A Space Efficient Implementation of a Tableau Calculus for a Logic with a Constructive Negation

Alessandro Avellone[1], Camillo Fiorentini[2],
Guido Fiorino[1], and Ugo Moscato[1]

[1] Dipartimento di Metodi Quantitativi per le Scienze Economiche Aziendali,
Università Milano-Bicocca, piazza dell'Ateneo Nuovo, 1, 20126 Milano
{alessandro.avellone, guido.fiorino, ugo.moscato}@unimib.it
[2] Dipartimento di Scienze dell'Informazione,
Università degli Studi di Milano, via Comelico, 39, 20135 Milano, Italy
fiorenti@dsi.unimi.it

Abstract. A tableau calculus for a logic with constructive negation and an implementation of the related decision procedure is presented. This logic is an extension of Nelson logic and it has been used in the framework of program verification and timing analysis of combinatorial circuits. The decision procedure is tailored to shrink the search space of proofs and it is proved correct by using a semantical technique. It has been implemented in C++ language.

1 Introduction

Since the works of Nelson [11] and Thomason [14], logics with *constructive negation* ($\sim$) have been deeply investigated in the literature. Unlike intuitionistic negation ($\neg$), where $\neg A$ is understood as "A implies falsehood", the meaning of $\sim A$ is defined according to the structure of A, where the notion of falsity of atomic formula is as primitive as the concept of its truth (for a thorough discussion about constructive negation see [17]). Nelson logic **N** extends intuitionistic logic by adding a constructive negation. Accordingly, both positive and negative information has a constructive nature; indeed, **N** enjoys *disjunction property* (if a formula $A \vee B$ belongs to **N**, then either A or B belongs to **N**) and its negative counterpart, namely *constructible falsity* (if $\sim(A \wedge B)$ belongs to **N**, then either $\sim A$ or $\sim B$ belongs to **N**). Beyond **N**, many other logical systems with the same constructive features have been studied; for a comprehensive picture, we refer the reader to [7], where sequent calculi and Kripke semantics of predicate constructive logics with constructive negation are presented. The interest in such logics has been increased thanks to their applications in Computer Science; first of all, we mention the relevance of constructive negation in logic programming and in knowledge representation (see, e.g., [12, 13]).

In this paper we focus on a particular propositional logic with constructive negation, namely the propositional fragment of the logic **E** introduced in [10] (where **E** stays for "effective"). Instead of two negations, only the constructive

J. Marcinkowski and A. Tarlecki (Eds.): CSL 2004, LNCS 3210, pp. 488–502, 2004.

negation is used, but a new unary logical operator ($\Box$) is introduced to represent *classical truth* inside **E**: a formula $\Box A$ belongs to **E** if and only if A is classically valid. In the classification of [7], **E** coincides with the logic **N3o**, namely the logic obtained by adding to Nelson logic the *potential omniscience axiom* $\neg\neg(A \vee \sim A)$. The interaction between constructive falsity and classical truth provides a powerful environment where one can embed classical reasoning in a constructive setting, and this has a fruitful impact for Computer Science. Two recent trends encourage the research in this direction: in [3] it is described a framework based on logic **E** oriented to verification of computer programs; in [5] formal proofs of **E** are used to extract information about the propagation delays of signals in a combinatorial circuit (timing analysis).

In this context, the main contribution of the paper is to supply a space efficient tool to generate proofs of **E**. Firstly, we present a tableau calculus for **E**. Differently from the calculi presented in [1, 7], we aim to avoid what might produce inefficiency in the proof search task. Along the lines of [4, 9, 16], we avoid duplications of formulas: when a rule is applied to a formula A, A must not occur in the obtained configuration. The rules for the formulas $A \vee B$ and $\sim(A \wedge B)$ are defined according to their constructive meaning. A peculiar feature of our calculus is the combination of constructive and non-constructive tools; indeed, in particular configurations, we are allowed to continue a proof by using the rules for classical logic. The more expensive task in proof search strategy is due to backtracking. Typically, if one fails to build a closed proof table, one has to restore some old configuration and try the application of a different rule. In our implementation we reduce this kind of backtracking and, using a semantical argumentation, we show that the backtracking can be actually limited to few rules.

We have implemented in C++ language a decision procedure for the logic **E** based on the proof search strategy. The program is available at `http://www.dimequant.unimib.it/elogic/index.html`.

2 The Logic E

We consider the propositional language $\mathcal{L}$ based on a denumerable set of *propositional variables* and the logical constants $\sim, \wedge, \vee, \rightarrow$ and $\Box$. We denote with $p, q, \ldots$ propositional variables and with $A, B, \ldots$ arbitrary formulas. We write $A \leftrightarrow B$ as an abbreviation of $(A \rightarrow B) \wedge (B \rightarrow A)$. A *literal* is any formula of the kind p or $\sim p$, where p is a propositional variable. We denote with **Int** the set of intuitionistic valid formulas of the propositional language $\mathcal{L}_{\mathbf{Int}}$ having as logical constants $\neg, \wedge, \vee, \rightarrow$; **Cl** denotes the set of classically valid formulas of $\mathcal{L}$, where $\Box$ has to be trivially understood as the identity operator (namely, $\Box A$ is equivalent to A).

The logic **E** (in the predicate language) has been introduced in [10], where both a natural deduction calculus and a Kripke semantics is provided. In this section we outline some results presented in [10]. The logic **E** can be axiomatized by adding to the positive axioms of **Int** (see, for instance, [15]) the following

axioms which characterize $\sim$ as a *constructive negation* and $\Box$ as an operator to represent *classical truth*:

(E1). $\sim(A \wedge B) \leftrightarrow (\sim A \vee \sim B)$
(E2). $\sim(A \vee B) \leftrightarrow (\sim A \wedge \sim B)$
(E3). $\sim(A \rightarrow B) \leftrightarrow (A \wedge \sim B)$
(E4). $\sim\sim A \leftrightarrow A$
(E5). $A \wedge \sim A \rightarrow B$
(E6). $(\Box A \wedge \Box \sim A) \rightarrow B$
(E7). $(\sim A \rightarrow B \wedge \sim B) \rightarrow \Box A$
(E8). $(A \rightarrow B \wedge \sim B) \rightarrow \sim \Box A$

Clearly, **E** is contained in **Cl**. Constructive negation (also called *strong negation*) is weaker, with respect to provability, than classical negation; as a matter of fact, the classical tautologies $\sim (A \wedge \sim A)$, $(A \rightarrow B) \rightarrow (\sim B \rightarrow \sim A)$ and $(A \rightarrow B) \rightarrow \sim A \vee B$ do not belong to **E**. Moreover, unlike intuitionistic negation, constructive negation satisfies the principle of *constructible falsity (cf)*, which is the negative counterpart of *disjunction property (dp)*. This means that:

(cf). $\sim(A \wedge B) \in \mathbf{E}$ implies $\sim A \in \mathbf{E}$ or $\sim B \in \mathbf{E}$;
(dp). $A \vee B \in \mathbf{E}$ implies $A \in \mathbf{E}$ or $B \in \mathbf{E}$.

