Labeled KE for Intuitionistic Propositional Logic*

Solares-Rojas, A., Baldi, P. & Rodriguez, R. O.

October 31, 2026

Abstract

The tableau-like system KE is generalized to intuitionistic propositional logic by means of labeled signed formulas and constraints between labels, mimicking the relational semantics. To improve on proof-search and exploiting the meaning of negation, we further endow the system with free-variables. The resulting system enjoys the subformula property and terminates, either with a proof or a finite countermodel, without any extra mechanism. Proof and countermodel search is guided by generalizations of traditional reasoning patterns.

Keywords: intuitionistic propositional logic, labeled deduction, KE tableaux, relational semantics, countermodel generation

1 Introduction

Intuitionistic propositional logic has many practical and theoretical applications. Accordingly, a plethora of proof systems and theorem provers based on them have been studied for it. In the context of automated deduction, sequent and tableau systems have been prominently investigated. Some important properties of a system in this context are termination, satisfaction of the subformula property, countermodel generation, modularity, efficiency and interpretability. In particular, from a human-oriented perspective, a good balance between the latter two is desirable in that, in addition to an efficient generation of proofs and countermodels, we also expect them to be understandable to us. Devising a system satisfying all the mentioned properties is clearly desirable but challenging. For example, Gentzen's early sequent system LJ, as well as its variant G3ip, enjoy the subformula property but do not terminate without the aid of extra mechanisms such as loop-checking. Their non-termination is due to the *duplication* of left-hand side implicative formulas. Namely, the main formula of the rule for left-implication occurs both in the left premise and the conclusion.

A solution to this duplication was popularized by Dyckhoff's sequent system LJT [8] (a.k.a. G4ip), by replacing the rule for left implication with four duplication-free rules depending on the form of the antecedent of the main implicative formula. Thus, root-first proof-search terminates in LJT without loops and,

*This article has been accepted for publication in Journal of Logic and Computation Published by Oxford University Press.

although the system gives up the subformula property, it can be guaranteed that its derivations have finite length by controlling a suitable measure of the degree of formulas. On the other hand, countermodel generation is not incorporated in this system. In fact, a dual terminating sequent system called CRIP [20]—based on invertibility of some rules of a multi-succedent version of LJT—is required to construct a countermodel from proof-search failure. Still, given the permanence of some non-invertible rules, refutability in CRIP is witnessed not by a single branch but by a tree. Each branch contains information to build only a partial countermodel of the initial sequent but, by "gluing" these partial models together, a total one is indeed obtained.

Adding control machinery to G3ip, Ferrari et al. [10] have introduced a terminating variant of it, called Gbu, which preserves the subformula property. Namely, sequents are annotated with u or b so to distinguish two phases in bottom-up proof-search: an $u$nblocked one where any rule can be backward applied, and a $b$locked one where only right-hand side rules can be so applied. In particular, the left premise of the left-implication rule is a b-sequent. This avoids the generation of branches of infinite length, and so guarantees termination. Analogously to LJT, however, countermodel generation in Gbu is not incorporated and requires a dual system called Rbu. Yet, some of the rules of this single-succedent dual system are non-invertible and so, as in CRIP, refutability therein is witnessed by a tree whose induced partial countermodels must be glued together.

Fiorentini et al. [13, 11] have introduced the currently most efficient sequent-like systems, which are equipped with SAT-based proof-search. These are essentially proof-theoretic versions of the SMT algorithm `intuit` introduced by Claessen and Rosén [2]. The latter was originally only a decision procedure providing no proof nor countermodel of its outputs. However, Fiorentini et al. [13] showed that it mimics a root-first proof-search strategy for a variant of LJT called $\mathrm{LJT}_{\mathtt{SAT}}$, to the effect that `intuit`'s search procedure can be reformulated to provide either an $\mathrm{LJT}_{\mathtt{SAT}}$ proof or a countermodel in the relational semantics. Those witnesses are thus formulated in a non-standard sequent system incorporating a fully classical rule whose applicability is checked by a SAT-solver. Indeed, this approach led to refine `intuit` with a restart operation, producing an algorithm outperforming the original version and whose proof-theoretic version is a yet less standard sequent system called $C^{\rightarrow}$ [11]. These sequent-like systems are based on a pre-processing of the tested formula into generalized clausal forms, and so are hardly comparable in proof-theoretic terms with standard proof systems. Thus, comparisons with them are rather focused on their relative efficiency via implementation, on an experimental basis, wherein SAT-based systems have shown expectedly superior. Nonetheless, the rewriting methods on which these systems are based dramatically depart from a direct treatment of the *meaning* of the intuitionistic connectives, which is crucial for producing witnesses where each inference step can be natively interpreted by human users.

The system investigated in this paper aims at balancing efficiency with interpretability. Namely, we introduce an intuitionistic version of the tableau-like system KE which, in the classical case, is proven to polynomially simulate standard tableaux while the reverse relation does not hold [6]. Indeed, KE has exponentially shorter proofs than standard tableaux w.r.t. some classes of formulas. KE's hallmark—inherited by our intuitionistic generalization—is reducing the amount of branching to a minimum by making all branches mutually exclusive. That is, it has only one branching rule that intuitively implements (respectively, a generalization

of) the Principle of Bivalence, and essentially corresponds to a non-eliminable *analytic*-cut rule. The rest of the rules have all a non-branching format and correspond to traditional reasoning patterns, such as (generalizations of) modus ponens, modus tollens, disjunctive syllogism, etc. It is the controlled application of the branching rule that avoids the redundant branchings intrinsically present in standard tableaux. In turn, the intuitionistic generalization is obtained by importing semantical notions into the object language via *labeled signed* formulas. Namely, signs, labels and constraints between the latter, are used to respectively mimic the forcing relation and the partial order relation at the object language level [see 14, 4, 5]. Indeed, our intuitionistic KE is designed so as to mirror countermodel search in the relational semantics [see 15]. So, it yields derivations where each inference step is directly interpretable in terms of that semantics.

Analogously to the duplication issue in sequent systems, however, tableau systems might involve a *reuse* of T-signed implicative formulas. This reuse can lead to a high inefficiency in proof-search and even yield derivations of infinite length. Unlabeled tableaux, such as Miglioli's et al. [18], can be shown to require no reuse at all, though at the cost of losing the subformula property, much as in LJT. Nonetheless, proof-finiteness is more easily shown in systems satisfying the subformula property, such as labeled tableaux. Thus, for example, Galmiche and Méry [15] provided labeled tableaux called TBI/IL, which enjoy the property and bound the reuse of T-signed implicative formulas. We shall detail the arguments in the context of our system in Section 5 below but, roughly, the reuse is bounded since the "universal scope" of those formulas can be shown to be over a finite domain of labels [see 4, 15]. Indeed, recently, Fellin and Negri [9] introduced a labeled multi-succedent sequent system called $\mathrm{G3I}_t$, for which termination is shown following essentially the same arguments. Yet, this latter system does not have the subformula property given that it incorporates a right-implication rule which violates it. What is more, Ferrari et al. Gbu's approach mentioned above essentially implements the same termination arguments, without directly appealing to semantic notions though, but drawing on a syntactic "evaluation" relation which ultimately mimics them. Our system adopts a further improvement on labeled tableaux, reducing the search-space by avoiding at all the reuse of T-signed negated formulas. Namely, we exploit the meaning of negation within the syntax by adding free-variables to the system [see 4, 15, 16]. This plays an analogous role as the third sign F_c in the unlabeled tableau system of Miglioli's et al. [18].

Unlike the unlabeled and single-succedent sequent systems mentioned above, the rules of Fellin and Negri's $\mathrm{G3I}_t$ are all invertible, rendering the system suitable for countermodel generation. A single completed (a.k.a. saturated) branch witnesses refutability, and there is no need of a refutation tree in a dual system nor model-gluing. The same effect is achieved in the labeled tableaux TBI/IL and the KE system introduced here. For these three systems, proof-search is stopped as soon as a single completed branch has been constructed and then a countermodel can be immediately extracted from such a branch without any extra recomposition from other completed branches. Indeed, given that labels mimic the semantics, all the information codifying the respective countermodel is readily readable from the completed branch. A further convenient property of labeled systems is their modularity for characterizing logics that admit relational semantics. Namely, they are usually designed by following easy translations from frame conditions or axioms into inference rules. This approach has been fruitfully applied to, for example, in-

termediate, modal and substructural logics [see 19], as well as combinations thereof [see 17]. Particularly, labeled KE systems have been developed for substructural [4], modal logics [16], and their combinations [5].

In this paper, we introduce a KE-style system for intuitionistic propositional logic that, in the spirit of [15, 9], closely follows countermodel bulding in the relational semantics [cf. 4]. We investigate its basic proof-theoretic and algorithmic properties. Next section recalls the relational semantics of the logic, which underlies the whole system. The latter is introduced in Section 3, where the labels mechanism is explained in detail. Intuitions behind the formulation of the rules are also briefly discussed. Section 4 is devoted to show soundness and completeness. The proof of the second given in Section 4, though simple, does not provide a complete algorithm for proof-search and countermodel generation. So, in Section 5 we make a slight modification to our system that allows us to introduce such a procedure and show that it terminates. Section 6 broadly compares our system with some popular alternative ones, pointing out future work.

2 Preliminaries

Formulas of intuitionistic propositional logic are built from a countable set $\mathcal{P} = \{p, q, r, \ldots\}$ of propositional variables and a set $\mathcal{C} = \{\neg, \wedge, \vee, \rightarrow\}$ of connectives, using the recursive definition $A ::= p \,|\, \neg A |\, A \wedge A \,|\, A \vee A \,|\, A \rightarrow A$.[1] We use $A, B, C, \ldots$ for denoting arbitrary formulas and say that the *degree* of a formula A is the number of occurrences of connectives in A. For every formula A, a *subformula* of A is defined inductively: A is a subformula of A; for every binary connective $\circ$, if $B \circ C$ is a subformula of A, then so are B and C; if $\neg B$ is a subformula of A, so is B; nothing else is a subformula of A.

Kripke relational semantics [see 1] underlies the KE tableaux introduced in this paper. A *frame* is a triple $\mathcal{F} = \langle S, R, \rho \rangle$, where S is a non-empty set, R is a partial order on S and ρ is the root of $\langle S, R \rangle$. Usually, elements of S are called *information states* and R is called *accessibility relation.* A *valuation in a frame* $\mathcal{F}$ is a mapping $v : \mathcal{P} \mapsto \wp(S)$, such that for all $a, b \in S$, if $a \in v(p)$ and aRb, then $b \in v(p)$, i.e. $v(p)$ is an upward closed set. A *model* is a pair $\mathcal{M} = \langle \mathcal{F}, v \rangle$. Given a model $\mathcal{M}$, v is recursively extended on arbitrary formulas, defining a *forcing relation* $\Vdash$:

$\mathcal{M}, a \Vdash p$ iff $a \in v(p)$;

$\mathcal{M}, a \Vdash A \wedge B$ iff $\mathcal{M}, a \Vdash A$ and $\mathcal{M}, a \Vdash B$;

$\mathcal{M}, a \Vdash A \vee B$ iff $\mathcal{M}, a \Vdash A$ or $\mathcal{M}, a \Vdash B$;

$\mathcal{M}, a \Vdash A \rightarrow B$ iff for all b, if aRb, then $\mathcal{M}, b \nVdash A$ or $\mathcal{M}, b \Vdash B$;

$\mathcal{M}, a \Vdash \neg A$ iff for all b, if aRb, then $\mathcal{M}, b \nVdash A$.

By structural induction on the degree of an arbitrary formula A, it follows that for every formula, if $\mathcal{M}, a \Vdash A$ and aRb, then $\mathcal{M}, b \Vdash A$. That is, the set of states that force A is upward closed. We shall refer to this property of the forcing

[1] As explained below, given that our system is aimed to exploit the meaning of negation for improving on proof-search, we treat this connective as primitive. However, our rules are of course compatible with the usual definition of the connective in terms of conditional and the *falsum* constant, i.e. $\neg A \stackrel{\text{def}}{=} A \rightarrow \bot$.

relation as *upward monotonicity*. Dually, the set of states that do not force A are downward closed. To wit, $\mathcal{M}, b \nVdash A$ and aRb imply $\mathcal{M}, a \nVdash A$. We in turn call this property *downward monotonicity*. Now, we say that A is *valid in* $\mathcal{M}$ iff $\mathcal{M}, \rho \Vdash A$. In contrast, $\mathcal{M}$ is said to be a *countermodel* of A iff $\mathcal{M}, \rho \nVdash A$. Finally, A is *valid* iff A is valid in all models $\mathcal{M}$.

3 Labeled KE tableaux

The system introduced below generalizes KE by using the expressive power of labels, and closely follows the reasoning behind countermodel building in the relational semantics [see 4, 15, 9]. Analogously to the unlabeled tableau system of Miglioli et al. [18], our system exploits the meaning of negation within the syntax for improving on proof-search by avoiding the reuse of T-signed negated formulas. However, differently from [18] where this is done by means of a sign F_c, we opt to do it via labels expressing free-variables that, from our viewpoint, are conceptually more transparent w.r.t. the underlying semantics and the search of countermodels therein. As one would expect, our system formulated without variables is complete since these are used only for reducing the search-space.

