How to Avoid the Formal Verification of a Theorem Prover

Alessandro Avellone, *Dipartimento di Metodi Quantitativi per l'Economia, Università degli Studi di Milano–Bicocca, Piazza dell'Ateneo Nuovo 1, 20126 Milano–Italy.*
E-mail: alessandro.avellone@unimib.it

Marco Benini, *Centro di Ricerca Informatica Interattiva, Università dell'Insubria, via Ravasi 2, 21100 Varese–Italy.*
E-mail: benini@dsi.unimi.it

Ugo Moscato, *Dipartimento di Metodi Quantitativi per l'Economia, Università degli Studi di Milano–Bicocca, Piazza dell'Ateneo Nuovo 1, 20126 Milano–Italy.*
E-mail: ugo.moscato@unimib.it

Abstract

The purpose of this papers to show a technique to automatically certify answers coming from a non-trustable theorem prover. As an extreme consequence, the development of non-sound theorem provers has been considered and investigated, in order to evaluate their relative efficiency on particular classes of difficult theorems.

The presentation will consider as a case study, a tableau-based theorem prover for first-order intuitionistic logic without equality [1].

Keywords: Program Verification, Automated Theorem Proving, Intuitionistic Logic, Program Translation

1 Introduction

The importance of formal verification in theorem proving has been pointed out by many authors, see, e.g. [9, 18, 10, 4]. Since the prominent application of theorem proving is formal verification of hardware and software, it is essential that the instruments used to reason on the verification problems are trustable. In fact, at least in applications, it is not required [26, 18, 19] for a theorem prover to be *complete*, that is, the ability to prove every theorem, but it is essential that, whenever the prover attributes to a formula the status of theorem, its judgment is correct. In other words, failures in proving are allowed, but failures in proofs are not.

Naturally, in the context of formal verification, theorem provers are developed as parts of wider systems, devoted to the representation and the solution of verification problems. Most of these verification systems are based on the so-called *logical frameworks* [13], i.e., programs able to represent a variety of logical theories, and to reason on them.

The problem we are approaching is not new; techniques to develop automatic reasoners which do not require formal validation have been developed for HOL [10, 11, 12] and Isabelle [20, 22]. All these techniques require that the logical system, i.e., the core of the logical framework which forms the basis of the verification system, is correct. This assumption is reasonable, since to have a correct theorem prover in a non-trustable environment still makes the whole system, environment plus prover, non-trustable.

To explain the novelty of our approach, and the differences with the cited works about auto-validating provers, we have to step back, and to analyze the building of a theorem prover inside a logical framework. In fact, a logical framework offers to the provers, human or mechanical, the set of inference rules which constitute the description of the logic or theory under examination.

In the following, the word *correct* referred to theorem provers often appears; the precise reading is *correct with respect to the logical framework*, that means, formally, a prover P is correct if the set T of theorems it can prove is a subset of the set L of theorems which can be proved using the logical framework alone.

A theorem prover may use only the inference rules of the framework to perform a proof, thus it is an algorithmic description of a strategy to prove theorems in the logical system. This class of provers is intrinsically correct: they cannot *prove* but theorems because of the assumption that the logical framework is correct.

In contrast, a theorem prover may reason on a different presentation of the same theory, e.g., using a tableau calculus instead of natural deduction. In this case, the trusting problem appears, since the proof of a theorem is performed *outside* the logical system provided by the environment.

The idea behind the cited approaches on auto-validation is *to embed* the internal logical system, i.e., the calculus the prover is based on, into the environment and to reduce the prover to an implementation of a strategy. As previously remarked, this shift of responsibility assures the prover correctness.