The $\Box$ operator allows us represent *classical truth* inside **E**; indeed:

(ct). $\Box A \in \mathbf{E}$ if and only if $A \in \mathbf{Cl}$.

Intuitionistic validity can be represented inside **E** by means of a translation map $\mathcal{T}$ defined on formulas of $\mathcal{L}_{\mathbf{Int}}$. As a matter of fact, let us define:

$$\mathcal{T}(p) = p, \text{ with } p \text{ a propositional variable;}$$
$$\mathcal{T}(A \oplus B) = \mathcal{T}(A) \oplus \mathcal{T}(B), \text{ with } \oplus \in \{\wedge, \vee, \rightarrow\};$$
$$\mathcal{T}(\neg A) = \Box \sim A.$$

Then:

(int). $A \in \mathbf{Int}$ if and only if $\mathcal{T}(A) \in \mathbf{E}$.

We point out that in the literature logics with both intuitionistic and constructive negation have been investigated (see, e.g., [7, 11, 14, 17]). The logic **E**, provided we define $\Box A$ as $\neg\neg A$, coincides with the logic **N3o** of [7], namely, the logic obtained by adding to Nelson logic **N3** the *potential omniscience axiom* $\neg\neg(A \vee \sim A)$. In [10] it is also presented the logic $\mathbf{E}^*$, which is maximal among the logics containing **E** and satisfying (dp), (cf) and (ct).

To treat constructive negation, we introduce a Kripke semantics equivalent to the one in [7, 10]. We denote with $\langle P, \leq \rangle$ a *poset* (partially ordered set), where P is a nonempty set and $\leq$ is a partial ordering between elements of P; $\langle P, \leq, \rho \rangle$ means that ρ is the minimum element of $\langle P, \leq \rangle$. We call *final element* of $\langle P, \leq \rangle$ any $\phi \in P$ that is maximal in $\langle P, \leq \rangle$ (that is, for every $\alpha \in P$, $\phi \leq \alpha$ implies $\phi = \alpha$). Given $\alpha \in P$, $\mathrm{Fin}(\alpha)$ denotes the set of final elements ϕ of $\langle P, \leq \rangle$

such that $\alpha \leq \phi$. Without loss of generality, we assume that, for every $\alpha \in P$, $\mathrm{Fin}(\alpha) \neq \emptyset$. A *Kripke model* for $\mathcal{L}$ is a structure $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$, where $\langle P, \leq, \rho \rangle$ is a poset and $\Vdash$ (the *forcing relation*) is a binary relation between elements α of P and literals l of $\mathcal{L}$ such that:

(K1). $\alpha \Vdash l$ and $\alpha \leq \beta$ implies $\beta \Vdash l$;
(K2). For every propositional variable p, it is not true that $\alpha \Vdash p$ and $\alpha \Vdash \sim p$;
(K3). For every final element ϕ of $\underline{K}$ and every propositional variable p, $\phi \Vdash p$ or $\phi \Vdash \sim p$.

The forcing relation is extended in a standard way to arbitrary formulas of $\mathcal{L}$ as follows:

1. $\alpha \Vdash A \wedge B$ iff $\alpha \Vdash A$ and $\alpha \Vdash B$;
2. $\alpha \Vdash A \vee B$ iff $\alpha \Vdash A$ or $\alpha \Vdash B$;
3. $\alpha \Vdash A \rightarrow B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\beta \Vdash A$ implies $\beta \Vdash B$;
4. $\alpha \Vdash \Box A$ iff, for every $\phi \in \mathrm{Fin}(\alpha)$, $\phi \Vdash A$;
5. $\alpha \Vdash \sim (A \wedge B)$ iff $\alpha \Vdash \sim A$ or $\alpha \Vdash \sim B$;
6. $\alpha \Vdash \sim (A \vee B)$ iff $\alpha \Vdash \sim A$ and $\alpha \Vdash \sim B$;
7. $\alpha \Vdash \sim (A \rightarrow B)$ iff $\alpha \Vdash A$ and $\alpha \Vdash \sim B$;
8. $\alpha \Vdash \sim \Box A$ iff, for every $\phi \in \mathrm{Fin}(\alpha)$, $\phi \Vdash \sim A$;
9. $\alpha \Vdash \sim\sim A$ iff $\alpha \Vdash A$.

We write $\alpha \nVdash A$ to mean that $\alpha \Vdash A$ does not hold. It is easy to check that properties (K1), (K2) and (K3) hold for arbitrary formulas as well. In this generalized formulation, (K1) is the usual monotonicity property of forcing relation, (K3) states that a final element ϕ of $\underline{K}$ behaves like a classical interpretation. Note that a classical interpretation $\mathcal{I}$ can be seen as a Kripke model having $\mathcal{I}$ as the only element and forcing relation defined in the obvious way.

A formula A *is valid in a Kripke model* $\underline{K}$ if and only if $\alpha \Vdash A$ for all elements α of $\underline{K}$. As proved in [10], **E** coincides with the set of formulas valid in all Kripke models.

3 The Tableau Calculus

The major contribute of this paper is the definition of a tableau calculus **Tab** for **E**. As far as we know, no tableau calculus for this logic has been presented in the literature. The object language of the calculus is based on the signs **T** and **F**. A *signed formula* (*sf* for short) is an expression of the form $\mathbf{T}A$ or $\mathbf{F}A$, where A is any formula; a **T**-formula is a sf with sign **T**, whereas an **F**-formula is a sf with sign **F**. The rules of **Tab** are in Tables 1-3. The meaning of the signs **T** and **F** is explained by the notion of *realizability*. Let $\underline{K} = \langle P, \leq, \Vdash \rangle$ be a Kripke model, let $\alpha \in P$, let A be a formula and let S be a set of sfs. We say that:

- $\alpha \rhd \mathbf{T}A$ (α *realizes* A) iff $\alpha \Vdash A$;
- $\alpha \rhd \mathbf{F}A$ iff $\alpha \nVdash A$;
- $\alpha \rhd S$ iff, for every $H \in S$, $\alpha \rhd H$.