3.1 Labels, signs and constraints

The system is formulated by means of *labeled signed* formulas, "ls-formulas" for short [see 14, 4, 5, 15, 16]. Namely, expressions of one of the next three forms:

$$\mathsf{T}\, A : c_i \qquad \mathsf{F}\, A : c_i \qquad \mathsf{F}\, A : x_i$$

where A is a formula, T and F are *signs*, and c_i and x_i are *labels*. The *labeling language* consists of a countable set of *constants* $C = \{c_0, c_1, \dots\}$, a countable set of *variables* intended as universally quantified $V = \{x_0, x_1, \dots\}$, and a binary relation symbol $\preceq$ denoting a partial order over C. Intuitively, constants represent particular states, while variables represent generic ones which, however, must satisfy constraints between constants. We use l_i to denote indifferently a constant c_i or a variable x_i. In turn, $\preceq$ stands for the accessibility relation at the object language level, and so a *constraint* is an expression of the form $c_i \preceq c_j$ [see 15, 9].

The intuitive meaning of $\mathsf{T}\, A : c_i$ (respectively, $\mathsf{F}\, A : c_i$) is that A is *forced* (*non-forced*) at the state denoted by the constant c_i, while $\mathsf{F}\, A : x_i$ intuitively means that A is non-forced *at each* state c_j such that $c_i \preceq c_j$. Note that only F-signed formulas are associated with variable labels. This is because ls-formulas of the form $\mathsf{F}\, A : x_i$ are aimed to capture the meaning of negation within the syntax, which we exploit to improve on proof-search. That is, if the negation of a formula A is forced at a state c_i, then A is non-forced at *every* state c_j such that $c_i \preceq c_j$ [see 18].

The *conjugate* of an ls-formula $\mathsf{T}\, A : c_i$ is $\mathsf{F}\, A : c_i$ and vice versa. We use $\varphi, \psi, \theta \dots$ to denote arbitrary ls-formulas, and S as a variable ranging over $\{\mathsf{T}, \mathsf{F}\}$. With $\overline{\varphi}$ we denote the conjugate of φ. Finally, given that the logic can be characterized in terms of rooted models, as recalled in Section 2, we shall work with sets of labels for which there is some *root label* c_0 such that $c_0 \preceq c_j$ for all c_j.

3.2 Rules

The rules of our system are displayed in Table 1. They act on labels, signs and formulas. Some rules require that their premises satisfy corresponding constraints, while some other rules introduce a constraint to be satisfied by their conclusions [see 15]. Only the pair of rules for $\rightarrow$ and $\neg$ involving variables (i.e., $\mathsf{F}\ \rightarrow_2$ and $\mathsf{F}\neg_2$ in Table 1) impose a constraint both in their premise and conclusions. In turn, constraints keep record of the underlying accessibility relation imported into the syntax, so they play the role of *side conditions* and do not count as premises. In the following, we refer to rules as two-premise or one-premise, counting only the number of the ls-formulas acting as premises, that is, excluding the constraints which involve only labels. On the other hand, labels and constraints do matter for assessing whether two rule instances are distinct. Indeed, we consider *different* two rule instances that differ only in the involved labels and constraints, but have otherwise the same (simple) formulas in premises and conclusions.

The rules in Table 1 generalize KE hallmark of reducing the amount of branching to a minimum by making all branches mutually exclusive. Namely, all elimination rules have a linear format, while there is a unique branching rule that has no premises and is called *PB* since it expresses a generalized Principle of Bivalence. As for the elimination rules, we call the premise containing the connective that is to be eliminated *major* and, in the case of two-premise rules, we call the other premise *minor*. Henceforth we identify each elimination rule by using the *sign* of the major premise, followed by its main connective. When more than one rule correspond to that description, we use an index, reflecting the order of occurrence of the rule in Table 1. Note that we have introduced multi-conclusion versions of the $\mathsf{F}\vee$ and $\mathsf{T}\wedge$ rules in order to extract, without reuse, as much information as possible from each rule application.

PB intuitively says that for any formula A and for each state, either A is forced or non-forced there. As usual with KE-style systems, this branching rule may introduce arbitrary formulas, but, as we shall show below, its applications can be restricted so as to satisfy the subformula property. Our system also includes a couple of branch closure rules (those with a mark $\times$ as conclusion in Table 1) that express that once a "metalevel" contradiction has been obtained according to our labels mechanism, we stop the derivation process since it does not correspond to a model in the semantics. That is, these rules intuitively express a generalized Principle of Non-contradiction: for no A and no state, A is forced and non-forced there. Given the potential use of variables within a derivation, we have two different closure rules. The first makes use of the fact that A is non-forced at the state denoted by c_j, and the monotonicity of A being forced at c_i, where $c_i \preceq c_j$. The second essentially amounts to instantiating the variable x_j, where A is non-forced, with the constant c_k. By the monotonicity of A being forced at c_i, we also have that A is forced at c_k, thus producing a contradiction.

Some further clarifications are due on the rules in Table 1. First, note that the unique rule that introduces *variables* is $\mathsf{T}\neg$. We emphasize that variables are *not* necessary for the completeness of our system, but are used for reducing the search-space by exploiting the meaning of negation. Free-variables allow us to reduce the search-space by postponing the choice of their instantiations to close a branch until

$$\frac{\mathsf{F}\, A \vee B : l_i}{\begin{array}{l}\mathsf{F}\, A : l_i \\ \mathsf{F}\, B : l_i\end{array}}\ (\mathsf{F}\vee) \qquad \frac{\mathsf{T}\, A \wedge B : c_i}{\begin{array}{l}\mathsf{T}\, A : c_i \\ \mathsf{T}\, B : c_i\end{array}}\ (\mathsf{T}\wedge)$$

$$\frac{\begin{array}{l}\mathsf{T}\, A \vee B : c_i \\ \mathsf{F}\, A : l_j \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{T}\, B : c_i}\ (\mathsf{T}\vee_1) \qquad \frac{\begin{array}{l}\mathsf{T}\, A \vee B : c_i \\ \mathsf{F}\, B : l_j \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{T}\, A : c_i}\ (\mathsf{T}\vee_2)$$

$$\frac{\begin{array}{l}\mathsf{T}\, A \vee B : c_j \\ \mathsf{F}\, A : x_i \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{T}\, B : c_j}\ (\mathsf{T}\vee_3) \qquad \frac{\begin{array}{l}\mathsf{T}\, A \vee B : c_j \\ \mathsf{F}\, B : x_i \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{T}\, A : c_j}\ (\mathsf{T}\vee_4)$$

$$\frac{\begin{array}{l}\mathsf{F}\, A \wedge B : l_j \\ \mathsf{T}\, A : c_i \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{F}\, B : l_j}\ (\mathsf{F}\wedge_1) \qquad \frac{\begin{array}{l}\mathsf{F}\, A \wedge B : l_j \\ \mathsf{T}\, B : c_i \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{F}\, A : l_j}\ (\mathsf{F}\wedge_2)$$

$$\frac{\begin{array}{l}\mathsf{F}\, A \wedge B : x_i \\ \mathsf{T}\, A : c_j \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{F}\, B : x_j}\ (\mathsf{F}\wedge_3) \qquad \frac{\begin{array}{l}\mathsf{F}\, A \wedge B : x_i \\ \mathsf{T}\, B : c_j \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{F}\, A : x_j}\ (\mathsf{F}\wedge_4)$$

$$\frac{\begin{array}{l}\mathsf{T}\, A \to B : c_i \\ \mathsf{T}\, A : c_j \\ \boxed{c_i \preceq c_k \text{ and } c_j \preceq c_k}\end{array}}{\mathsf{T}\, B : c_k}\ (\mathsf{T}\ \to_1) \qquad \frac{\begin{array}{l}\mathsf{T}\, A \to B : c_i \\ \mathsf{F}\, B : l_j \\ \boxed{c_i \preceq c_j}\end{array}}{\mathsf{F}\, A : l_j}\ (\mathsf{T}\ \to_2)$$

$$\frac{\mathsf{F}\, A \to B : c_i}{\begin{array}{l}\mathsf{T}\, A : c_j \\ \mathsf{F}\, B : c_j \\ c_i \preceq c_j\end{array}}\ \begin{array}{l}c_j \text{ new} \\ (\mathsf{F}\ \to_1)\end{array} \qquad \frac{\begin{array}{l}\mathsf{F}\, A \to B : x_i \\ \boxed{c_i \preceq c_j}\end{array}}{\begin{array}{l}\mathsf{T}\, A : c_k \\ \mathsf{F}\, B : x_i \\ c_j \preceq c_k\end{array}}\ \begin{array}{l}c_k \text{ new} \\ (\mathsf{F}\ \to_2)\end{array}$$

$$\frac{\mathsf{T}\, \neg A : c_i}{\mathsf{F}\, A : x_i}\ (\mathsf{T}\neg) \qquad \frac{\mathsf{F}\, \neg A : c_i}{\begin{array}{l}\mathsf{T}\, A : c_j \\ c_i \preceq c_j\end{array}}\ \begin{array}{l}c_j \text{ new} \\ (\mathsf{F}\neg_1)\end{array}$$

$$\frac{\begin{array}{l}\mathsf{F}\, \neg A : x_i \\ \boxed{c_i \preceq c_j}\end{array}}{\begin{array}{l}\mathsf{T}\, A : c_k \\ c_j \preceq c_k\end{array}}\ \begin{array}{l}c_k \text{ new} \\ (\mathsf{F}\neg_2)\end{array} \qquad \frac{\begin{array}{l}\mathsf{T}\, A : c_i \\ \mathsf{F}\, A : c_j \\ \boxed{c_i \preceq c_j}\end{array}}{\times}\ (\text{closure}_1) \qquad \frac{\begin{array}{l}\mathsf{T}\, A : c_i \\ \mathsf{F}\, A : x_j \\ \boxed{c_i \preceq c_k \text{ and } c_j \preceq c_k}\end{array}}{\times}\ (\text{closure}_2)$$

$$\frac{}{\mathsf{T}\, A : c_i \mid \mathsf{F}\, A : c_i}\ (PB)$$

Table 1: KE-style rules with variables

enough information becomes available, and so discarding unnecessary expansion steps [4, 15, 16].[2] A *complete* KE-style system without variables is simply obtained from the rules in Table 1 by: discarding all the rules containing an explicit variable label x_i, writing c instead of l in the remaining rules, and adding in place of the $\mathsf{T}\neg$ rule the simpler version:

$$\frac{\mathsf{T}\,\neg A : c_i \qquad \boxed{c_i \preceq c_j}}{\mathsf{F}\,A : c_j}$$

We display this system in the Appendix. It has less rules than its version with variables, but it generally requires more expansion steps and so induces a wider search-space.

Note that the rules of $\neg$ are compatible with the usual definition $\neg A \stackrel{\text{def}}{=} A \to \bot$. As just explained, these rules are included only to improve the system's efficiency by reducing the search-space. Indeed, each of these rules can be viewed as a particular instance of a more general rule of $\to$. For example, $\mathsf{T}\neg$ can be regarded as a special case of $\mathsf{T}\ \to_2$ by taking $B = \bot$, which is forced in *no* state. Similarly, $\mathsf{F}\neg_1$ and $\mathsf{F}\neg_2$ may be seen as instances of $\mathsf{F}\ \to_1$ and $\mathsf{F}\ \to_2$, respectively. It is straightforward to verify that, under this interpretation of the rules for $\neg$, the syntactical character of the logic is preserved.

Second, observe that the rules make implicit use of monotonicity, as already pointed out for the pair of closure rules above. For another example, take the case of the generalized *modus tollens* ($\mathsf{T}\ \to_2$): given that $A \to B$ is forced at the state denoted by c_i and $c_i \preceq c_j$, by monotonicity, it is also forced at c_j. So that, since B is not forced at c_j, it follows that A is not forced at c_j. The rest of the cases making implicit use of monotonicity are similar. Thereby, our system does not need an explicit monotonicity rule and our closure rules do not require their corresponding pair of formulas to be forced at the same state [cf. 4]. Indeed, the system's completeness shown below guarantees us that this is actually the case. In this respect, note also that some rules make implicit use of the *upward* monotonicity of F-signed formulas associated with variables since, recall, these formulas exploit the meaning of $\neg A$ [see 18].

Third, note that rules for reflexivity and transitivity of $\preceq$ are not needed [cf. 17, 9]. Those properties are encoded in the fact that $\preceq$ denotes a partial order. So, within a derivation, we shall not display but leave implicit all constraints that can be obtained via the reflexivity and transitivity of $\preceq$.

We conclude this section with some basic definitions [see 4, 5, 15].

Definitions 3.1.

- Let φ be an ls-formula. A *derivation from* φ is any tree $\mathcal{T}$ whose root is labeled by φ, whose nodes are labeled by ls-formulas and whose edges correspond to applications of rules in Table 1 to preceding ls-fromulas and, possibly constraints in the same branch.