The embedding technique is widely used, and it has many benefits, but, in our opinion, it suffers from some drawbacks as well: in particular, it requires to search for a proof in the proof-space of the logical framework, which is usually richer than the proof-space of the internal representation. For example, in a logical framework like Isabelle, working in higher-order classical logic, if a prover for first-order intuitionistic logic has been developed, then it has to carry information on the representation of formulas, such as types, which the prover is unable to use. We cannot avoid these pieces of information when translating a formula to be proven, i.e., a *goal*, from the Isabelle language to the internal language of the prover; but they are useless during the proof-search process. In this case, useless means dangerous because these pieces of information require memory space, and this is the most critical resource when searching for a proof.

In the embedding technique, not having an no internal representation for goals and proofs, the prover has not to cope with the correctness of representation nor with the correctness of goal manipulation, but representing a proof attempt requires more space. In the prover with an internal logical system, one has to verify that the representation of goals and proofs is adequate, and that their manipulations are correctly implemented, nevertheless the representation is oriented to the search of a proof with a particular strategy, the one of the prover, thus avoiding the coding of

useless pieces of information in the representation of proof attempts.

As a matter of fact, the technique we are proposing tries to combine the benefits of both approaches: like the embedding technique it does not require a formal verification of the prover to ensure that a proof is valid, but it uses an internal representation of goals and proofs to minimize the amount of information to keep in memory when searching for a proof.

Moreover, the proposed approach permits to develop provers whose internal logical system is not sound, but they allow, in some cases, to develop sound proofs for difficult theorems quite faster than sound provers.

In the following, we will develop the theory underlying our approach, together with a discussion of the lack of completeness it introduces in a prover. Then we will analyze a case of non-sound calculus which permits to prove a class of formulas more efficiently than the sound version of the same prover. The approach description is specialized for a tableau prover which works in Isabelle's higher-order classical logic and proves goals in the intuitionistic first-order fragment.

2 Preliminary Definitions

The set of *well formed formulas*, *wffs* for short, is defined in the usual way [28], starting from the propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, the quantifiers $\forall$ and $\exists$ and a denumerable set of individual variables.

The theory **IL** denotes the set of all intuitionistically valid wffs. Our choice for a natural deduction system is the calculus **Ni** as shown in [28]; for reference, the complete set of its inference rules are reported in Table 1. Since in [29], the **Ni** calculus is proven to be sound and complete, we will refer to it as **IL**.

The tableau calculus for first-order intuitionistic predicate logic the prover of the case study is based on, is defined in [15, 16, 17] and it uses the three signs **T**, **F** and $\mathbf{F_c}$. Given a wff A, a *signed well formed formula*, *swff* for short, will be every expression of the kind $\mathcal{S}A$, where $\mathcal{S} \in \{\mathbf{T}, \mathbf{F}, \mathbf{F_c}\}$, and A is a wff.

The intuitionistic tableau calculus **IL**-T is given by the rules in Table 2, where S_c is the *certain part* of S, formally,

$$S_c = \{\mathbf{T}X \mid \mathbf{T}X \in S\} \cup \{\mathbf{F_c}X \mid \mathbf{F_c}X \in S\} \ .$$

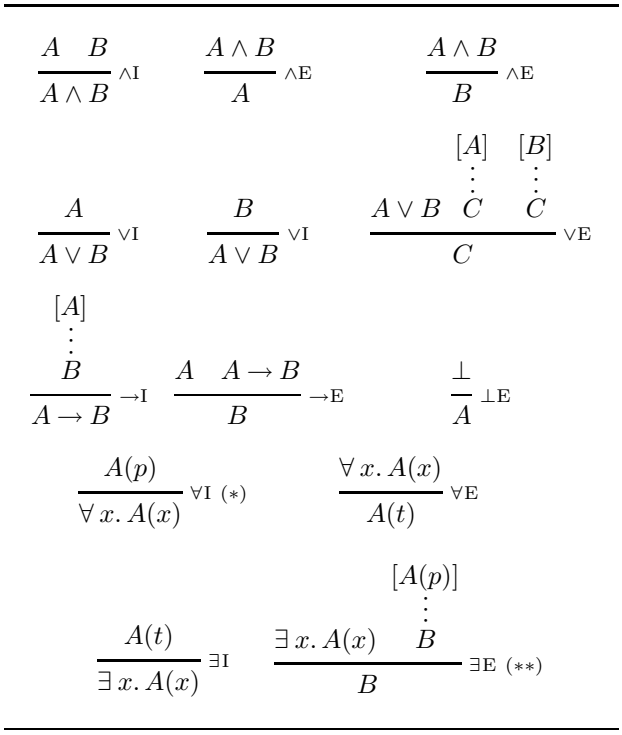
A *node* is any set of swffs in a *configuration*, that is, a finite sequence of sets of swffs.