We say that S is *realizable* iff there exists an element α of some model $\underline{K}$ such that $\alpha \rhd S$. A *configuration* is an expression of the form $S_1 \mid \ldots \mid S_n$ where, for all $i = 1, \ldots, n$, S_i is a set of sfs. In the rules of the calculus, we denote with $S, H_1, \ldots, H_m$ the set $S \cup \{H_1, \ldots, H_m\}$ and with S_T the set of $\mathbf{T}$-formulas of S. Every rule is applied to a signed formula of a configuration $S_1 \mid \ldots \mid S_i \mid \ldots S_n$; e.g., the notation $S, \mathbf{T}(A \wedge B)$ points out that the rule $\mathbf{T}\wedge$ is applied to the formula $\mathbf{T}(A \wedge B)$ of the set $S \cup \{\mathbf{T}(A \wedge B)\}$, where S is possibly empty; the schema

$$\frac{S_1 \mid \ldots \mid S, \mathbf{T}(A \wedge B) \mid \ldots \mid S_n}{S_1 \mid \ldots \mid S, \mathbf{T}A, \mathbf{T}B \mid \ldots \mid S_n}\,\mathbf{T}\wedge$$

illustrates an application of the rule $\mathbf{T}\wedge$. In every rule we distinguish two parts: the *premise*, that is the configuration above the line, and the *conclusion*, that is the configuration below the line. Differently from the calculi for logics with

Table 1.

$$\frac{S, \mathbf{T}(A \wedge B)}{S, \mathbf{T}A, \mathbf{T}B}\,\mathbf{T}\wedge \qquad \frac{S, \mathbf{F}(A \wedge B)}{S, \mathbf{F}A \mid S, \mathbf{F}B}\,\mathbf{F}\wedge$$

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

$$\frac{S, \mathbf{F}(A \to B)}{S_T, \mathbf{T}A, \mathbf{F}B}\,\mathbf{F}\to$$

$$\frac{S, \mathbf{T} \sim(A \wedge B)}{S, \mathbf{T} \sim A \mid S, \mathbf{T} \sim B}\,\mathbf{T}\sim\wedge \qquad \frac{S, \mathbf{F} \sim(A \wedge B)}{S, \mathbf{F} \sim A}\,\mathbf{F}\sim\wedge_1 \qquad \frac{S, \mathbf{F} \sim(A \wedge B)}{S, \mathbf{F} \sim B}\,\mathbf{F}\sim\wedge_2$$

$$\frac{S, \mathbf{T} \sim(A \vee B)}{S, \mathbf{T} \sim A, \mathbf{T} \sim B}\,\mathbf{T}\sim\vee \qquad \frac{S, \mathbf{F} \sim(A \vee B)}{S, \mathbf{F} \sim A \mid S, \mathbf{F} \sim B}\,\mathbf{F}\sim\vee$$

$$\frac{S, \mathbf{T} \sim(A \to B)}{S, \mathbf{T}A, \mathbf{T} \sim B}\,\mathbf{T}\sim\to \qquad \frac{S, \mathbf{F} \sim(A \to B)}{S, \mathbf{F}A \mid S, \mathbf{F} \sim B}\,\mathbf{F}\sim\to$$

$$\frac{S, \mathbf{T} \sim\sim A}{S, \mathbf{T}A}\,\mathbf{T}\sim\sim \qquad \frac{S, \mathbf{F} \sim\sim A}{S, \mathbf{F}A}\,\mathbf{F}\sim\sim$$

strong negation presented in [1, 7], we are interested in a calculus oriented to an efficient implementation. First of all, we aim to avoid duplications, thus to treat $\mathbf{T}(A \to B)$ we need several rules according to the structure of A (see [4, 16]). Moreover, to further reduce the depth of the proofs, the rules

$$\frac{S, \mathbf{T}((A \vee B) \to C)}{S, \mathbf{T}(A \to C), \mathbf{T}(B \to C)}\,\mathbf{T}\to\vee \qquad \frac{S, \mathbf{T}((A \to B) \to C)}{S_T, \mathbf{F}(A \to B), \mathbf{T}(B \to C) \mid S, \mathbf{T}C}\,\mathbf{T}\to\to$$

of [4, 16] are rewritten as in Table 2, where the new propositional variable p avoids the repetition of C in the former rule and of B in the latter rule (see [6, 8]).

Table 2.

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

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

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

$$\frac{S, \mathbf{T}((A \to B) \to C)}{S_T, \mathbf{T}A, \mathbf{F}p, \mathbf{T}(B \to p), \mathbf{T}(p \to C) \mid S, \mathbf{T}C}\,\mathbf{T}\to\to \qquad \frac{S, \mathbf{T}(\sim(A \to B) \to C)}{S, \mathbf{T}(A \to (\sim B \to C))}\,\mathbf{T}\to\sim\to$$

$$\frac{S, \mathbf{T}(\sim\sim A \to B)}{S, \mathbf{T}(A \to B)}\,\mathbf{T}\to\sim\sim$$

where the propositional variable p in $\mathbf{T}\to\sim\wedge$, $\mathbf{T}\to\vee$ and $\mathbf{T}\to\to$ is new

As already mentioned in the introduction, in some configurations we are allowed to apply to a set S the rules of a tableau calculus for **Cl**. To mark these sets, we use the notation $[S]_{\mathbf{Cl}}$ and we say that $[S]_{\mathbf{Cl}}$ is a *classical set*; intuitively, the signs $\mathbf{T}$ and $\mathbf{F}$ occurring in $[S]_{\mathbf{Cl}}$ have to be understood in a classical way (see the rules of Table 3).

Table 3.

$$\frac{S, \mathbf{T}\Box A}{[S_T, \mathbf{T}A]_{\mathbf{Cl}}}\,\mathbf{T}\Box \qquad \frac{S, \mathbf{F}\Box A}{[S_T, \mathbf{T}\sim A]_{\mathbf{Cl}}}\,\mathbf{F}\Box$$

$$\frac{S, \mathbf{T}\sim\Box A}{[S_T, \mathbf{T}\sim A]_{\mathbf{Cl}}}\,\mathbf{T}\sim\Box \qquad \frac{S, \mathbf{F}\sim\Box A}{[S_T, \mathbf{T}A]_{\mathbf{Cl}}}\,\mathbf{F}\sim\Box$$

$$\frac{S, \mathbf{T}(\Box A \to B)}{[S_T, \mathbf{T}\sim A]_{\mathbf{Cl}} \mid S, \mathbf{T}B}\,\mathbf{T}\to\Box \qquad \frac{S, \mathbf{T}(\sim\Box A \to B)}{[S_T, \mathbf{T}A]_{\mathbf{Cl}} \mid S, \mathbf{T}B}\,\mathbf{T}\to\sim\Box$$

A set S of sfs is *contradictory* iff one of the following conditions holds:

1. $\mathbf{T}A \in S$ and $\mathbf{F}A \in S$;
2. $\mathbf{T}A \in S$ and $\mathbf{T}\sim A \in S$;
3. S is a classical set and S is not **Cl**-consistent.