[2] It is easy to show that rules $\mathsf{T}\vee_3$ and $\mathsf{T}\vee_4$ can be simulated via *PB* and, respectively, $\mathsf{T}\vee_1$ or $\mathsf{T}\vee_2$. However, we opt to include them as primitive rules for naturalness rather than efficiency. Indeed, in the branching of the respective simulation, one branch closes immediately and the respective *PB* application is clearly ad hoc.

- A derivation $\mathcal{T}$ satisfies the *subformula property* iff for every ls-formula $\mathsf{S}A : l_i$ occuring in it, the formula A is a subformula of some formula B such that $\mathsf{S}B : l_j$ occurs above in the same branch.
- Given a branch $\mathtt{b}$ in a derivation $\mathcal{T}$, the *domain of* $\mathtt{b}$ is the set $C_{\mathtt{b}}$ of constants occurring in $\mathtt{b}$.
- The *accessibility relation induced by* $\mathtt{b}$, denoted by $\preceq_{\mathtt{b}}$ is the reflexive and transitive closure of $\preceq\restriction_{C_{\mathtt{b}}^2}$.
- A branch $\mathtt{b}$ is *closed* if, either it contains $\mathsf{T}\,A : c_i$ and $\mathsf{F}\,A : c_j$ with $c_i \preceq c_j$, or it contains $\mathsf{T}\,A : c_i$ and $\mathsf{F}\,A : x_j$ with $c_i \preceq c_k$ and $c_j \preceq c_k$; otherwise, $\mathtt{b}$ is *open*.
- A derivation $\mathcal{T}$ is *closed* when all its branches are closed; otherwise, $\mathcal{T}$ is *open*.
- A *proof* of a formula A is a closed derivation $\mathcal{T}$ from $\mathsf{F}\,A : c_0$. So, A is a *theorem* iff there is a proof $\mathcal{T}$ of A.

Some examples of proofs are displayed in Figure 1. Let us explain the first one, which corresponds to one of the Heyting's axioms. Each node is numbered for easy reference. Node ① contains only the ls-formula associated with the initial formula that we are testing for theoremhood. The ls-formulas and constraint in ② are obtained by applying the rule $\mathsf{F}\ \to_1$ to the ls-formula in ①. Now, the ls-formulas in ③ are obtained by applying rule $\mathsf{T}\wedge$ to the first ls-formula in ②. In turn, $\mathsf{T}\,A : c_2$ and $c_1 \preceq c_2$ in ④ is obtained by applying the rule $\mathsf{F}\,\neg_1$ to the second ls-formula in ②. Then, $\mathsf{T}\,B : c_2$ in ⑤ is obtained by applying rule $\mathsf{T}\ \to_1$ to the ls-formulas $\mathsf{T}\,A \to B : c_1$ and $\mathsf{T}\,A : c_2$, in ③ and ④ respectively, provided the constraint $c_1 \preceq c_2$ in ④ and the (non-displayed) constraint $c_2 \preceq c_2$. Analogously, $\mathsf{T}\,\neg B : c_2$ in ⑥ is obtained from $\mathsf{T}\,A \to \neg B : c_1$ and $\mathsf{T}\,A : c_2$. Thus, $\mathsf{F}\,B : x_2$ in ⑦ is obtained from $\mathsf{T}\,\neg B : c_2$ by applying the rule $\mathsf{T}\,\neg$. Finally, closure in ⑧ is obtained from $\mathsf{T}\,B : c_2$ and $\mathsf{F}\,B : x_2$ by applying the second closure rule in Table 1, given the constraint $c_2 \preceq c_2$. Now, the second example particularly illustrates the role of variables. Indeed, contrast it with the analogous derivation in the Appendix' Figure 6 which uses no variables. In turn, the third example requires at least one application of the branching rule. The last example is a proof of the so-called *weak* Peirce law.

4 Basic properties

4.1 Soundness

Given that the labels and the relational structure on them transparently simulate the underlying semantics, soundness is straightforward.

Definition 4.1. Let $\mathtt{b}$ be a branch in a derivation and $\mathcal{M} = \langle S, R, \rho, v\rangle$ be a model. A *realization of* $\mathtt{b}$ *in* $\mathcal{M}$ is a mapping, $\mathtt{r} : C_{\mathtt{b}} \mapsto S$, satisfying:

- $\mathtt{r}(c_0) = \rho$;
- $\mathcal{M}, \mathtt{r}(c_i) \Vdash A$ for any ls-formula $\mathsf{T}\,A : c_i$ in $\mathtt{b}$;
- $\mathcal{M}, \mathtt{r}(c_i) \nVdash A$ for any ls-formula $\mathsf{F}\,A : c_i$ in $\mathtt{b}$;

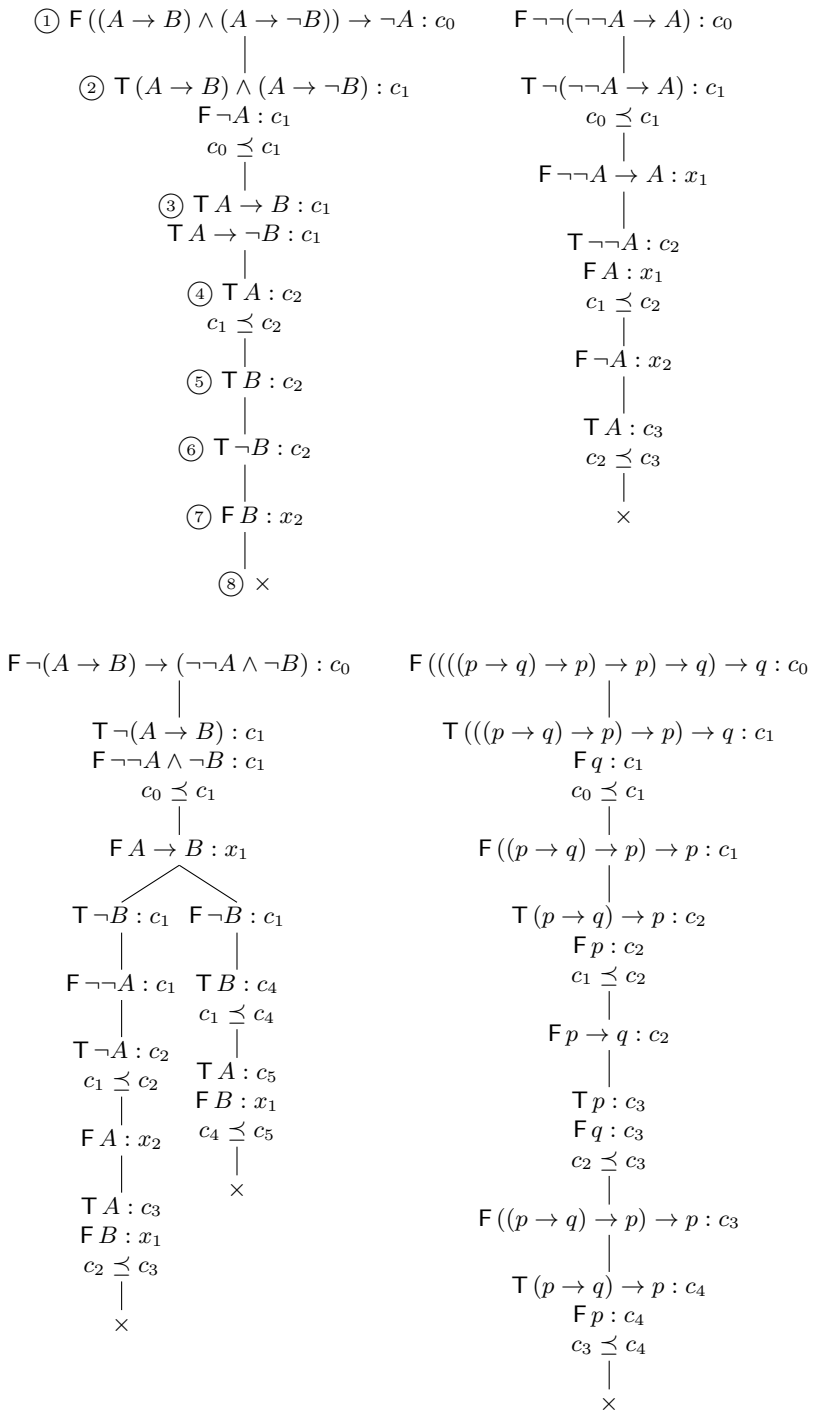


Figure 1: Some proofs (nodes are numbered in the first)

- $\mathcal{M}, \mathtt{r}(c_j) \nVdash A$ for any ls-formula $\mathsf{F}\, A : x_i$ in $\mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$;
- $\mathtt{r}(c_i) R \mathtt{r}(c_j)$ whenever $c_i \preceq_{\mathtt{b}} c_j$.

A branch $\mathtt{b}$ is *realizable* if there is a realization of it in some model. A derivation $\mathcal{T}$ is *realizable* if it contains a realizable branch.

Remark 4.2. For readability, henceforth we leave realization mappings implicit. That is, we write $\mathcal{M}, c_i \Vdash A$ instead of $\mathcal{M}, \mathtt{r}(c_i) \Vdash A$ (analogously for $\nVdash$), and $c_i R c_j$ instead of $\mathtt{r}(c_i) R \mathtt{r}(c_j)$.

Lemma 4.3. If a derivation $\mathcal{T}'$ is obtained from $\mathcal{T}$ by an application of an operational rule or *PB*, then, if $\mathcal{T}$ is realizable so is $\mathcal{T}'$.

Proof. Since $\mathcal{T}$ is realizable, it contains a branch $\mathtt{b}$ that is realizable in some model $\mathcal{M}$ for some realization $\mathtt{r}$. Now, if $\mathcal{T}'$ was obtained by a rule application extending a branch other than $\mathtt{b}$, then $\mathcal{T}'$ is realizable since it still contains $\mathtt{b}$. Otherwise, we show by case analysis that the operational rules and *PB* preserve realizability of derivations. Let us consider some illustrative cases, leaving the remaining ones to the reader.

- ($\mathsf{T}\neg$) $\mathtt{b}$ contains $\mathsf{T}\neg A : c_i$ and is expanded into $\mathtt{b}'$ with last node $\mathsf{F}\, A : x_i$. By assumption, $\mathtt{b}$ is realizable in a model $\mathcal{M}$ and so we have $\mathcal{M}, c_i \Vdash \neg A$. Thus, for all c_j such that $c_i \preceq_{\mathtt{b}} c_j$, we have $\mathcal{M}, c_j \nVdash A$. Thus, by Definition 4.1, $\mathtt{b}'$ is realizable.
- (*PB*) $\mathtt{b}$ is expanded into $\mathtt{b}'$ and $\mathtt{b}''$ with, respectively, last node $\mathsf{T}\, A : c_i$ and $\mathsf{F}\, A : c_i$. By definition of $\mathcal{M}$, either $\mathcal{M}, c_i \Vdash A$ or $\mathcal{M}, c_i \nVdash A$ for *any* A. So, by Definition 4.1, either $\mathtt{b}'$ or $\mathtt{b}''$ is realizable.
- ($\mathsf{F}\ \to_2$) $\mathtt{b}$ contains $\mathsf{F}\, A \to B : x_i$ and $c_i \preceq_{\mathtt{b}} c_j$, and it is expanded into $\mathtt{b}'$, adding the nodes $\mathsf{T}\, A : c_k$, $\mathsf{F}\, B : x_i$ and $c_j \preceq_{\mathtt{b}'} c_k$. Since $\mathtt{b}$ is realizable in $\mathcal{M}$, we have $\mathcal{M}, c_j \nVdash A \to B$ for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$. But this means that, for each such a c_j, there is a c_k such that $c_j \preceq_{\mathtt{b}'} c_k$, $\mathcal{M}, c_k \Vdash A$ and $\mathcal{M}, c_k \nVdash B$. By downward monotonicity, the latter implies only that $\mathcal{M}, c_i \nVdash B$. We are thus left to show that the condition for $\mathsf{F}\, B : x_i$ is satisfied. Assume, toward a contradiction, that there is c_l such that $c_i \preceq_{\mathtt{b}'} c_l$ and $\mathcal{M}, c_l \Vdash B$. Since $c_i \preceq_{\mathtt{b}'} c_l$, we would also have that $\mathcal{M}, c_l \nVdash A \to B$, and this implies in turn that there is a c_m such that $c_l \preceq_{\mathtt{b}'} c_m$ and $\mathcal{M}, c_m \Vdash A$ and $\mathcal{M}, c_m \nVdash B$. This is in contradiction with $\mathcal{M}$ being a model, for we have obtained $\mathcal{M}, c_l \Vdash B$ and $\mathcal{M}, c_m \nVdash B$ with $c_l \preceq_{\mathtt{b}'} c_m$. Hence we can only have $\mathcal{M}, c_l \nVdash B$ for each c_l such that $c_i \preceq_{\mathtt{b}'} c_l$. Therefore, the condition for $\mathsf{F}\, B : x_i$ is satisfied, and so $\mathtt{b}'$ is realizable.
- ($\mathsf{F}\wedge_3$) $\mathtt{b}$ contains $\mathsf{F}\, A \wedge B : x_i, \mathsf{T}\, A : c_j, c_i \preceq_{\mathtt{b}} c_j$, and is expanded into $\mathtt{b}'$, adding the node $\mathsf{F}\, B : x_j$. Given that $\mathtt{b}$ is realizable in $\mathcal{M}$, $\mathcal{M}, c_j \Vdash A$ and, for each c_k such that $c_j \preceq_{\mathtt{b}'} c_k$, we have that $\mathcal{M}, c_k \nVdash A \wedge B$. From the latter we have that either $\mathcal{M}, c_k \nVdash A$ or $\mathcal{M}, c_k \nVdash B$. On the other hand, since $c_j \preceq_{\mathtt{b}'} c_k$ and $\mathcal{M}, c_j \Vdash A$, by upward monotonicity, we also obtain $\mathcal{M}, c_k \Vdash A$. Thus we have that $\mathcal{M}, c_k \nVdash B$ for each c_k such that $c_j \preceq_{\mathtt{b}'} c_k$. But this shows that $\mathcal{M}$ realizes $\mathtt{b}'$, since it satisfies the condition for $\mathsf{F}\, B : x_j$.