Definition 2.1 Let S be a set of swffs, a *tableau*, or *proof-table* for S is a finite sequence of configurations $C_1, \ldots, C_n$ such that

- $C_1 = \{S\}$, and
- $C_{i+1} = \{S_1, \ldots, S_n\} \cup \{N_1, \ldots, N_k\}$, where $C_i = \{S_1, \ldots, S_n\} \cup \{M\}$ and

$$\frac{M}{N_1 \mid \ldots \mid N_k}$$

is an instance of an inference rule as in Table 2.



where, in $(*)$ and $(**)$, p is an eigenvariable.

TABLE 1. The **IL** calculus.

The *active* swff in a node is the swff an inference rule is applied to, the other swffs in that node are called the *context*. A node in the last configuration, the *terminal configuration* of a tableau is called a *terminal node*.

A tableau is *closed* iff all the sets S_j of its final configuration are contradictory, where a set S is *contradictory* if, for some A, either $\{\mathbf{T}A, \mathbf{F}A\} \subseteq S$ or $\{\mathbf{T}A, \mathbf{F_c}A\} \subseteq S$. Accordingly, a*complementary pair* is any set of swffs of the form $\{\mathbf{T}A, \mathbf{F}A\}$ or of the form $\{\mathbf{T}A, \mathbf{F_c}A\}$. A *proof* of a wff B in **IL** is a closed tableau whose initial node is $\{\mathbf{F}B\}$. Finally, a set of swffs S is **IL**-*consistent* iff no tableau starting from S is closed.

We remark that all the rules of the **IL**-T calculus, excepting $\mathbf{T}\forall$, $\mathbf{F}\exists$, $\mathbf{F_c}\exists$, $\mathbf{F_c}\forall$ and $\mathbf{T} \rightarrow \forall$, are duplication free, in the sense explained in [8, 15, 16, 3, 5, 17].

In [15, 16, 17], it is proven that **IL**-T is sound and complete according to the standard intuitionistic semantics based on Kripke models. Henceforth, the **IL**-T calculus is equivalent to the **IL** calculus.

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

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

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

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

$$\frac{S, \mathbf{T}(\forall\, x.\, A(x))}{S, \mathbf{T}A(t), \mathbf{T}(\forall\, x.\, A(x))}\ \mathbf{T}\forall \qquad \frac{S, \mathbf{F}(\forall\, x.\, A(x))}{S_c, \mathbf{F}A(a)}\ \mathbf{F}\forall$$

$$\frac{S, \mathbf{T}(\exists\, x.\, A(x))}{S, \mathbf{T}A(a)}\ \mathbf{T}\exists \qquad \frac{S, \mathbf{F}(\exists\, x.\, A(x))}{S, \mathbf{F}A(t)}\ \mathbf{F}\exists$$

$$\frac{S, \mathbf{F_c}(\forall\, x.\, A(x))}{S_c, \mathbf{F_c}A(a), \mathbf{F_c}(\forall\, x.\, A(x))}\ \mathbf{F_c}\forall \qquad \frac{S, \mathbf{F_c}(\exists\, x.\, A(x))}{S, \mathbf{F_c}A(t), \mathbf{F_c}(\exists\, x.\, A(x))}\ \mathbf{F_c}\exists$$