It is immediate to prove that:

Proposition 1. *Let $\underline{K} = \langle P, \leq, \Vdash \rangle$ be a Kripke model, let $\alpha \in P$ and let S be a contradictory set.*

1. *If S is not a classical set, then $\alpha \rhd S$ does not hold.*
2. *If S is a classical set and α is a final element of $\underline{K}$, then $\alpha \rhd S$ does not hold.*

A *proof table* for a set S is a finite sequence of configurations $\Gamma_1, \ldots, \Gamma_n$, where Γ_1 is the set S and the configuration Γ_{i+1} is obtained from $\Gamma_i = S_1 \mid \ldots \mid S_m$ by applying a rule to a non-contradictory set S_i. A *closed proof table* is a proof table $\Gamma_1, \ldots, \Gamma_n$ where all the sets in the last configuration are contradictory. We point out that to check that a classical set is contradictory, we can use any classical tableau calculus (extended in a trivial way to the language $\mathcal{L}$). Closed proof tables are the proofs of our calculus **Tab**. A set S is provable in **Tab** iff there exists a closed proof table for S; *A is provable in* **Tab** iff there exists a closed proof table for $\{\mathbf{F}A\}$. We remark that the rules of the calculus do not increase the number of **F**-formulas in a set. In particular, if the set in the first configuration of a proof table contains an **F**-formula at most, then every set occurring in the proof table contains an **F**-formula at most. Note that the rule $\mathbf{F} \rightarrow$ applied to a set S of this kind is invertible. On the other hand, the rules $\mathbf{F}\vee_i$ and $\mathbf{F} \sim \wedge_i$ are non-invertible, but they capture the constructive meaning of disjunction and negation.

Our aim is to exhibit an "efficient" sound and complete proof search strategy for closed proof tables of **Tab**. We begin by proving that **Tab** is sound for **E**. The main step consists in showing that the rules of **Tab** preserve realizability. It is easy to prove that:

Lemma 1. *Let $\underline{K} = \langle P, \leq, \Vdash \rangle$ be a Kripke model, let $\alpha \in P$ and let R be a rule of the calculus having S as premise and S_1 or $S_1 \mid S_2$ as consequence. If $\alpha \rhd S$, then there is $\beta \in P$ and $i \in \{1, 2\}$ such that $\alpha \leq \beta$ and $\beta \rhd S_i$. Moreover, if S_i is a classical set, then β is a final element of $\underline{K}$.*

From the above lemma we deduce that, if A does not belong to **E**, then no closed proof table for $\{\mathbf{F}A\}$ can exist. Indeed, let $\underline{K} = \langle P, \leq, \Vdash \rangle$ be a model such that A is not valid in $\underline{K}$ and let us assume that there exists a closed proof table $\Gamma_1, \ldots, \Gamma_n$ for $\{\mathbf{F}A\}$. Since $\underline{K}$ realizes $\{\mathbf{F}A\}$, by the previous lemma $\underline{K}$ realizes a set S of Γ_n, moreover if S a classical set, then S is realized in a final element of $\underline{K}$. This contradicts Proposition 1. It follows that A is not provable in **Tab**, hence:

Theorem 1 (Soundness). *If A is provable in* **Tab***, then A belongs to* **E***.*

In the following sections we prove that every formula of **E** is provable in **Tab** (Completeness Theorem).

4 The Proof Search Strategy

In this section we describe a procedure TAB which, given a set S of sfs, searches for a closed proof table for S. The main issue is to reduce backtracking in proof search. In our calculus the rules requiring backtracking are:

$$\mathbf{F}\vee_i\ ,\ \mathbf{F}\sim\wedge_i\ ,\ \mathbf{T}\to\to, \mathbf{T}\to\Box\ ,\ \mathbf{T}\to\sim\Box\ ,\ \mathbf{T}\Box\ ,\ \mathbf{T}\sim\Box \quad (i=1,2)$$

Since we consider sets of sfs having an $\mathbf{F}$-formula at most, the rules $\mathbf{F}\to$, $\mathbf{F}\Box$ and $\mathbf{F}\sim\Box$ are invertible, thus they do not require backtracking. Moreover, if S_T satisfies some properties (see Definition 1 in the next section), also the rules $\mathbf{F}\vee_i$, $\mathbf{F}\sim\wedge_i$, $\mathbf{T}\Box$ and $\mathbf{T}\sim\Box$ are invertible, as proved in Lemma 2 (see [2] for a thorough discussion).

To describe our procedure we introduce some classes $\mathcal{C}_j$ to identify sfs with the same behaviour:

$\mathcal{C}_1 = \{\, \mathbf{F}\Box A,\ \mathbf{F}\sim\Box A \,\};$
$\mathcal{C}_2 = \{\, \mathbf{T}(A\wedge B),\ \mathbf{F}(A\to B),\ \mathbf{T}\sim(A\vee B),\ \mathbf{T}\sim(A\to B),\ \mathbf{T}\sim\sim A, \mathbf{F}\sim\sim A,$
$\mathbf{T}((A\wedge B)\to C),\ \mathbf{T}(\sim(A\wedge B)\to C),\ \mathbf{T}((A\vee B)\to C),$
$\mathbf{T}(\sim(A\vee B)\to C),\ \mathbf{T}(\sim(A\to B)\to C),\ \mathbf{T}(\sim\sim A\to B)\,\};$
$\mathcal{C}_3 = \{\, \mathbf{F}(A\wedge B),\ \mathbf{T}(A\vee B),\ \mathbf{T}\sim(A\wedge B),\ \mathbf{F}\sim(A\vee B),\ \mathbf{F}\sim(A\to B)\,\};$
$\mathcal{C}_4 = \{\, \mathbf{T}(\Box A\to B),\ \mathbf{T}(\sim\Box A\to B)\,\};$
$\mathcal{C}_5 = \{\, \mathbf{T}((A\to B)\to C)\,\};$
$\mathcal{C}_6 = \{\, \mathbf{F}(A\vee B),\ \mathbf{F}\sim(A\wedge B)\,\};$
$\mathcal{C}_7 = \{\, \mathbf{T}\Box A,\ \mathbf{T}\sim\Box A\,\}.$