□

Lemma 4.4. A closed derivation $\mathcal{T}$ cannot be realizable.

Proof. Suppose that $\mathcal{T}$ is closed and realizable. So, it contains a closed branch $\mathtt{b}$ which is realizable in some model $\mathcal{M}$. Given that $\mathtt{b}$ is closed, it contains either (1) a couple of ls-formulas $\mathsf{T}\,A : c_i$ and $\mathsf{F}\,A : c_j$ such that $c_i \preceq_{\mathtt{b}} c_j$, or (2) $\mathsf{T}\,A : c_i$ and $\mathsf{F}\,A : x_j$ such that both $c_i \preceq_{\mathtt{b}} c_k$ and $c_j \preceq_{\mathtt{b}} c_k$. If (1), by Definition 4.1, we have that $\mathcal{M}, c_i \Vdash A$ and $\mathcal{M}, c_j \nVdash A$ for some c_j such that $c_i R c_j$. But this is impossible due to upward monotonicity. If (2), again by Definition 4.1, we have $\mathcal{M}, c_i \Vdash A$ and $\mathcal{M}, c_l \nVdash A$ for any l such that $c_j R c_l$. By assumption, we have that there is c_k such that both $c_i R c_k$ and $c_j R c_k$. So, we have that $\mathcal{M}, c_k \nVdash A$ but also, again by upward monotonicity, that $\mathcal{M}, c_k \Vdash A$ which is impossible. □

As an immediate corollary, we obtain the following

Corollary 4.5. For any formula A, if there is a closed derivation $\mathcal{T}$ from $\mathsf{F}\,A : c_0$, then $\mathcal{M}, c_0 \Vdash A$ for all models $\mathcal{M}$.

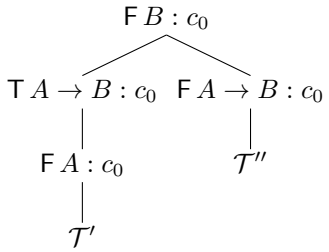
Proof. By Lemma 4.4, $\mathcal{T}$ is not realizable. This means that each branch of $\mathcal{T}$ is not realizable. On other hand, the root $\mathsf{F}\,A : c_0$ of $\mathcal{T}$ belongs to each of its branches. Hence for no model $\mathcal{M}$ we have that $\mathcal{M}, c_0 \nVdash A$, proving the thesis. □

4.2 Completeness via axiom system simulation

A straightforward way of proving completeness of our system is by showing that its set of theorems includes some complete set of axioms, and is closed under modus ponens. Given that we take negation as primitive, let us show this w.r.t. Heyting's original set of axioms.

Proposition 4.6. If A is derivable in Heyting's axiom system, then there is a closed derivation from $\mathsf{F}\,A : c_0$ in our KE tableaux.

Proof. Checking Heyting's 11 axioms for theoremhood in our system is routine. As an example, a proof of one of those axioms corresponds to the first derivation in Figure 1. Now, modus ponens can be easily simulated in our system as follows: suppose that there are closed derivations, $\mathcal{T}'$ and $\mathcal{T}''$, respectively from $\mathsf{F}\,A : c_0$ and $\mathsf{F}\,A \to B : c_0$. Then, the derivation



is also closed. □

The reader can confirm that completeness of the system without free-variables is easily proven via an almost identical simulation. In any case, note that these simulations do not preserve the subformula property. Namely, in the corresponding simulation of modus ponens, applying *PB* to formulas that are not subformulas of the theorem to be proved is essential. In the next section, we provide a more informative completeness proof for a slight variant of the system presented above.

This is achieved by means of a proof-search procedure which *terminates* either with a proof of the tested formula or with a finite countermodel of it, and where the applications of PB are restricted to subformulas of the tested formula.

5 Termination and countermodel generation

Following [4, 15], let us then consider a slight variant of our system in terms of which a terminating and complete procedure can be defined. Observe that only the rules $\mathsf{F}\,{\to_{1,2}}$ and $\mathsf{F}\,{\neg_{1,2}}$ introduce new constants and so, when applied unrestrictedly, can yield infinitely long branches. As a first countermeasure to avoid this problem in proof-search, we impose that each rule can be applied at most once for each particular choice of ls-formulas as premises. This means that, once we apply $\mathsf{F}\,{\to_1}$ to, say $\mathsf{F}\,A \to B : c_i$, then we can only apply such a rule, either on a formula different from $A \to B$, or to the same formula but with a label different from c_i.[3]

Note that this measure also guarantees finite width of the putative countermodel in the underlying semantics. That is, guarantees that, for each state denoted by c_i in that model, c_i has finitely many immediate successors that are incomparable with each other by $\preceq$. To wit, assume that a label c_i is introduced in our system. Then we can only generate finitely many c^i_js such that, for all $0 \leq j \leq n$: (a) $c_i \preceq c^i_j$, there is no $c^i_k \neq c^i_j$ such that $c_i \preceq c^i_k \preceq c^i_j$, and (b) neither $c^i_{j_1} \preceq c^i_{j_2}$ nor $c^i_{j_2} \preceq c^i_{j_1}$ for all $0 \leq j_1 \neq j_2 \leq n$.[4] Indeed, such constants c^i_js may only be introduced from applications of rules $\mathsf{F}\,{\to_{1,2}}$ or $\mathsf{F}\,{\neg_{1,2}}$ on subformulas of the initial formula, with label c_i or x_i, and those are finitely many.

Limiting rule application does not however suffice for avoiding the creation of infinitely long branches in derivations. These may still be produced when a pair of ls-formulas of the form $\mathsf{T}\,(A \to B) \to B : c_i$ and $\mathsf{F}\,A \to B : l_j$, with $c_i \preceq_{\mathtt{b}} c_j$, occur in a branch. An example of this situation is displayed in Figure 2 [cf. 15].[5] We can obtain however a terminating procedure by imposing a suitable proviso on the rule $\mathsf{F}\,{\to_1}$ and adding the rule $\mathsf{F}\,{\to_3}$ below, that allow us to *reuse existing constants* [see 15, 4].

Recall the rule $\mathsf{F}\,{\to_1}$:

$$\dfrac{\begin{array}{l}\mathsf{F}\,A \to B : c_i\\ \hline \mathsf{T}\,A : c_j\\ \mathsf{F}\,B : c_j\\ c_i \preceq c_j\end{array}}{\begin{array}{l}c_j \text{ new}\\ {\scriptstyle(\mathsf{F}\,\to_1)}\end{array}}$$

We now impose the *proviso* that $\mathsf{F}\,{\to_1}$ is applicable only when $\mathsf{T}\,A : c_h$ does not occur for any $c_h \preceq_{\mathtt{b}} c_i$ in the branch $\mathtt{b}$ where the rule is to be applied. In such a case, we would already know—by upward monotonicity of the underlying

[3] This is taken care of in our Algorithm 1, where the set `rinst` keeps a record of used rule instances.

[4] For instance, each state of the countermodel of Example 5.15 below has finitely many immediate and incomparable successors. In particular, for c_1, these are: c_2, c_3 and c_4.

[5] Note that infinitely long branches, as that displayed in Figure 2, arise when dealing with formulas that are *non*-theorems. Indeed, when dealing with theorems, our first countermeasure suffices to guarantee finiteness of branch length: any branch eventually closes. In particular, the proviso on $\mathsf{F}\,{\to_1}$ and the rule $\mathsf{F}\,{\to_3}$ to be introduced next are not required, as illustrated by the fourth example in Figure 1.

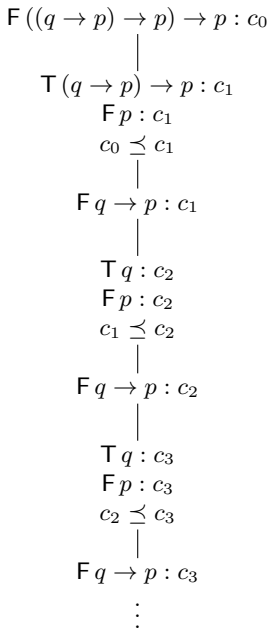


Figure 2: A derivation with an infinite branch

semantics—that A is proven at the state denoted by c_i and there would be no need to create a new constant by applying $\mathsf{F}\ \to_1$. We then add the following rule:

$$\frac{\mathsf{F}\, A \to B : c_j \quad \mathsf{T}\, A : c_i \quad \boxed{c_i \preceq c_j}}{\mathsf{F}\, B : c_j}\ (\mathsf{F}\ \to_3)$$

Lemma 5.1. The rule $\mathsf{F}\ \to_3$ is sound.

Proof. Let $\mathcal{M}$ be a model. By assumption, $\mathcal{M}, c_j \nVdash A \to B$, $\mathcal{M}, c_i \Vdash A$, and $c_i R c_j$. By upward monotonicity, it follows that $\mathcal{M}, c_j \Vdash A$. Now, the first assumption implies that there is c_k such that $c_j R c_k$, $\mathcal{M}, c_k \Vdash A$ and $\mathcal{M}, c_k \nVdash B$ which, by downward monotonicity, implies $\mathcal{M}, c_j \nVdash B$. □

Remark 5.2. As discussed in Theorem 5.9 below, the added proviso on $\mathsf{F}\ \to_1$ and the rule $\mathsf{F}\ \to_3$ result in producing *at most one* new constant for each ls-formula of the form $\mathsf{F}\, A \to B : c_i$ that may be obtained in a derivation from the initial formula. Note that, under the proviso on $\mathsf{F}\ \to_1$, the applicability of this rule and that of $\mathsf{F}\ \to_3$ are mutually exclusive. Indeed, $\mathsf{F}\ \to_3$ is *not* a derivable rule in the terminating system's variant resulting of adding this rule together with the proviso on $\mathsf{F}\ \to_1$. Namely, trying to simulate $\mathsf{F}\ \to_3$ would require a violation of that proviso.

A terminating decision procedure for proof-search or countermodel construction can now be introduced, based on the system with the added rule $\mathsf{F}\ \to_3$ and the

proviso on $\mathsf{F} \to_1$. Before that, we need the following notions, that will be handy in determining when the proof-search procedure can be considered completed.

Definition 5.3 (see [4, 15])**.** Let $\mathcal{T}$ be a derivation and $\mathtt{b}$ one of its branches. We define recursively when an ls-formula φ is *completely analyzed* ("c.a." for short) in $\mathtt{b}$, according to φ's form:

- $\mathsf{T}\, p : c_j$ is c.a. in $\mathtt{b}$ iff there is $\mathsf{T}\, p : c_i \in \mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$.
- $\mathsf{F}\, p : c_i$ is c.a. in $\mathtt{b}$ iff there is $\mathsf{F}\, p : c_j \in \mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$.
- $\mathsf{F}\, p : x_j$ is c.a. in $\mathtt{b}$ iff either there is $\mathsf{F}\, p : x_i \in \mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$ or for each $c_k \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_k$, $\mathsf{F}\, p : c_k \in \mathtt{b}$.
- $\mathsf{F}\, A \vee B : c_i$ is c.a. in $\mathtt{b}$ iff both $\mathsf{F}\, A : c_i$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, B : c_i$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, A \vee B : x_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, both $\mathsf{F}\, A : c_j$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, B : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{T}\, A \wedge B : c_i$ is c.a. in $\mathtt{b}$ iff both $\mathsf{T}\, A : c_i$ is c.a. in $\mathtt{b}$ and $\mathsf{T}\, B : c_i$ is c.a. in $\mathtt{b}$.
- $\mathsf{T}\, A \vee B : c_i$ is c.a. in $\mathtt{b}$ iff $\mathsf{T}\, A : c_i$ is c.a. in $\mathtt{b}$ or $\mathsf{T}\, B : c_i$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, A \wedge B : c_i$ is c.a. in $\mathtt{b}$ iff $\mathsf{F}\, A : c_i$ is c.a. in $\mathtt{b}$ or $\mathsf{F}\, B : c_i$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, A \wedge B : x_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, $\mathsf{F}\, A : c_j$ is c.a. in $\mathtt{b}$ or $\mathsf{F}\, B : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{T}\, A \to B : c_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, $\mathsf{F}\, A : c_j$ is c.a. in $\mathtt{b}$ or $\mathsf{T}\, B : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, A \to B : c_i$ is c.a. in $\mathtt{b}$ iff there is a $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, and both $\mathsf{T}\, A : c_j$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, B : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, A \to B : x_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, there is a $c_k \in C_{\mathtt{b}}$ such that $c_j \preceq_{\mathtt{b}} c_k$, and both $\mathsf{T}\, A : c_k$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, B : c_k$ is c.a. in $\mathtt{b}$.
- $\mathsf{T}\, \neg A : c_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, $\mathsf{F}\, A : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, \neg A : c_i$ is c.a. in $\mathtt{b}$ iff there is $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$ and $\mathsf{T}\, A : c_j$ is c.a. in $\mathtt{b}$.
- $\mathsf{F}\, \neg A : x_i$ is c.a. in $\mathtt{b}$ iff for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$, there is a $c_k \in C_{\mathtt{b}}$ such that $c_j \preceq_{\mathtt{b}} c_k$ and $\mathsf{T}\, A : c_k$ is c.a. in $\mathtt{b}$.