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

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

$$\frac{S, \mathbf{T}((\exists\, x.\, A(x)) \to B)}{S, \mathbf{T}(\forall\, x.\, (A(x) \to B))}\ \mathbf{T}{\to}\exists$$

$$\frac{S, \mathbf{T}((\forall\, x.\, A(x)) \to B)}{S, \mathbf{F}(\forall\, x.\, A(x)), \mathbf{T}((\forall\, x.\, A(x)) \to B) \mid S, \mathbf{T}B}\ \mathbf{T}{\to}\forall$$

where t is a term, a is a new variable, and P is an atomic or negated wff.

TABLE 2. The **IL**-T calculus.

3 Architecture of the Prover

The architecture of the proving system we are describing is composed by: Isabelle/HOL [23], the classical higher-order logic developed as an Isabelle theory, and a tactic which takes as input the number of the subgoal it has to prove, and which returns void with the side effect of canceling the selected subgoal, if it proves it, otherwise it generates an exception which signals that the tactic failed.

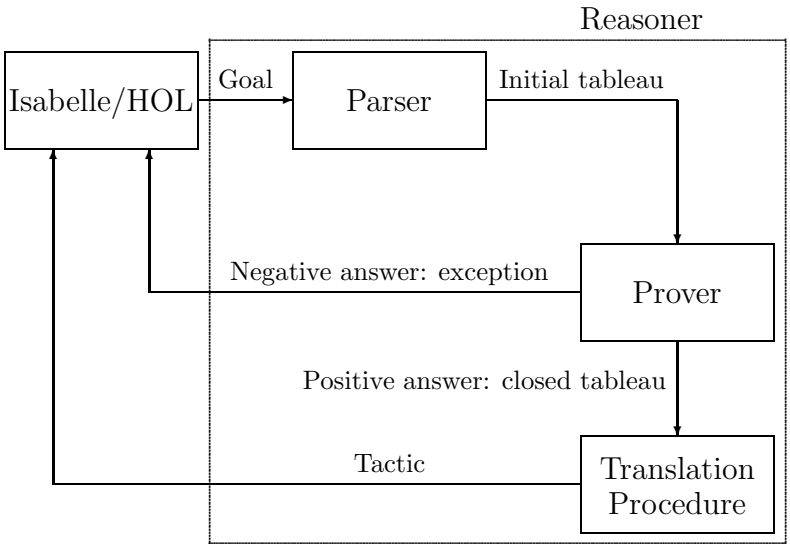


Fig. 1. The architecture of the prover.

Internally, the architecture of the reasoner is depicted in Figure 1. First, the selected subgoal is transformed into the internal syntax of the prover, constructing an initial tableau. Then, the theorem prover tries to close that tableau following the built-in strategy.

There are two possible outcomes:

1. a closed tableau is generated;
2. the prover is not able to generate a closed tableau.

In the second case, the prover raises an ML exception [19, 21], which signals the failure in proving the goal, and the control returns to the user. It has be be remarked that, if a proof for the goal exists, this behaviour shows that the reasoner is not complete, since it is unable to prove the goal, but it does not signal that the reasoner is not correct. On the other hand, if the goal is not a theorem, the behaviour of the prover is correct, not being able to prove it. Thus, in any case, when a negative answer is generated, the correctness of the proving system is preserved.

In the first case, a closed proof-table is generated, thus the prover believes that the goal is true. But, in our assumption, the prover is not trustable, so a check on the proof must be performed. Consequently, the closed tableau is passed to a translation procedure, whose details are described in the following section, which generates an

independent Isabelle tactic, that, using only the basic inference rules of the logical framework, performs the same inference steps as the ones necessary to generate the closed tableau. Finally, the generated tactic is executed on the subgoal, and its outcome becomes the answer the reasoner gives to the user.

