

[15] M.C. Fitting. *Intuitionistic Logic, Model Theory and Forcing*. North-Holland, 1969.

[16] M.C. Fitting. Introduction. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*. Kluwer, Dordrecht, 1999.

[17] R. Hähnle. Tableaux and related methods. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

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

[19] P. Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theory. Bibliopolis, Napoli, 1984.

[20] F. Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. In H. de Swart, editor, *Proc. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Oosterwijk, The Netherlands*, volume 1397 of *Lecture Notes in Computer Science*, pages 217–232. Springer-Verlag, 1998.

[21] M. Mendler. Timing analysis of combinational circuits in intuitionistic propositional logic. *Formal Methods in System Design*, 17(1):5–37, 2000.

[22] P. Miglioli, U. Moscato, and M. Ornaghi. Trees in Kripke semantics and in intuitionistic refutation system. In E. Astesiano C. Bhoem, editor, *CAAP '81*, volume 112 of *Lecture Notes in Computer Science*, pages 316–330. Springer-Verlag, 1981.

[23] P. Miglioli, U. Moscato, and M. Ornaghi. An improved refutation system for intuitionistic predicate logic. *Journal of Automated Reasoning*, 12:361–373, 1994.

[24] P. Miglioli, U. Moscato, and M. Ornaghi. Abstract parametric classes and abstract data types defined by classical and constructive logical methods. *Journal of Symbolic Computation*, 18:41–81, 1994.

[25] P. Miglioli, U. Moscato, and M. Ornaghi. Avoiding duplications in tableau systems for intuitionistic logic and Kuroda logic. *Logic Journal of the IGPL*, 5(1):145–167, 1997.

[26] P. Miglioli, U. Moscato, M. Ornaghi, and G. Usberti. A constructivism based on classical truth. *Notre Dame Journal of Formal Logic*, 30(1):67–90, 1989.

[27] U. Moscato. *Intuizionismo e teoria della dimostrazione. Su un'estensione dei sistemi di refutazione aderente alla semantica dell'intuizionismo.* PhD thesis, Department of Philosophy, University of Milan, 1980.

[28] T. Raths, J. Otten, and C. Kreitz. The ILTP problem library for intuitionistic logic. *Journal of Automated Reasoning*, 38(1-3):261–271, 2007.

[29] D. Sahlin, T. Franzén, and S. Haridi. An intuitionistic predicate logic theorem prover. *Journal of Logic and Computation*, 2(5):619–656, 1992.

[30] R. Statman. Intuitionistic logic is polynomial-space complete. *Theoretical Computer Science*, 9(1):67–72, 1979.

[31] K. Weich. Decision procedures for intuitionistic propositional logic by program extraction. In H. de Swart, editor, *TABLEAUX*, volume 1397 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 1998.

Appendices

A Proof of Proposition 1

If S is closed because the first condition holds, then for some formula A, $\{\mathbf{T}A, \mathbf{F}A\} \subseteq S$ or $\{\mathbf{T}A, \mathbf{F_c}A\} \subseteq S$ holds. By the meaning of the signs and the definition of the forcing relation in Kripke models, the claim immediately follows. If S is closed because the second condition holds, then by definition of forcing of $\top$ and $\bot$ the claim immediately follows. $\square$

B Proof of Proposition 3

We prove only some meaningful cases. Let α be a world of a Kripke model.
Case $\top \to A/A$. We prove $((\top \to A) \to A) \wedge (A \to (\top \to A))$ is valid. Let $\alpha \Vdash \top \to A$. Since in every world β of every Kripke model $\beta \Vdash \top$ holds, immediately $\alpha \Vdash A$ follows. The case $\alpha \Vdash A \to (\top \to A)$ is immediate.
Case $S[\neg\bot/\top]$. We prove $(\neg\bot \to \top) \wedge (\top \to \neg\bot)$ is valid. The only interesting case is $\top \to \neg\bot$. Since there is no world forcing $\bot$, we immediately deduce that in every world of every Kripke model $\neg\bot$ is forced (please note that the result can be also obtained by noting that $\neg\bot$ is equivalent to $\bot \to \bot$ which is an intuitionistic formula)
Case $S[\bot \to A/\top]$. We prove $((\bot \to A) \to \top) \wedge (\top \to (\bot \to A))$ is valid. The only interesting case is $\top \to (\bot \to A)$. Since $\bot$ is not forced in any Kripke world, we immediately deduce that $\bot \to A$ is forced in every world of every Kripke model and then $\top \to (\bot \to A)$ is valid. $\square$

C Boolean simplification rules

Below we give the Boolean simplification rules that in the previous examples are denoted by Bool Simplif.

$$\frac{S}{S[A \vee \bot/A]} \qquad \frac{S}{S[\bot \vee A/A]} \qquad \frac{S}{S[A \vee \top/\top]} \qquad \frac{S}{S[\top \vee A/\top]}$$

$$\frac{S}{S[A \wedge \bot/\bot]} \qquad \frac{S}{S[\bot \wedge A/\bot]} \qquad \frac{S}{S[A \wedge \top/A]} \qquad \frac{S}{S[\top \wedge A/A]}$$