In turn, a branch $\mathtt{b}$ in a derivation $\mathcal{T}$ is said to be *completed* if it is open and every ls-formula $\varphi \in \mathtt{b}$ is completely analyzed. Any ls-formula belonging to a non-completed branch $\mathtt{b}$ for which the corresponding condition above fails is said to be *not completely analyzed* in $\mathtt{b}$. The term "completed" is the usual one in tableau-like systems [e.g., 4, 15], but "saturated" is also commonly used [e.g., 3, 9].[6]

[6]Note that some authors use "completed" as *not* implying that the branch is open, keeping those notions separated. Here, as in [15], we use the term as implying branch openness.

Remark 5.4. Note that, in contrast to $\mathsf{F}\ \rightarrow_1$, a variant of the rule $\mathsf{F}\ \rightarrow_2$ is not needed for guaranteeing branch length finiteness. Recall indeed Definition 5.3, in particular for what concerns ls-formulas of the form $\mathsf{F}\, A \rightarrow B : x_i$. Once such a formula is encountered in proof-search, it suffices to apply the rule $\mathsf{F}\ \rightarrow_2$ *once* for each immediate successor c_j of c_i, in order to check that it is completely analyzed. After all the immediate successors have been considered, there will be no need for further application of $\mathsf{F}\ \rightarrow_2$ to $\mathsf{F}\, A \rightarrow B : x_i$, thus discarding the possibility of producing an infinite branch. Note in turn that, as discussed above, the width of the putative countermodel is guaranteed to be finite in that the immediate successors of c_i are finitely many.

Intuitively, a branch is completed when no rule from Table 1 ($\mathsf{F}\ \rightarrow_1$ under the proviso) nor $\mathsf{F}\ \rightarrow_3$ is applicable on this branch for decomposing an ls-formula and appending a new one whose respective constituent formula is of lesser degree, and none of the closure rules was applied along it. This idea gives support to the next definition.

Definitions 5.5.

- A derivation $\mathcal{T}$ from φ is *completed* if it has a completed branch.
- A *refutation* of a formula A is a completed derivation from $\mathsf{F}\, A : c_0$.[7]

Some examples of *refutations* are displayed in Figure 3 below. The first refutation corresponds to the failure of double negation elimination. Note that, in this example, the branch is not closed, since we have $\mathsf{T}\, A : c_2$ and $\mathsf{F}\, A : c_1$ with $c_1 \preceq c_2$, while the respective closure rule requires, on the contrary, that a formula signed T holds at a predecessor of the state where the same formula signed F holds. The second example shows the invalidity of Peirce law. The third particularly illustrates our notion of completed branch, as implementing the proviso on $\mathsf{F}\ \rightarrow_1$ and the rule $\mathsf{F}\ \rightarrow_3$. Contrast it with the situation exemplified in Figure 2. Finally, the last example in Figure 3 is a refutation of the well-known Scott axiom.

We are now in a position to introduce our decision procedure, which is analogous to the "canonical" procedure for classical KE given in [6]. The adjective "canonical" for these procedures refers to the fact that—even if they involve some non-determinism by making room for some choices—they are "as mechanical as" the respective procedures based on standard tableaux, given that their applications of the corresponding PB rule are restricted to subformulas of the initial formula tested for theoremhood.

Definition 5.6. The *canonical procedure* for a formula A starts from the one-node tree consisting of the ls-formula $\mathsf{F}\, A : c_0$ and applies the rules in Table 1, prohibiting repeated rule instance applications and adding the proviso for $\mathsf{F}\ \rightarrow_1$ and the rule $\mathsf{F}\ \rightarrow_3$, until the resulting derivation $\mathcal{T}$ is either closed or completed. This procedure is described in Algorithm 1.

To illustrate the functioning of Algorithm 1, a few clarifications are in place. First, note that we abbreviate one-premise and two-premise rules with 1-rule and 2-rule respectively. Given a rule instance r, we denote by $maj(r)$ the major premise of r (that is, the only premise in case of a 1-rule), and by $min(r)$ the minor premise of r (in case of a 2-rule). We further denote by $conc(r)$ the conclusions of r.

[7] As explained below, in our system, a refutation is associated with a *single* model invalidating the corresponding initial formula, i.e. a countermodel of it.

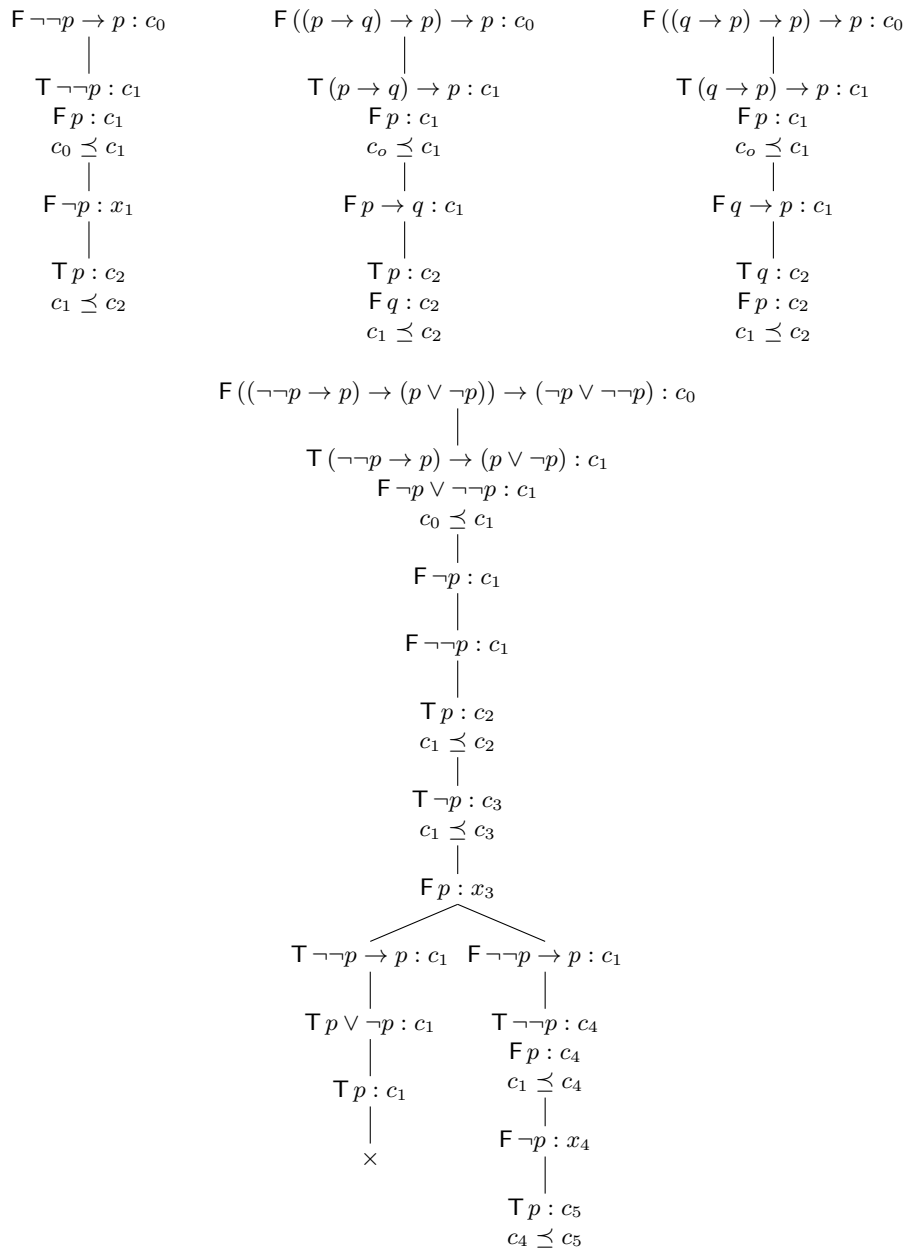


Figure 3: Some refutations

Algorithm 1 Canonical procedure

Input: a formula A

1: $\mathcal{T} \leftarrow \mathsf{F}\, A : c_0$
2: **while** $\mathcal{T}$ is neither closed nor completed **do**
3: select an open branch $\mathtt{b}$ in $\mathcal{T}$
4: $\mathtt{rinst} \leftarrow \emptyset$
5: **while** $\mathtt{b}$ is neither closed nor completed **do**
6: select a φ in $\mathtt{b}$ which is not completely analyzed
7: select a rule r such that $\varphi = maj(r)$ **and** $r \notin \mathtt{rinst}$
8: **if** r is a 2-rule and $min(r) \notin \mathtt{b}$ **then**
9: $\mathcal{T} \leftarrow (\mathcal{T}$ expanded with a new branch $\mathtt{b} \cup \overline{min(r)})$
10: $\mathtt{b} \leftarrow (\mathtt{b}$ expanded with $min(r))$
11: **end if**
12: $\mathtt{b} \leftarrow (\mathtt{b}$ expanded with $conc(r))$
13: $\mathtt{rinst} \leftarrow \mathtt{rinst} \cup \{r\}$
14: **end while**
15: **end while**
16: **return** $\mathcal{T}$

Second, we use the set `rinst` to keep track of the rule application instances that have been performed. Each rule in `rinst` has be to represented collecting major, minor premise, constant constraints and conclusions. Keeping track of `rinst` guarantees that each application of a rule over a certain major premise uses at least a different minor premise or different constant constraints thus avoiding that the algorithm goes into an endless loop.

Third, note that we are applying *PB* in lines 10 and 11. The innermost while loop there produces two branches out of the current branch `b`: one containing the minor premise of the rule and the other containing the conjugate of the minor premise. While we update the tree with both branches, we update the current branch `b` only with the minor premise of the 2-rule, and we further proceed to the application of the 2-rule.

Remark 5.7. Note that all proofs but the last of Figure 1 are constructed following the canonical procedure. Modifying this proof so that it follows the procedure is routine and we leave it to the reader.[8] As for the refutations of Figure 3, all follow the procedure.

Lemma 5.8. The canonical procedure produces only finitely many constants.

Proof. The proviso on $\mathsf{F} \to_1$ and the use of the rule $\mathsf{F} \to_3$ guarantee that, applying each rule instance once, we obtain a bound on constant generation. The argument

[8] We obtain $\mathsf{F}\, p : c_3$ from $\mathsf{F}\,((p \to q) \to p) \to p : c_3$ and $\mathsf{T}\,(p \to q) \to p : c_2$, and thus closure is obtained from $\mathsf{T}\, p : c_3$ and $\mathsf{F}\, p : c_3$. Note that, as exemplified by the third proof of Figure 1, unlike the analogous procedure for classical KE in [6], our canonical procedure does not necessarily push *PB* applications down. As shown in Theorem 5.9 below, pushing down that rule is not necessary for termination. Analogously to classical KE, however, further or less control on *PB* might dramatically affect proof-size and proof-search complexity.

for this is essentially the same given for standard tableaux in [15]. Namely, a *single* application of the rule $\mathsf{F}\ \to_1$ to an ls-formula of the form $\mathsf{F}\, A \to B : c_i$ adds an ls-formula of the form $\mathsf{T}\, A : c_j$, with $c_i \preceq_b c_j$, to $\mathtt{b}$. Once this latter ls-formula is added to $\mathtt{b}$, the canonical procedure can only apply the rule $\mathsf{F}\ \to_3$ every time an ls-formula of the form $\mathsf{F}\, A \to B : c_k$ with $c_j \preceq_{\mathtt{b}} c_k$ is to be analyzed. Indeed, $\mathsf{F}\ \to_3$ generates no further constants but reuses previously introduced ones. Hence the interaction of the proviso on $\mathsf{F}\ \to_1$ and $\mathsf{F}\ \to_3$ avoids the generation of infinitely many constants. A similar argument holds for $\mathsf{F}\ \to_2$, but without the need of a proviso nor additional rules, as pointed out in Remark 5.4 above. □

Theorem 5.9. *The canonical procedure terminates.*

Proof. Let us start from the innermost while loop. Each open branch $\mathtt{b}$ eventually gets either closed or completed by applying to each $\varphi \in \mathtt{b}$ all the rule instances having φ as a major premise (possibly together with a minor premise and constant constraints) with no repeated instances. To see this, first note that any elimination rule appends to $\mathtt{b}$ only subformulas (ignoring signs and labels) of the constituent formula of the major premise to which it is applied. Whereas, in case a major premise requires the use of a 2-rule with a non-present minor premise, the latter is added in the branch by an application of *PB*, and the 2-rule is afterwards applied. So, in any case, each rule application attaches to $\mathtt{b}$ only subformulas of the constituent formula of the major premise to which it has been applied. As the instance rules are applied, subformulas of increasingly lower degree are obtained until either branch closure or propositional variables are hit. In turn, ls-formulas of the form $\mathsf{T}\, p : c_i$ or $\mathsf{F}\, p : l_i$ that belong to $\mathtt{b}$ are, by default, completely analyzed in $\mathtt{b}$ and closed under up- and downward monotonicity as mimicked by the signs and $\preceq_{\mathtt{b}}$. Thus, the set of ls-formulas that must be considered for rule instance applications along a branch is partly bounded by the set of subformulas of the initial formula tested for theoremhood. Note, however, that the introduction of new constants may turn previously completely analyzed ls-formulas into unanalyzed ones. Nonetheless, the number of new constants is finite by Lemma 5.8 and the algorithm will hence have to go back to the same ls-formula for applying a new rule instance only a finite number of times, completing the bound on the set of ls-formulas that must be considered for rule applications. By these considerations, the innermost while loop, on $\mathtt{b}$, terminates.