A first analysis of the described behaviour is needed: the goal is to ensure that the answer of the proving system is correct, even if some of its components may be wrongly implemented. As remarked before, the proposed technique sacrifices completeness to meet this goal.

In particular, the possible cases are summarized in Table 3.

	Parser	Prover	Translation Procedure
1	Correct	Correct	Correct
2	Correct	Correct	Wrong
3	Correct	Wrong	Correct
4	Correct	Wrong	Wrong
5	Wrong	Correct	Correct
6	Wrong	Correct	Wrong
7	Wrong	Wrong	Correct
8	Wrong	Wrong	Wrong

Table 3. The possible cases in the completeness analysis.

The cases 5 to 8 are characterized by the fact that the parser is wrongly implemented; thus, the initial tableau will not correspond to the goal, and the consequence is that it is almost impossible for the tableau tactic[1] to produce a positive answer, since the generated closed tableau, if any, will not be translated in a tactic which is able to solve the original goal. As a matter of fact, it is quite difficult to build an entirely wrong parser, because, at least in the **IL**-T calculus, the Isabelle format for formulas and terms is very close to the internal representation. In fact, the internal representation simply discharges information on types, and decorates formulas with the $\mathbf{T}$, $\mathbf{F}$ and $\mathbf{F_c}$ signs.

More interesting are the cases 3 and 4, where the prover is wrong. The interesting possibilities, given a goal G, which correctly gets translated into a initial tableau by the parser, are that G is a theorem and the prover produces a negative answer, and that G is not a theorem but the prover produces a positive answer. The former possibility does not change the correctness character of the whole tableau tactic, but just its completeness. The latter possibility transfers the responsibility of discarding the wrong answer to the translation procedure. Independently from the correctness of the translation procedure, an Isabelle tactic which behaves like the closed tableau is generated and then executed. This tactic does not make use of the reasoner, but it deterministically reproves the goal G applying valid inference rules in the logical framework environment, and, since G is not a theorem, we are guaranteed by the correctness of the logical framework core that no combination of valid rules can prove G, since it is not a theorem. Henceforth, *independently from the correctness of the*

[1] Since the proving system we use as an example is implemented as a tableau tactic, we will use the expressions *proving system*, *reasoner* and *tableau tactic* as synonyms.

translation procedure, the goal G receives a negative answer.

Of course, case 1 is trivial: every component being correct, any answer from the prover gets certified.

The most critical case is the second one: in fact, every answer from the prover is, by assumption, correct. If the prover gives a negative answer, the whole tableau tactic gives a negative answer, thus preserving correctness. If the prover produces a positive answer, the translation procedure may produce an ISABELLE tactic which does not correspond to the closed tableau which proves the goal. When this tactic gets applied, ISABELLE will fail somewhere, and being the ISABELLE tactic deterministic, the failure is the final result of the computation of the tableau tactic, thus a theorem gets discarded. Of course, this fact amounts to introduce an incompleteness in the tableau tactic.

Summarizing, even if the implementation of the components may be wrong, it is never the case that a false wff gets accepted as a theorem, hence it follows that the tableau tactic is correct; on the other hand, the correctness of the components affect the tableau tactic completeness, since it may give negative answers when proving a goal which is, indeed, a theorem.

Henceforth, the technique we are proposing, *validation by translation*, ensures that the tableau tactic is correct in any case.

An interesting development of the technique, which puts it in evidence with respect to other auto-validation techniques, is the possibility to adopt as the internal calculus of the prover, a non sound variant of a known calculus. For example, in the tableau reasoner, a prover which uses the **IL**-T calculus not checking for eigenvariables in the application of the tableau rules has been implemented. The precise results of this experiment are discussed later in this paper, but some consequences can be drawn immediately.