We describe a recursive procedure TAB(S, `applyAll`) that, given a set S of sfs containing at most an $\mathbf{F}$-formula and a boolean value `applyAll`, returns either a closed proof table for S or `NULL` if S is realizable (hence, no closed proof table for S can exist). The role of `applyAll` will be clarified in the next section; here we only point out that, when `applyAll` is `false`, we do not apply any rule to signed formulas in $S\cap(\mathcal{C}_4\cup\mathcal{C}_5)$ (see line 29 of the procedure). We assume to have a subroutine TABCL(S) that, given a set of sfs S, searches for a classical closed proof table for S. If a proof is found, TABCL(S) returns $[S]_{\mathbf{Cl}}$, otherwise it returns `NULL` (this means that S is $\mathbf{Cl}$-consistent). Let S be a set of sfs, let $H\in S$ and let S_1 or $S_1\mid S_2$ the configuration obtained by applying to S the rule $Rule(H)$ corresponding to H (when $H\in\mathcal{C}_6$, we write $Rule_1(H)$ or $Rule_2(H)$ to identify the rule). If Tab_1 and Tab_2 are closed proof tables for S_1 and S_2 respectively, then $\dfrac{S}{Tab_1}\,{\scriptstyle Rule(H)}$ or $\dfrac{S}{Tab_1\mid Tab_2}\,{\scriptstyle Rule(H)}$ denotes the closed proof table for S defined in the obvious way. Moreover, $\mathcal{R}_i(H)$ $(i=1,2)$ denotes the set containing the sfs of S_i which replace H. For instance:

$\mathcal{R}_1(\mathbf{T}(A\wedge B)) = \{\, \mathbf{T}A, \mathbf{T}B\,\};$
$\mathcal{R}_1(\mathbf{T}(A\vee B)) = \{\mathbf{T}A\}\ ;\ \mathcal{R}_2(\mathbf{T}(A\vee B)) = \{\mathbf{T}B\};$
$\mathcal{R}_1(\mathbf{T}((A\to B)\to C)) = \{\, \mathbf{T}A, \mathbf{F}p, \mathbf{T}(B\to p), \mathbf{T}(p\to C)\,\};$
$\mathcal{R}_2(\mathbf{T}((A\to B)\to C)) = \{\mathbf{T}C\}.$

The pseudo-code for TAB is the following:

```
FUNCTION TAB(S, applyAll )
 1  if ( (TA, FA ∈ S) OR (TA, T ∼A ∈ S) )
 2    then return S;
 3  if (S ∩ C1 ≠ ∅)
 4    then  Let H be the F-formula of S;
 5          Tab1 ← TABCL(ST ∪ R1(H));
 6          if (Tab1 ≠ NULL )
 7            then return  S / Tab1  Rule(H);
 8            else  return  NULL ;
 9  if ( TA, T(A → B) ∈ S )
10    then Tab1 ← TAB((S \ T(A → B)) ∪ {TB},  true );
11          if (Tab1 ≠ NULL )
12            then return  S / Tab1  T→;
13            else  return  NULL ;
14  if (S ∩ C2 ≠ ∅)
15    then  Let H ∈ S ∩ C2;
16          Tab1 ← TAB((S \ {H}) ∪ R1(H),  true );
17          if (Tab1 ≠ NULL )
18            then return  S / Tab1  Rule(H);
19            else  return  NULL ;
20  if (S ∩ C3 ≠ ∅)
21    then  Let H ∈ S ∩ C3;
22          Tab1 ← TAB((S \ {H}) ∪ R1(H),  true );
23          if (Tab1 ≠ NULL )
24            then Tab2 ← TAB((S \ {H}) ∪ R2(H),  true );
25                 if (Tab2 ≠ NULL )
26                   then return  S / Tab1 | Tab2  Rule(H);
27                   else  return  NULL ;
28            else  return  NULL ;
29  if ( applyAll AND  (S ∩ (C4 ∪ C5) ≠ ∅) )
30    then for (H ∈ (S ∩ (C4 ∪ C5)))
31           do Tab2 ← TAB((S \ {H}) ∪ R2(H),  true );
32              if (Tab2 =  NULL )
33                then return  NULL ;
34              if (H ∈ C4)
35                then Tab1 ← TABCL((ST \ {H}) ∪ R1(H));
36                else  Tab1 ← TAB((ST \ {H}) ∪ R1(H),  true );
37              if (Tab1 ≠ NULL )
38                then return  S / Tab1 | Tab2  Rule(H);
39  if (S ∩ C6 ≠ ∅)
40    then  Let H be the F-formula of S;
41          Tab1 ← TAB(ST ∪ R1(H),  false );
42          if (Tab1 ≠ NULL )
```

43 **then return** $\dfrac{S}{Tab_1}\,{}_{Rule_1(H)}$;
44 **else** $Tab_2 \leftarrow$ TAB$(S_T \cup \mathcal{R}_2(H),$ `false`);
45 **if** $(Tab_2 \neq$ `NULL`)
46 **then return** $\dfrac{S}{Tab_2}\,{}_{Rule_2(H)}$;
47 **else return** `NULL` ;
48 **if** ($(S \cap (\mathcal{C}_4 \cup \mathcal{C}_5) = \emptyset)$ AND $(S \cap \mathcal{C}_7 \neq \emptyset)$)
49 **then** Let $H \in S \cap \mathcal{C}_7$;
50 $Tab_1 \leftarrow$ TABCL$((S \setminus \{H\}) \cup \mathcal{R}_1(H))$;
51 **if** $(Tab_1 \neq$ `NULL`)
52 **then return** $\dfrac{S}{Tab_1}\,{}_{Rule(H)}$;
53 **else return** `NULL` ;
54 **return** `NULL` ;

We remark that, when one of the **if** conditions at lines 1, 3, 9, 14, 20, 39 and 48 is matched, the corresponding **then** instruction is executed and the procedure ends returning a value. This means that, independently of the choice of H, no backtracking is needed. On the contrary, in the **for** instruction at line 30 it might be necessary to try the application of a rule to all the formulas H in $S \cap (\mathcal{C}_4 \cup \mathcal{C}_5)$ and possibly to continue in line 39. We emphasize that to implement $\mathbf{F}\vee_i$, $\mathbf{F}\sim\wedge_i$, $\mathbf{T}\Box$ and $\mathbf{T}\sim\Box$ without backtracking, it is essential to apply these rules after having tried the application of all the other rules (see the proof of Proposition 2 in the next section).