Furthermore, new branches are produced only by applications of *PB* which, as pointed out above, are only applied to subformulas of the formulas in a branch. Hence the number of branches obtained is also bounded, and the outermost while loop, on $\mathcal{T}$, terminates.

□

Remark 5.10. A constructive proof of the subformula property of our system—i.e. one giving a procedure for transforming any closed or completed derivation into an equivalent one satisfying the property—can be given along the lines of [5, Proposition 10.1].

Example 5.11. Let us illustrate how the procedure works by following closely the steps performed in the third refutation in Figure 3, until we obtain a completed branch. Let us start from the tree with the unique node $\mathsf{F}\, ((q \to p) \to p) \to p : c_0$. That ls-formula is not completely analyzed, since the corresponding condition in Definition 5.3 clearly fails. Moreover, $\mathsf{F}\, ((q \to p) \to p) \to p : c_0$ is a major premise of

the one-premise rule $\mathsf{F}\ \to_1$, the branch satisfies the proviso on $\mathsf{F}\ \to_1$, and $\mathtt{rinst} = \emptyset$ w.r.t. $\mathtt{b}$. Thus, the algorithm extends the tree and its unique branch $\mathtt{b}$ with the conclusion of the rule $\mathsf{F}\ \to_1$, i.e. $\mathsf{T}\,(q \to p) \to p : c_1$ and $\mathsf{F}\,p : c_1$, with $c_0 \preceq_{\mathtt{b}} c_1$. This application instance of $\mathsf{F}\ \to_1$ is added to $\mathtt{rinst}$.

In the new iteration, $\mathtt{b}$ is not completed and in particular the formula $\mathsf{T}\,(q \to p) \to p : c_1$ is not completely analyzed, since neither $\mathsf{F}\,(q \to p) : c_1$ nor $\mathsf{T}\,p : c_1$ are completely analyzed in $\mathtt{b}$. The algorithm thus picks $\mathsf{T}\,(q \to p) \to p : c_1$ and, since this is a major premise of a two-premise rule $\mathsf{T}\ \to_2$, and a corresponding minor premise $\mathsf{F}\,p : c_1$ is in $\mathtt{b}$, it extends $\mathtt{b}$ with the conclusion of that rule instance. This instance is added to $\mathtt{rinst}$ and the branch $\mathtt{b}$ is extended with the ls-formula $\mathsf{F}\,q \to p : c_1$.

In the next iteration, $\mathtt{b}$ again fails to be completed, due to the formula $\mathsf{F}\,q \to p : c_1$ that is not completely analyzed. Hence the algorithm picks such formula up. Note that $\mathsf{F}\,q \to p : c_1$ is the major premise of the one-premise rule $\mathsf{F}\ \to_1$ and the proviso on the latter is satisfied. Thus an instance of $\mathsf{F}\ \to_1$ is added to $\mathtt{rinst}$ and the conclusions $\mathsf{T}\,q : c_2$ and $\mathsf{F}\,p : c_2$ with $c_1 \preceq_{\mathtt{b}} c_2$ are added to the branch $\mathtt{b}$.

Finally, observe that $\mathsf{T}\,(q \to p) \to p : c_1$ is completely analyzed under the addition of the new label c_2, $c_1 \preceq_{\mathtt{b}} c_2$, in the previous iteration, since $\mathsf{F}\,(q \to p) : c_2$ is completely analyzed in turn. Thus the branch $\mathtt{b}$ is finally completed, and the algorithm terminates. □

We conclude this section by showing that, when the canonical procedure does not terminate with a proof, it is possible to extract a finite countermodel from a completed branch obtained by its execution.

Definition 5.12. Let $\mathtt{b}$ be a completed branch. The *canonical model* associated with $\mathtt{b}$, denoted $\mathcal{M}_{\mathtt{b}}$, is the tuple $\langle C_{\mathtt{b}}, \preceq_{\mathtt{b}}, c_0, v_{\mathtt{b}} \rangle$, where $C_{\mathtt{b}}$, $\preceq_{\mathtt{b}}$ and c_0 are defined as in Definition 3.1, while $v_{\mathtt{b}}$ is defined by the following:

$$v_{\mathtt{b}}(p) = \{\, c_i \in C_{\mathtt{b}} \mid \text{ there is } c_h \in C_{\mathtt{b}} \text{ such that } c_h \preceq_{\mathtt{b}} c_i \text{ and } \mathsf{T}\,p : c_h \in \mathtt{b} \,\}.$$

Remark 5.13. Note that, according to the previous definition, only a finite number of propositional variables can satisfy $v_{\mathtt{b}}(p) \neq \emptyset$. Since the set $\mathcal{P}$ of propositional variables is countable, there must be an infinite subset of propositional variables q such that $v_{\mathtt{b}}(q) = \emptyset$.

Lemma 5.14. Let $\mathtt{b}$ be a completed branch rooted in $\mathsf{F}\,A : c_0$ and $\mathcal{M}_{\mathtt{b}}$ the canonical model associated with it. Then $\mathcal{M}_{\mathtt{b}}, c_0 \nVdash A$.

Proof. To prove the claim, we first show the following

- $\mathcal{M}_{\mathtt{b}}, c_i \Vdash B$ for any ls-formula $\mathsf{T}\,B : c_i$ that is c.a. in $\mathtt{b}$;
- $\mathcal{M}_{\mathtt{b}}, c_i \nVdash B$ for any ls-formula $\mathsf{F}\,B : c_i$ that is c.a. in $\mathtt{b}$;
- $\mathcal{M}_{\mathtt{b}}, c_j \nVdash B$ for any ls-formula $\mathsf{F}\,B : x_i$ that is c.a. in $\mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$.

In particular the claim would then follow from the fact that $\mathsf{F}\,A : c_0$ is c.a. in $\mathtt{b}$.

We proceed to show the above by induction on the degree of the formulas. For the base case, assume $\mathsf{T}\,p : c_i$ is c.a. in $\mathtt{b}$, that is, there is $\mathsf{T}\,p : c_h \in \mathtt{b}$ such that $c_h \preceq_{\mathtt{b}} c_i$. Then, by the definition of $\mathcal{M}_{\mathtt{b}}$, we obtain $\mathcal{M}_{\mathtt{b}}, c_i \Vdash p$. Consider now $\mathsf{F}\,p : c_i$ is c.a. in $\mathtt{b}$. So, there is $\mathsf{F}\,p : c_j \in \mathtt{b}$ such that $c_i \preceq_{\mathtt{b}} c_j$. By the assumption

that $\mathtt{b}$ is open, there is no $c_h \preceq_{\mathtt{b}} c_j$ such that $\mathsf{T}\, p : c_h \in \mathtt{b}$ and so, in particular, for all $c_k \preceq_{\mathtt{b}} c_i$, $\mathsf{T}\, p : c_k \notin \mathtt{b}$. Hence, $\mathcal{M}_{\mathtt{b}}, c_i \nVdash p$ by the definition of $\mathcal{M}_{\mathtt{b}}$. Finally, consider the case where $\mathsf{F}\, p : x_i$ is c.a. in $\mathtt{b}$. That is, there is $\mathsf{F}\, p : x_h \in \mathtt{b}$ such that $c_h \preceq_{\mathtt{b}} c_i$ or for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$, $\mathsf{F}\, p : c_j \in \mathtt{b}$. In the first case we have that, by the assumption that $\mathtt{b}$ is open, for each c_k such that $c_h \preceq_{\mathtt{b}} c_k$, and for all c_l such that $c_l \preceq_{\mathtt{b}} c_k$, $\mathsf{T}\, p : c_l \notin \mathtt{b}$. So, in particular, for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$, we have that for all $c_m \preceq_{\mathtt{b}} c_j$, $\mathsf{T}\, p : c_m \notin \mathtt{b}$. Thus, by definition of $\mathcal{M}_{\mathtt{b}}$, $\mathcal{M}_{\mathtt{b}}, c_j \nVdash p$, for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$. The second case is similar.

For the inductive step, we consider some of the subcases, leaving the remaining ones to the reader.

Assume $\mathsf{F}\, C \to D : c_i$ is c.a. in $\mathtt{b}$. By definition, there is c_j such that $c_i \preceq_{\mathtt{b}} c_j$, and both $\mathsf{T}\, C : c_j$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, D : c_j$ is c.a. in $\mathtt{b}$. So, by induction hypothesis, $\mathcal{M}_{\mathtt{b}}, c_j \Vdash C$ and $\mathcal{M}_{\mathtt{b}}, c_j \nVdash D$, thus proving the inductive step, i.e. $\mathcal{M}_{\mathtt{b}}, c_i \nVdash C \to D$.

Consider now the case where $\mathsf{F}\, C \to D : x_i$ is c.a. in $\mathtt{b}$. Hence, for each c_j such that $c_i \preceq_b c_j$, there is a c_k such that $c_j \preceq_{\mathtt{b}} c_k$ and both $\mathsf{T}\, C : c_k$ is c.a. in $\mathtt{b}$ and $\mathsf{F}\, D : c_k$ is c.a. in $\mathtt{b}$. As before, by induction hypothesis, $\mathcal{M}_{\mathtt{b}}, c_k \Vdash C$ and $\mathcal{M}_{\mathtt{b}}, c_k \nVdash D$, showing that $\mathcal{M}_{\mathtt{b}}, c_j \nVdash C \to D$ for each $c_j \in C_{\mathtt{b}}$ such that $c_i \preceq_{\mathtt{b}} c_j$.

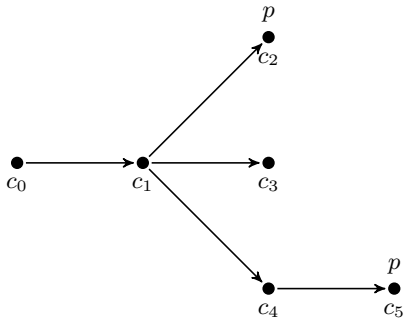
Let us now assume $\mathsf{T}\, C \to D : c_i$ is c.a. in $\mathtt{b}$. Thus, for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$, $\mathsf{F}\, C : c_j$ is c.a. in $\mathtt{b}$ or $\mathsf{T}\, D : c_j$ is c.a. in $\mathtt{b}$. By induction hypothesis this entails $\mathcal{M}_{\mathtt{b}}, c_j \nVdash C$ or $\mathcal{M}_{\mathtt{b}}, c_j \Vdash D$. Since this holds for each c_j such that $c_i \preceq_{\mathtt{b}} c_j$, we obtain $\mathcal{M}_{\mathtt{b}}, c_i \Vdash C \to D$.

Now, consider $\mathsf{F}\, C \wedge D : c_i$ is c.a. in $\mathtt{b}$. Hence $\mathsf{F}\, C : c_i$ is c.a. in $\mathtt{b}$ or $\mathsf{F}\, D : c_i$ is c.a. in $\mathtt{b}$. By the induction hypothesis, $\mathcal{M}_{\mathtt{b}}, c_i \nVdash C$ or $\mathcal{M}_{\mathtt{b}}, c_i \nVdash D$, thus proving the inductive step, i.e. $\mathcal{M}_{\mathtt{b}}, c_i \nVdash C \wedge D$.

Let us finally consider the case of $\mathsf{F}\, \neg C : c_i$ c.a. in $\mathtt{b}$. We get that $\mathsf{T}\, C : c_j$ is c.a. in $\mathtt{b}$, for some c_j such that $c_i \preceq_{\mathtt{b}} c_j$. By induction hypothesis, $\mathcal{M}_{\mathtt{b}}, c_j \Vdash C$, thus proving the inductive step that $\mathcal{M}_{\mathtt{b}}, c_i \nVdash \neg C$.