A prover based on a non-sound calculus may yield three results: a negative answer whose meaning is *the prover is not able to prove the goal*; a positive answer together with a proof which is valid in the sound calculus; a positive answer together with an invalid proof. The first case determines a *structural incompleteness* when the goal is a theorem: in fact, the prover is not able to prove a provably true statement, thus it is incomplete because its algorithm is so. The second case, because of the presence of the translation procedure, yields a positive answer of the reasoner, since the proof is valid in the sound version of the calculus.

The third case, the interesting one, fails the checking performed by the translation procedure, because the prover result is not a sound proof, thus a negative answer is generated.

The consequence is that the whole reasoner is sound, since it produces positive answers only on theorems, but the third case introduces another kind of incompleteness we call *methodological*. In fact, it amounts to say that the prover correctly produces a proof, but the proof itself is non acceptable, thus the incompleteness is given by the method the prover adopts; it is possible that, with further search in the proof-space of the prover, a valid proof may be found when the goal is indeed a theorem, but the method, that is, the combination of the calculus and the search strategy, is not complete.

There is an important difference in the quality between the concepts of structural

and methodological incompleteness, since the former arises because of the search strategy alone, due to a faulty prover, or due to a incomplete searching procedure, while the latter incompleteness is caused by the presence of non-sound inference rules, which prevent the finding of a correct proof in the sound calculus, even if such a proof may be produced in the non-sound variant.

4 Validation of the Tableau Tactic

In this section, the formal development of the translation procedure is presented, as well as the mathematics underlying its applicability. The discussion has been specialized for the example we adopted as a case study through the whole paper, although the same schema can be replicated for other logical systems, see, e.g. [6]. A preliminary development of the theory shown here has already been documented in [1, 2], where the core results, needed to establish the schema which leads to the development of an adequate translation procedure are derived. With respect to those works, the actual presentation tries to be less specialized versus a peculiar logical theory, and it integrates a comparison with the embedding technique on a complexity basis.

A convenient notation to indicate a part of a natural deduction proof which has not yet been developed is the notion of *gap*: a gap has the form of an inference rule

$$\frac{\gamma_1 \cdots \gamma_n}{\alpha}\ \mathrm{G}$$

where α, γ_1, ... , γ_n are wffs; the meaning of a gap is that of a placeholder for a derivation of α from the assumptions γ_1, ... , γ_n.

The convenience of the gap notation comes from the fact that a gap $G \equiv \dfrac{\Gamma}{\alpha}\ \mathrm{G}$ occurring in a proof $\Pi(G)$ may be substituted by a proof $\begin{matrix}\Gamma\\ \vdots\\ \alpha\end{matrix}$; more precisely, $\Pi\left(\begin{matrix}\Gamma\\ \vdots\\ \alpha\end{matrix}\right)$ is the proof $\Pi(G)$ where the indicated occurrence of the gap G has been replaced by the proof $\begin{matrix}\Gamma\\ \vdots\\ \alpha\end{matrix}$, eventually with a change in the names of eigenvariables to prevent variable capturing.

The very first step in the construction of a translation procedure is the definition of a calculus which forms the natural deduction counterpart of the internal calculus of the prover.

In the case of the tableau tactic, this image of **IL**-T is given by the **ILD** calculus, defined as follows:

Definition 4.1 The **ILD** calculus is defined by the inference rules in Tables 4, 5 and 6. In particular the rules in Table 5 are referred to as the *implicative* rules, while the rules in Table 6 are called the *closure* rules.