Example 1. Let us consider the set of signed formulas

$$S \;=\; \{\, \mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{F}(a \vee \Box((b \to c) \to d)) \,\}$$

To search for a closed proof table for S, we call TAB$(S,$ `true`$)$.

(1). Since $\mathbf{T}((a \to (b \to c)) \to \Box d) \in \mathcal{C}_5$ and `applyAll` is `true`, the condition in the **if** statement of line 29 is matched. This means that the procedure tries to apply $\mathbf{T}\to\!\!\to$ to S, therefore closed proof tables for the sets

$$S_1 \;=\; \{\, \mathbf{T}a\;,\; \mathbf{T}((b \to c) \to p)\;,\; \mathbf{T}(p \to \Box d)\;,\; \mathbf{F}p \,\}$$
$$S_2 \;=\; \{\, \mathbf{T}\Box d\;,\; \mathbf{F}(a \vee \Box((b \to c) \to d)) \,\}$$

are searched.

(2). The call TAB$(S_2,$ `true`$)$ of line 31 is executed in order to build a closed proof table for S_2.

(3). The call TAB$(S_3,$ `false`$)$ of line 41 is executed, where $S_3 = \{\mathbf{T}\Box d, \mathbf{F}a\}$, which corresponds to the application of $\mathbf{F}\vee_1$ to S_2.

(4). The application of $\mathbf{T}\Box$ to S_3 is tried by the call TABCL$(\{\mathbf{T}d\})$ (line 50). The `NULL` value is returned (indeed, $\mathbf{T}d$ is **Cl**-consistent) and also TAB$(S_3,$ `false`$)$ fails (line 53 is executed and `NULL` is returned).

(5). The execution of TAB$(S_2,$ `true`$)$ continues in line 44 with the computation of TAB$(S_4,$ `false`$)$, where

$$S_4 \;=\; \{\, \mathbf{T}\Box d\;,\; \mathbf{F}\Box((b \to c) \to d) \,\}$$

namely, $\mathbf{F}\vee_2$ is applied to S_2.

(6). The call TABCL($\{\mathbf{T}\Box d, \mathbf{T} \sim ((b \to c) \to d)\}$) of line 5 is executed ($\mathbf{F}\Box$ is applied to S_4) and a classical proof table is found. Thus, both the calls in (5) and (2) succeed and the closed proof table Tab_2 is built as follows:

$$\dfrac{\dfrac{\mathbf{T}\Box d\,,\, \mathbf{F}(a \vee \Box((b \to c) \to d))}{\mathbf{T}\Box d\,,\, \mathbf{F}\Box((b \to c) \to d)}\mathbf{F}\vee_2}{[\mathbf{T}\Box d\,,\, \mathbf{T} \sim ((b \to c) \to d)]_{\mathbf{Cl}}}\mathbf{T}\Box$$

(7). Now, the computation of TAB(S, `true`) continues with the call TAB(S_1, `true`) (line 36) in order to build a closed proof table for S_1.

(8). The condition in the **if** statement of line 29 is matched, thus the **for** loop in line 30 is executed. This means that it is tried the application of the rule $\mathbf{T} \to\!\to$ to S_1 for *all* the signed formulas of the kind $\mathbf{T}((H_1 \to H_2) \to H_3)$. We have *two* signed formulas of this kind and it is easy to check that in both cases the search for a closed proof table fails. It follows that no closed proof table for S_1 can be built; nevertheless, TAB(S, `true`) does not fail, but the computation continues with the statements after line 38.

(9). The condition in the **if** statement of line 39 is satisfied, thus the computation continues with the call TAB(S_5, `false`) of line 41, where

$$S_5 \;=\; \{\, \mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{F}a \,\}$$

(it corresponds to apply $\mathbf{F}\vee_1$ to S). The procedure immediately fails. As a matter of fact, the application of $\mathbf{T} \to\!\to$ is not tried since the value of `applyAll` is `false`, and no other **if** statement can be executed; hence, line 54 is executed and `NULL` is returned.

(10). The call TAB(S_6, `false`) of line 44, where

$$S_6 \;=\; \{\, \mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{F}\Box((b \to c) \to d) \,\}$$

is executed (it corresponds to apply $\mathbf{F}\vee_2$ to S).

(11). Since the value of `applyAll` is false, the instructions inside the **if** statement of line 29 are not executed (namely, the application of $\mathbf{T} \to\!\to$ is not tried), but the call TABCL(S_7) in line 5 is executed ($\mathbf{F}\Box$ is applied to S_6), where

$$S_7 \;=\; \{\, \mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{T} \sim ((b \to c) \to d) \,\}$$

The procedure succeeds in finding out a classical closed proof table for S_7; accordingly, both TAB(S_6, `false`) and TAB(S, `true`) succeed and the returned closed table for S is:

$$\dfrac{\dfrac{\mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{F}((a \vee \Box((b \to c) \to d)))}{\mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{F}\Box((b \to c) \to d)}\mathbf{F}\vee_2}{[\mathbf{T}((a \to (b \to c)) \to \Box d)\;,\; \mathbf{T} \sim ((b \to c) \to d)]_{\mathbf{Cl}}}\mathbf{F}\Box$$

Example 2. Let us consider the set

$$S = \{ \mathbf{T}((p \vee q) \vee r) \, , \, \mathbf{F}q \}$$

To build a closed proof table for S, we call TAB$(S, \texttt{true})$. Line 22 is executed and TAB$(S_1, \texttt{true})$ is called, where $S_1 = \{\mathbf{T}(p \vee q), \ \mathbf{F}q\}$ ($\mathbf{T}\vee$ is applied to S). Again, line 22 is executed and TAB$(S_2, \texttt{true})$ is called, where $S_2 = \{\mathbf{T}p, \ \mathbf{F}q\}$. Now, no condition associated with the **if** statements is matched, hence TAB$(S_2, \texttt{true})$ immediately fails (line 54 is executed and `NULL` is returned). This implies that TAB$(S_1, \texttt{true})$ and TAB$(S, \texttt{true})$ immediately fail (indeed, in both cases line 28 is executed and `NULL` is returned) and no proof table for S is found.