□

Example 5.15. Coming back to the fourth refutation of Figure 3, Lemma 5.14 allows us to extract the following countermodel of the Scott axiom out of the right open branch:



Where, as usual [see 1, 15], states are depicted as points, an arrow from an state a to b is drawn if aRb (omitting arrows that can be obtained by reflexivity and transitivity), and atomic formulas are depicted next to a state if they are forced in it. More specifically, the diagram above depicts the model $\mathcal{M}_{\mathtt{b}} = \langle C_{\mathtt{b}}, \preceq_{\mathtt{b}}, c_0, v_{\mathtt{b}} \rangle$, where

$$C_{\mathtt{b}} = \{c_0, c_1, c_2, c_3, c_4, c_5\};$$

$\preceq_{\mathfrak{b}} = \mathrm{Id}_{C_{\mathfrak{b}}} \cup \{\langle c_0, c_i\rangle | c_i \in C_{\mathfrak{b}}\} \cup \{\langle c_1, c_2\rangle, \langle c_1, c_3\rangle, \langle c_1, c_4\rangle, \langle c_4, c_5\rangle, \langle c_1, c_5\rangle\}$;

$v_{\mathfrak{b}}(p) = \{c_2, c_5\}$ and $v_{\mathfrak{b}}(q) = \emptyset$ for all q different from p.

The reader can verify that $\mathcal{M}_{\mathfrak{b}}, c_0 \nVdash ((\neg\neg p \to p) \to (p \vee \neg p)) \to (\neg p \vee \neg\neg p)$.[9]

We thereby obtain a more informative completeness proof for the KE-style system endowed with the rule $\mathsf{F} \to_3$ and the respective proviso for $\mathsf{F} \to_1$.

Theorem 5.16. *The canonical procedure is complete.*

Proof. We show the counterpositive. Suppose that the procedure terminates with no proof of A, i.e. there is no closed derivation from $\mathsf{F}\, A : c_0$. Then, by Theorem 5.9, the canonical procedure terminates with a refutation, i.e. a derivation that contains a completed branch. From this branch, we extract, using Lemma 5.14, a countermodel where A is not forced. Therefore, A is not valid. □

6 Broad comparisons with other systems

In this final section, we broadly compare our system with some popular sequent-like and tableau systems mentioned in the introduction. There, we pointed out some properties that are important for a system in the context of automated deduction. Our broad analysis here is based on whether or not a system satisfies those properties, considering systems that at least enjoy termination.

6.1 Subformula property

Roughly, the subformula property of a system ensures that every formula appearing in the conclusion of a derivation step is a subformula of some formula occurring in the premises of that step. This property provides neat arguments of termination, since it ensures that applying a rule—bottom-up, in the case of sequent-like systems—strictly decreases the size of the formula or sequent (i.e., the number of occurrences of symbols in it) to which it is applied. Then, given that the initial formula tested for theoremhood has finite size, derivation finiteness and termination follow.

However, termination can be proven quite independently of this property. Indeed, Miglioli's tableau system [18], G3I$_t$ [9], LJT [8] as well as its variants LJT$_{\mathtt{SAT}}$ [13] and $C^{\to}$ [11], all terminate but lack the property. In particular, they have a rule or set of rules that violate the property and provide strategies showing that, despite that lack, termination is guaranteed. Those strategies differ in details but their common idea is roughly that of rewriting each implicative formula into intuitionistically equivalent formula(s) that is (are) "simpler" as regards implications, and thus obtaining rules decreasing a certain measure of formula size. Then, with a suitable measure of the degree of formulas, it can be guaranteed that derivations in each of these systems have finite length. Nonetheless, these arguments are less direct and elegant than those based on the satisfaction of the subformula property.

[9] This is not a "minimal size" countermodel of the Scott axioms w.r.t. the number of states, for c_0 and c_2 can be dispensed with. Producing minimal size countermodels in our system requires modifications that we leave for future work.

In these terms, systems satisfying the property, such as Gbu [10], TBI/IL [15], IG, SIC [3], and ours, are preferable. We emphasize that, as observed in Definition 3.1 and Theorem 5.9, our system satisfies the *standard* subformula property, which holds at the level of *simple* formulas. Our signs and labels play a crucial role in showing termination but keep the property as it is usually stated. Indeed, although—as regards labels mechanism—the termination argument for $\mathrm{G3I}_t$ is analogous to ours, $\mathrm{G3I}_t$'s rule $R \to_t$ demands a relaxation of the subformula property which we recall in Subsection 6.3 below.

6.2 Countermodel generation

As it is well-known, intutionistic propositional logic enjoys the finite model property. That is, for any non-theorem, there is a finite countermodel witnessing its non-validity. Indeed, these witnesses are more informative and compact than a plethora of proof failures. In our system, finite countermodels in the relational semantics can be directly extracted from a *single* completed (i.e. saturated) branch, without any other additional device such as a complementary dual system or model-gluing.

By contrast, for example, given the non-invertibility of some of its rules, both LJT and Gbu [10] require a dual system for refutation-search, called CRIP [20] and Rbu respectively. Yet, given that those refutation systems still contain some non-invertible rules, a single completed branch in a derivation therein does not contain enough information to build a countermodel, but a completed tree is required. Then, the partial information contained in the branches must be "glued together" to build a single countermodel refuting the initial sequent.

Exploiting the expressive power of labels, TBI/IL, $\mathrm{G3I}_t$, and our KE system do not require a dual refutation system. These labeled systems are actually designed so as to closely follow countermodel-search in the relational semantics. Particularly, roof-first proof-search in $\mathrm{G3I}_t$ works fine since all its rules are invertible: for each rule, a countermodel of *one* of the premises is also a countermodel of the conclusion. Indeed, depth-first search is terminating in these three systems, so we can apply the rules until every branch is closed or a completed branch is obtained. In the case of a proof, the whole derivation tree must be completed but, for a countermodel, the search can be stopped as soon as a completed branch has been obtained. Namely, a countermodel can be directly extracted from a single completed branch without any extra recomposition. Such a branch contains enough information to build a countermodel and, given that the labels mimic the relational semantics, all that information is readily readable from it.

6.3 Simulations and proof-size

It is not difficult to show that our KE system can simulate all the rules of the tableaux [18, 15] and the *standard* sequent systems [8, 20, 10, 9] mentioned in the introduction. Here we do not carry out all those simulations, but give some illustrative examples for the sequent systems—those for tableaux being more direct.

To begin with, in the Gbu system, the rules $R \overset{\rhd}{\to}$ and $R \overset{\not\rhd}{\to}$ are essentially and respectively unlabeled-sequent versions of our rules $\mathsf{F} \to_3$ and $\mathsf{F} \to_1$ (which are in turn the same as the rules for the tableaux TBI/IL), under the proviso that $\mathsf{F} \to_1$ applies with label c_i only if $\mathsf{T}\, A : c_h$ does not occur for any $c_h \preceq_{\mathtt{b}} c_i$, and otherwise $\mathsf{F} \to_3$ applies. As discussed in Section 5 above, the justification of this proviso is

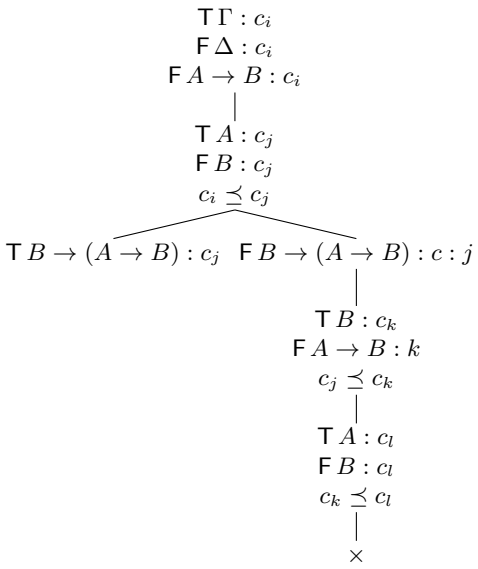


Figure 4: Simulation of $R \to_t$

semantical, and it is easy to see that it corresponds to the syntactic "evaluation" side conditions on those Gbu's rules [see 10].

Now, as pointed out above, $\mathrm{G3I}_t$ and LJT do not satisfy the subformula property but a relaxation thereof. This is witnessed by our simulations. Namely, the $R \to_t$ rule of $\mathrm{G3I}_t$ allows in derivations not only subformulas of formulas already in the derivation, but also formulas of the form $A \to (B \to A)$ for subformulas $B \to A$. The rule can be simulated in our system as shown in Figure 4.

In turn, the four implicative duplication-free rules of LJT, alias G4ip, imply a relaxation of the subformula property. Particularly, the rule $L \to\to$ of G4ip allows formulas of the form $B \to C$ for subformulas $(A \to B) \to C$ of formulas already in the derivation.[10] This rule can be simulated in our system as displayed in Figure 5. Actually, this simulation corresponds to the bottom-up semantic interpretation of $L \to\to$ given in [13, p. 128] to obtain its generalized formulation for the non-standard sequent system $\mathrm{LJT}_{\mathtt{SAT}}$.

Analogously to classical KE, our intuitionistic version might yield exponentially shorter derivations than standard tableau and cut-free standard sequent systems. Indeed, a class of formulas designed to prove this conjecture—but generalized to all signed tableau systems sharing the $\wedge/\vee$-rules with the classical one and their KE systems counterparts—was introduced in [21]. Moreover, endowed with free-variables, our intuitionistic KE reduces the search-space by delaying the choice of constants used to close a branch until enough information becomes available, and thus avoiding unnecessary expansion steps. A key subsequent work is then showing that our system is indeed superior in terms of proof-size and—since short proofs might be rather difficult to find—a detailed study of proof-search shall complement

[10] In original formulation of LJT [8], the rule $L \to\to$ is slightly different. The formulation of G4ip is the one underlying LJT's SAT-based variants [13, 11], and so we choose to simulate that formulation here.

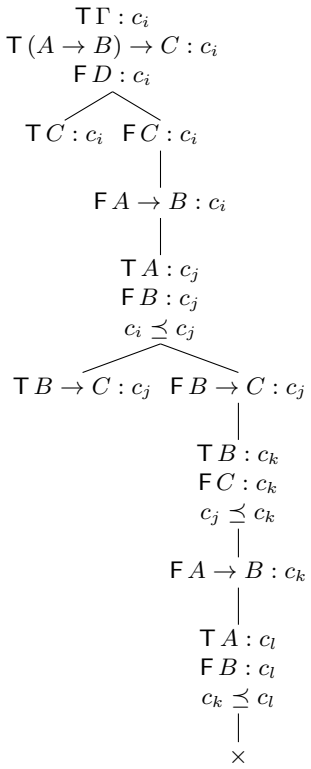


Figure 5: Simulation of $L \to\to$

our system's relative efficiency panorama. For the time being, we simply observe that w.r.t. popular examples as those presented in Figures 1 and 3—namely, (weak) Peirce law, Scott axiom, Heyting's axioms, double negation elimination, etc.—our system produces, respectively, shorter proofs and refutations than those produced by standard tableau and cut-free sequent systems.

6.4 Control on proof-search

As it is well-known, proof-search in sequent systems is executed from root to leaves by applying the rules to the node which, in each step of computation, is at the end of a search path. This requires some control—or "metarules" to use the terminology of [3]—to decide the rule or rules that can be applied to that last node. Usually, this is a simple pattern-matching between the formula labeling that node and the schema of the rule's conclusion. In case that more than one rule matches the schema under consideration, the resulting derivation should not depend on the chosen rule. However, there are controls that—even if decidable—cannot be expressed in a schematic notation, so that they are simply formulated in natural language as side conditions. In general, following [3], control over sequent and alike systems can be divided into two broad kinds: "local" and "global". Moreover, at least two kinds of global control can be distinguished: "look-back" and "look-ahead". The former extends its scope from the node at issue to the root, while the latter stretches its scope from the current node to the end nodes of every subtree that can be constructed from that point on.

In this subsection, we briefly discuss the kind of proof-search control that is used on some of the systems mentioned in this paper. To begin with, our KE system uses local control only for choosing which rule is to be applied on the current node from the list of applicable rules that match it. In turn, our system uses global look-back control in two respects: 1) listing out all the applicable rules on a particular node; 2) checking if either the rule $\mathsf{F} \rightarrow_1$ or $\mathsf{F} \rightarrow_3$ is applicable. By contrast, we do not use any form of look-ahead control. Further, any backtracking is avoided because only one branch, out of a list of branches, is active. Indeed, in line 5 of our Algorithm 1, we take a fixed branch and run a depth-first search over this branch. During this search, if the *PB* rule is applied, then we extend the current branch with a formula φ and add a new branch to the list of open branches, which is the previous one with $\overline{\varphi}$ instead. In line 2 of our Algorithm 1, it is checked if either the proof tree is closed or there is a completed branch (in which case, the Algorithm finishes with a countermodel). If neither is the case, in line 3 of our Algorithm 1, a new open branch is chosen.