To prove the termination of TAB and the Completeness Theorem we define the function dg as follows:

- if l is a literal, then $\mathrm{dg}(l) = 0$;
- $\mathrm{dg}(A \wedge B) = \mathrm{dg}(A) + \mathrm{dg}(B) + 2$;
- $\mathrm{dg}(A \vee B) = \mathrm{dg}(A) + \mathrm{dg}(B) + 3$;
- $\mathrm{dg}(A \to B) = \mathrm{dg}(A) + \mathrm{dg}(B) +$ (number of implications occurring in A)$+1$;
- $\mathrm{dg}(\sim A) = \mathrm{dg}(A) + 1$;
- $\mathrm{dg}(\Box A) = \mathrm{dg}(A)$;
- if S is a set of sfs, we set $\mathrm{dg}(S) = \sum_{H \in S} \mathrm{dg}(H)$.

It is easy to check that, if S is a set of sfs and S' is obtained from S by an application of a rule of **Tab**, then $\mathrm{dg}(S') < \mathrm{dg}(S)$. Using this fact, it is immediate to prove that TAB always terminates.

Remark 1. Along the lines of [6], it is possible to prove that the depth of every proof table of **Tab** is linearly bounded in the proved formula. This property implies the space efficiency of TAB (see the discussion in Section 6).

5 Completeness

We prove that, when the call TAB$(S, \texttt{applyAll})$ returns `NULL`, S is realizable and we can actually build a countermodel for S (namely, a model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd S$). To justify the lack of backtracking in rules $\mathbf{F}\vee_i$, $\mathbf{F} \sim \wedge_i$, $\mathbf{T}\Box$ and $\mathbf{T} \sim \Box$ we introduce the notion of $\to$-realizability.

Definition 1. *A set S_T of $\mathbf{T}$-formulas is* $\to$-realizable *iff the following holds:*

1. $S_T \subseteq \{\mathbf{T}l \mid l \text{ is a literal}\} \cup \mathcal{C}_4 \cup \mathcal{C}_5 \cup \mathcal{C}_7$;
2. *For all* $\mathbf{T}(l \to B) \in S_T$, *with* l *a literal,* $\mathbf{T}l \notin S_T$;
3. *For all* $\mathbf{T}(\Box A \to B) \in S_T$, *the set* $(S_T \setminus \{\mathbf{T}(\Box A \to B)\}) \cup \{\mathbf{T} \sim A\}$ *is realizable;*
4. *For all* $\mathbf{T}(\sim \Box A \to B) \in S_T$, *the set* $(S_T \setminus \{\mathbf{T}(\sim \Box A \to B)\}) \cup \{\mathbf{T}A\}$ *is realizable;*
5. *For all* $\mathbf{T}((A \to B) \to C) \in S_T$, *the set* $(S_T \setminus \{\mathbf{T}((A \to B) \to C)\}) \cup \{\mathbf{T}A, \mathbf{F}p, \mathbf{T}(B \to p), \mathbf{T}(p \to C)\}$, *where p is new, is realizable.*

Remark 2. If S_T is $\rightarrow$-realizable and $S_T \cap (\mathcal{C}_4 \cup \mathcal{C}_5) \neq \emptyset$, then S_T is realizable as well.

Using a semantical construction on Kripke models, in [2] it is proved that:

Lemma 2. *Let S_T be $\rightarrow$-realizable. Then:*

(i). $S_T, \mathbf{F}(A \vee B)$ is realizable iff both $S_T, \mathbf{F}A$ and $S_T, \mathbf{F}B$ are realizable.
(ii). $S_T, \mathbf{F} \sim (A \wedge B)$ is realizable iff both $S_T, \mathbf{F} \sim A$ and $S_T, \mathbf{F} \sim B$ are realizable.
Moreover, if l is a literal and $\mathbf{T}l \notin S_T$, then:
(iii). $S_T, \mathbf{F}l$ is realizable iff S_T is $\mathbf{Cl}$-consistent.

We say that the call TAB$(S, \texttt{applyAll})$ is *sound* iff `applyAll` is `true` or S_T is $\rightarrow$-realizable.

Proposition 2. *Let S be a set of sfs containing at most one $\mathbf{F}$-formula, let* TAB$(S, \texttt{applyAll})$ *be a sound call and suppose that* TAB$(S, \texttt{applyAll})$ *returns the* `NULL` *value. Then, there is a Kripke model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd S$.*

Proof. Let us assume, by induction hypothesis, that the proposition holds for all sets S' such that $\mathrm{dg}(S') < \mathrm{dg}(S)$. We prove that the proposition holds for S by inspecting all the possible cases where the procedure returns the `NULL` value. We show some significant cases (the whole proof is in [2]).

The instruction at line 33 has been executed.
Let us assume, for instance, that, at line 31, the call TAB$(S', \texttt{true})$ has been executed, with $S' = (S \setminus \{\mathbf{T}(\Box A \rightarrow B)\}) \cup \{\mathbf{T}B\}$. By induction hypothesis there exists a Kripke model $\underline{K} = \langle P, \leq, \rho, \Vdash \rangle$ such that $\rho \rhd S'$, hence $\rho \rhd S$.

Remark 3. If none of the conditions at lines 1, 3, 9, 14, 20 and 29 holds, we claim that S_T is $\rightarrow$-realizable. Indeed, if the second parameter of the function is `false`, this follows by the hypothesis of the proposition. Otherwise, using the induction hypothesis, one can easily check that S_T satisfies the definition of $\rightarrow$-realizability. In particular, the realizability of $(S_T \setminus \{H\}) \cup \mathcal{R}_1(H)$, for $H \in \mathcal{C}_4 \cup \mathcal{C}_5$, follows by the fact that the procedure has not terminated inside the **for** instruction at line 30, since the value of Tab_1 is `NULL`.

The instruction at line 47 has been executed.
Suppose that $S = S_T \cup \{\mathbf{F}(A \vee B)\}$ and let $S_A = S_T \cup \{\mathbf{F}A\}$, $S_B = S_T \cup \{\mathbf{F}B\}$. Then, both the call TAB$(S_A, \texttt{false})$ and TAB$(S_B, \texttt{false})$ have returned the `NULL` value. Note that both calls are sound (indeed, $(S_A)_T = (S_B)_T = S_T$ and S_T is $\rightarrow$-realizable) thus, by induction hypothesis, both $S_T, \mathbf{F}A$ and $S_T, \mathbf{F}B$ are realizable. By Lemma 2(i), S is realizable. The case $S = S_T \cup \{\mathbf{F} \sim (A \wedge B)\}$ is similar. □

By the above proposition, it immediately follows the Completeness Theorem.