Other systems implement similar or more involved kinds of control. For example, Corsi and Tassi's sequent systems IG and SIC [3] achieve termination by means of control on the application of a special rule called *a fortiori*. This control indeed resembles the proviso on $\mathsf{F} \rightarrow_1$ in our terminating KE-system. In a more involved way, loop-checking over G3ip guarantees termination by checking that the "same" sequent does not occur twice along a branch. However, as well-known, the effect of loop-checking on complexity is unclear. Indeed, the system $\mathrm{G3i}_t$ [9] uses a labeled variant of the principle of *a fortiori*, which makes it unnecessary to locally check for loops. In turn, the system Gbu [10] implements control and imposes "meta-conditions" over the applicability of the G3ip right-hand side rule for implicative formulas, directly replacing this rule by specialized variants $R \overset{\triangleright}{\rightarrow}$ and $R \overset{\not\triangleright}{\rightarrow}$. For

instance, the $R \overset{\not\in}{\to}$ rule is applicable only if the premises do not "evaluate" the antecedent of the respective implicative formula. This implies a kind of look-back control. Moreover, countermodel construction in Gbu requires a complex procedure, because it requires to run a dual system called Rbu, which in turn partially uses the corresponding open tree generated by Gbu. Finally, the SAT-based proof-search systems $\mathrm{LJT}_{\mathtt{SAT}}$ [13] and $C^{\to}$ [11] require heavy global control in order to resort to classical SAT tools, as mentioned in the next subsection.

6.5 Interpretability

As regards overall efficiency of decision procedures, the SAT-based proof-search systems of Fiorentini et al. [13, 11] are currently the best according to implementations on benchmark suites. These systems however require a pre-processing of the input formula into a generalized clausal form which, though computationally inexpensive, might hinder the interpretability of the output proofs and refutations. Namely, the pre-processing yields two kinds of clauses, flat and implicative. The former are interpreted classically, whereas the latter are interpreted intuitionistically. Based on this distinction, Fiorentini's non-standard sequent system $C^{\to}$ [11]—which, as mentioned in the introduction, underpins a refined version of the `intuit` algorithm—incorporates an additive variant of the $L \to\to$ rule. This variant, called cpl_1, essentially splits the search space into two branches: a classical branch, handled by a SAT solver, and an intuitionistic branch, which processes implicative clauses, one by one, by recursively applying the cpl_1 rule. In this way, most of the computation is performed by a SAT solver that is represented by a fully classical rule.

Thus, a comparison with this non-standard system must incorporate the clausification pre-processing. In particular, to compare it with our system through direct rule simulations, we would have to adapt our simulation of the $L \to\to$ rule in Figure 5 above to handle the rule cpl_1. So, we would use classical KE to simulate the classical branch mentioned above and, recursively, use intuitionistic KE to simulate the other branch—both with language restrictions. Analogous observations apply to the other SAT-based sequent system of Fiorentini et al. $\mathrm{LJT}_{\mathtt{SAT}}$ [13], for it shares with $C^{\to}$ the core classical-intuitionistic context split. Comparing with these systems via direct rule simulations seems contrived, and a more reasonable comparison would be in terms of implementation efficiency on benchmarks, which we leave for future work.

However, from a human-oriented perspective, a comparison with SAT-based methods should be done not only in terms of efficiency, but also interpretability. SAT-based rewriting methods depart from direct and natural treatments of the *meaning* of the intuitionistic connectives, so that their output proofs and refutation are not explanatory in intuionistic terms. A human user might expect that each inference step in a derivation witnessing theoremhood, or failure thereof, should be interpretable in the logical language at issue. By contrast, in our KE-style system, each stage of the derivation is transparent w.r.t. the meaning of the connectives according to the relational semantics. Proof and countermodel search is guided by "metalevel" traditional reasoning patterns, such as generalizations of modus ponens, modus tollens, disjunctive syllogism, etc.

6.6 Modularity and measure of inferential depth

Even though the SAT-based approach of [13, 11] can be extended to intermediate logics [12], it is *not* amenable for accommodating modalities. The main reason for this is that, unlike intermediate logics, the "forcing relation of modal Kripke models is not persistent; thus worlds are loosely related and must be handled by independent solvers" [12, p. 72]. The mentioned SAT-based methods seem to follow a rather case-by-case basis, which makes it difficult to use them for "mixed" reasoning. Say, for incorporating modalities or non-monotonicity in an intuitionistic environment. As for the Gbu's approach [10], it is initially less restrictive, but it does not show a comprehensive modularity at first glance.

In contrast, again exploiting the expressive power of labels, the approaches underlying G3I$_t$ [9], TBI/IL [15], and our KE system are clearly modular w.r.t. logics that admit relational semantics. Indeed, these are based on the methodology of translating frame conditions or axioms to rules via the labels mechanism [see 19]. Accordingly, we aim at extending our approach to other logics characterized by Kripke-style semantics, such as substructural, intermediate and modal, as well as combinations thereof.

Finally, KE-style systems provide a natural way of measuring the "depth" of their derivations by parameterizing the maximum number of nested applications of their branching rules and the set of formulas on which these rules are applied. In fact, for classical propositional logic [6] and some basic many-valued propositional logics [7], it has been shown that, whenever that number is fixed and that set is polynomially bounded in the length of the input problem, the associated deduction relation can be decided in polynomial time. In that line, we shall subsequently study the complexity of the corresponding deduction relation so parametrized in our intuitionistic KE system.

Funding

Solares-Rojas was funded by CONICET Postdoctoral Fellowships Program, and partly supported by the PAPIIT projects IA105923 and IG400422. The authors also acknowledge partial support by the MOSAIC project [EU H2020-MSCA-RISE-2020 Project 101007627].

Acknowledgements

We are grateful to Camillo Fiorentini, Ariana Manusovich, Marcello D'Agostino, Favio Miranda-Perea, and two anonymous referees for helpful comments and suggestions. We also thank the members of the BA Logic Group for useful discussion.

References

[1] A. Chagrov and M. Zakharyaschev. *Modal Logic*, volume 35 of *Oxford logic guides*. Oxford University Press, 1997.

[2] K. Claessen and D. Rosén. SAT modulo intuitionistic implications. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Proceedings of*

Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, volume 9450 of *Lecture Notes in Computer Science*, pages 622–637. Springer, 2015.

[3] G. Corsi and G. Tassi. Intuitionistic logic freed of all metarules. *The Journal of Symbolic Logic*, 72(4):1204–1218, 2007.

[4] M. D'Agostino and D. Gabbay. A generalization of analytic deduction via labelled deductive systems. Part I: Basic substructural logics. *Journal of Automated Reasoning*, 13(2):243–281, 1994.

[5] M. D'Agostino, D. Gabbay, and A. Russo. Grafting modalities onto substructural implication systems. *Studia Logica*, 59:65–102, 1997.

[6] 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.

[7] M. D'Agostino and A. Solares-Rojas. Tractable depth-bounded approximations to FDE and its satellites. *Journal of Logic and Computation*, 34(5):815–855, 2024.

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

[9] G. Fellin and S. Negri. A terminating intuitionistic calculus. *The Journal of Symbolic Logic*, 90(1):278–297, 2025.

[10] M. Ferrari, C. Fiorentini, and G. Fiorino. A terminating evaluation-driven variant of G3i. In D. Galmiche and D. Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 104–118, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[11] C. Fiorentini. Efficient SAT-based Proof Search in Intuitionistic Propositional Logic. In A. Platzer and G. Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 217–233, Cham, 2021. Springer International Publishing.

[12] C. Fiorentini and M. Ferrari. SAT-Based Proof Search in Intermediate Propositional Logics. In J. Blanchette, L. Kovács, and D. Pattinson, editors, *Automated Reasoning*, pages 57–74, Cham, 2022. Springer International Publishing.

[13] C. Fiorentini, R. Goré, and S. Graham-Lengrand. A Proof-Theoretic Perspective on SMT-Solving for Intuitionistic Propositional Logic. In S. Cerrito and A. Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 111–129, Cham, 2019. Springer International Publishing.

[14] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 34 of *Synthese Library*. D. Reidel Publishing, Dordrecht, 1983.

[15] D. Galmiche and D. Méry. Semantic Labelled Tableaux for Propositional BI without $\bot$. *Journal of Logic and Computation*, 13(5):707–753, 2003.

[16] G. Governatori. Labelled modal tableaux. In C. Areces and R. Goldblatt, editors, *Proceedings of Advances in Modal Logic*, pages 87–110. College Publications, 2008.

[17] S. Marin, M. Morales, and L. Straßburger. A fully labelled proof system for intuitionistic modal logics. *Journal of Logic and Computation*, 31(3):998–1022, 2021.

[18] 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.

[19] S. Negri. Proof analysis in non-classical logics. In C. Dimitracopoulos, L. Newelski, D. Normann, and J. Steel, editors, *Logic Colloquium 2005*, Lecture Notes in Logic, page 107–128. Cambridge University Press, 2007.

[20] L. Pinto and R. Dyckhoff. Loop-free construction of counter-models for intuitionistic propositional logic. In M. Behara, R. Fritsch, and R. Lintz, editors, *Proceedings of the 2nd Gauss Symposium: Conference A*, pages 226–232. de Gruyter, 1995.

[21] A. Solares-Rojas. *Tractable depth-bounded approximations to some propositional logics*. PhD thesis, University of Milan, 2022.

A Appendix A: Labeled KE without variables

Our intuitionistic KE formulated without free-variables is shown in Table 2. To contrast with the version using variables, in Figure 6, we show a proof in the system without them of the second example of Figure 1. It is not difficult to see that the termination of a slight variant of this system follows from essentially the same arguments provided for the version with variables above. That is, the rule $\mathsf{F}\ \to$ in Table 2 must be applied under the proviso of the same rule called $\mathsf{F}\ \to_1$ in the version with variables, and the rule $\mathsf{F}\ \to_3$ of the latter must also be incorporated.[11]

[11] Accordingly, in this terminating variant, note that the ls-formulas obtained in the penultimate node of the proof in Figure 6 are substituted by $\mathsf{F}\, A : c_3$.

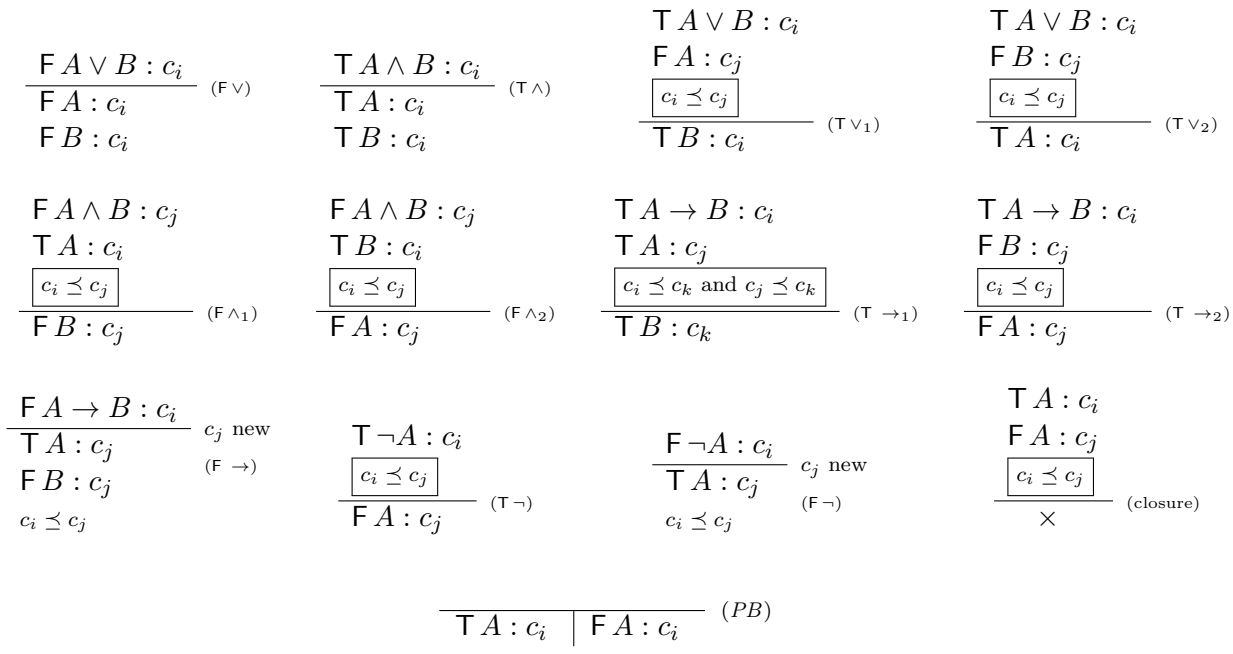


Table 2: KE-style rules without variables

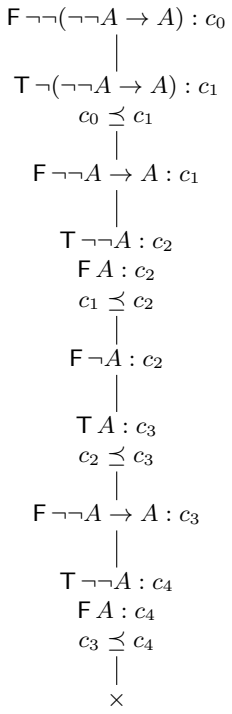


Figure 6: A proof using no variables