Theorem 2 (Completeness). *If A belongs to $\mathbf{E}$, then* TAB$(\{\mathbf{F}A\}, \texttt{true})$ *returns a closed proof table for $\mathbf{F}A$.*

As a consequence of the above theorem, we have a trivial proof of properties (cf), (dp) and (ct) of $\mathbf{E}$ stated in Section 2

6 Implementation of the Decision Procedure

We have implemented a decision procedure based on TAB and TABCL. The implementation uses the signs $\mathbf{T}, \mathbf{F}, \mathbf{T_c}$ and $\mathbf{F_c}$. The signs $\mathbf{T}$ and $\mathbf{F}$ are used as in TAB. When the procedure TABCL is called, the signs of the formulas are turned into $\mathbf{T_c}$ (classical truth) and $\mathbf{F_c}$ (classical falsity). To treat formulas with signs $\mathbf{T_c}$ and $\mathbf{F_c}$ rules of a classical tableau calculus are used. Since the rules for Classical logic are invertible backtracking is not required. To reduce the number of nodes in the classical tableau the rules having one set in the conclusion are applied first. Since signed formulas $\mathbf{T}H$ and $\mathbf{T_c}H$, where H is an axiom of **Cl**, are not needed to close a proof table, they are deleted any time they appear in a configuration. Whenever a rule is applied, the condition in Line 1 is checked as follows. If the rule related to H is applied to "$(S \setminus \{H\}), H$", the consistence of the resulting set S_i is checked considering every formula in $\mathcal{R}_i(H)$: every $\mathbf{T}$-formula in $\mathcal{R}_i(H)$ is checked against the $\mathbf{F}$-formula and the $\mathbf{T}$-formulas of $S_i \setminus \mathcal{R}_i(H)$. If $\mathcal{R}_i(H)$ contains the $\mathbf{F}$-formula, then it is checked against the $\mathbf{T}$-formulas of $S_i \setminus \mathcal{R}_i(H)$. The implementation proceeds in a similar way for the signs $\mathbf{T_c}$ and $\mathbf{F_c}$.

TAB is implemented as an iterative procedure. The implementation uses a stack to take into account two different levels of backtracking. The former level of backtracking, related to the **for** statement in line 30, is used to explore the search space of the proof table. The latter level of backtracking, related to lines 24 and 36, is used to visit with a depth-first strategy a single proof table to determine if it is closed. The stack has, at most, as many elements as the longest branch of the deepest proof table in the search space. Every element of the stack contains the sets of formulas of the nodes in the branch the procedure is visiting and two integers denoting, respectively, which formula of the set has been used to get the subsequent set and if the right subtree of the node has already been visited. By Remark 1, the stack has a number of elements linearly bounded in the length of the formula to be proved. Moreover, the number of symbols in each node of every proof table is linearly bounded in the length of the formula to be proved. Thus, the implementation is $O(n^2)$-SPACE. Finally, the iteration in line 30 is implemented to apply the rules of $\mathcal{C}_4$ first, since the first set in the conclusion gives rise to a classical set of formulas.

7 Conclusion and Future Work

We have provided a tableau calculus for the logic $\mathbf{E}$ and the related decision procedure that minimizes the backtracking. The implementation has been developed in C++ language. [3] Since we are interested in using the logic $\mathbf{E}$ in the field of timing analysis, we plan to extend our program in order to extract timing information from proofs of $\mathbf{E}$, to implement the algorithms described in [5].

[3] The program is available at `http://www.dimequant.unimib.it/elogic/index.html`.

References

1. S. Akama. Tableaux for logic programming with strong negation. In *Automated reasoning with analytic tableaux and related methods (Pont-à-Mousson, 1997)*, Lecture Notes in Artificial Intelligence, pages 31–42. Springer, Berlin, 1997.
2. A. Avellone, C. Fiorentini, G. Fiorino, and U. Moscato. An efficient implementation of a tableau calculus for a logic with a constructive negation. Technical Report 83, Dipartimento di Metodi Quantitativi per le Scienze Economiche Aziendali, Università Milano-Bicocca., 2004. Available at `http://homes.dsi.unimi.it/~fiorenti`.
3. M. Benini. *Verification and Analysis of Programs in a Constructive Environment*. PhD thesis, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy, 1999.
4. R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.
5. M. Ferrari, C. Fiorentini, and M. Ornaghi. Extracting exact time bounds from logical proofs. In A. Pettorossi, editor, *Logic Based Program Synthesis and Transformation, 11th International Workshop, LOPSTR 2001, Selected Papers*, volume 2372 of *Lecture Notes in Computer Science*, pages 245–265. Springer-Verlag, 2002.
6. G. Fiorino. Space-efficient decision procedures for three interpolable propositional intermediate logics. *J. Logic Comput.*, 12(6):955–992, 2002.
7. I. Hasuo and R. Kashima. Kripke completeness of first-order constructive logics with strong negation. *IGPL*, 11(6):615–646, 2003.
8. J. Hudelmaier. An $O(n \log n)$-SPACE decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.
9. 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.
10. P. Miglioli, U. Moscato, M. Ornaghi, and G. Usberti. A constructivism based on classical truth. *Notre Dame J. Formal Logic*, 30(1):67–90, 1989.
11. D. Nelson. Constructible falsity. *J. Symbolic Logic*, 14:16–26, 1949.
12. D. Pearce. Reasoning with negative information. II. Hard negation, strong negation and logic programs. In *Nonclassical logics and information processing (Berlin, 1990)*, volume 619 of *Lecture Notes in Comput. Sci.*, pages 63–79. Springer, Berlin, 1992.
13. D. Pearce and G. Wagner. Logic programming with strong negation. In *Extensions of logic programming (Tübingen, 1989)*, volume 475 of *Lecture Notes in Comput. Sci.*, pages 311–326. Springer, Berlin, 1991.
14. R.H. Thomason. A semantical study of constructible falsity. *Z. Math. Logik Grundlagen Math.*, 15:247–257, 1969.
15. A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.
16. N. N. Vorob'ev. A new algorithm of derivability in a constructive calculus of statements. In *Sixteen papers on logic and algebra*, volume 94 of *American Mathematical Society Translations, Series 2*, pages 37–71. American Mathematical Society, Providence, R.I., 1970.
17. H. Wansing. *The logic of information structures*, volume 681 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1993. Lecture Notes in Artificial Intelligence.