In order to guarantee that **ILD** is the image of **IL**-T in **IL** via the translation procedure, it is necessary to prove that the set of theorems of **ILD** is contained in the set of theorems of **IL**, and it is necessary to prove that the map between proofs of **IL**-T and proofs of **ILD** as defined by the translation procedure is correct.

$$\frac{A \wedge B \qquad \dfrac{\Gamma, [A], [B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}\wedge \qquad \frac{\dfrac{\Gamma}{A}\,\mathrm{G} \quad \dfrac{\Gamma}{B}\,\mathrm{G}}{A \wedge B}\,\mathbf{F}\wedge_R$$

$$\frac{\dfrac{\Gamma}{D \vee A}\,\mathrm{G} \quad \dfrac{\Gamma}{D \vee B}\,\mathrm{G}}{D \vee (A \wedge B)}\,\mathbf{F}\wedge \qquad \frac{\neg(A \wedge B) \qquad \dfrac{\Gamma, [\neg A]}{\bot}\,\mathrm{G} \quad \dfrac{\Gamma, [\neg B]}{\bot}\,\mathrm{G}}{D}\,\mathbf{F_c}\wedge$$

$$\frac{A \vee B \qquad \dfrac{\Gamma, [A]}{D}\,\mathrm{G} \quad \dfrac{\Gamma, [B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}\vee \qquad \frac{\neg(A \vee B) \qquad \dfrac{\Gamma, [\neg A], [\neg B]}{D}\,\mathrm{G}}{D}\,\mathbf{F_c}\vee$$

$$\frac{\dfrac{\Gamma, [A]}{\bot}\,\mathrm{G}}{D \vee \neg A}\,\mathbf{F}\neg \qquad \frac{\dfrac{\Gamma, [A]}{\bot}\,\mathrm{G}}{\neg A}\,\mathbf{F}\neg_R \qquad \frac{\neg\neg A \qquad \dfrac{\Gamma, [A]}{\bot}\,\mathrm{G}}{D}\,\mathbf{F_c}\neg$$

$$\frac{\exists\, x.\, A(x) \qquad \dfrac{\Gamma, [A(p)]}{D}\,\mathrm{G}}{D}\,\mathbf{T}\exists\ (*) \qquad \frac{\neg\exists\, x.\, A(x) \qquad \dfrac{\Gamma, [\neg A(a)], [\neg\exists\, x.\, A(x)]}{D}\,\mathrm{G}}{D}\,\mathbf{F_c}\exists$$

$$\frac{\dfrac{\Gamma}{D \vee A(a)}\,\mathrm{G}}{D \vee \exists\, x.\, A(x)}\,\mathbf{F}\exists \qquad \frac{\dfrac{\Gamma}{A(a)}\,\mathrm{G}}{\exists\, x.\, A(x)}\,\mathbf{F}\exists_R$$

$$\frac{\forall\, x.\, A(x) \qquad \dfrac{\Gamma, [A(a)], [\forall\, x.\, A(x)]}{D}\,\mathrm{G}}{D}\,\mathbf{T}\forall \qquad \frac{\neg\forall\, x.\, A(x) \qquad \dfrac{\Gamma, [\neg\forall\, x.\, A(x)]}{A(p)}\,\mathrm{G}}{D}\,\mathbf{F_c}\forall\ (*)$$

$$\frac{\dfrac{\Gamma}{A(p)}\,\mathrm{G}}{D \vee \forall\, x.\, A(x)}\,\mathbf{F}\forall\ (*) \qquad \frac{\dfrac{\Gamma}{A(p)}\,\mathrm{G}}{\forall\, x.\, A(x)}\,\mathbf{F}\forall_R\ (*)$$

where a is any term and, in $(*)$, p is an eigenvariable.

TABLE 4. Inference rules of **ILD**, part I.

$$\frac{\dfrac{\Gamma,[A]}{B}\,\mathrm{G}}{D\vee(A\to B)}\,\mathbf{F}{\to} \qquad \frac{A\vee B\to C \quad \dfrac{\Gamma,[A\to C],[B\to C]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}{\vee}$$

$$\frac{\dfrac{\Gamma,[A]}{B}\,\mathrm{G}}{A\to B}\,\mathbf{F}{\to_R} \qquad \frac{(\exists\, x.\, A(x))\to B \quad \dfrac{\Gamma,[\exists\, x.\, A(x)\to B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}{\exists}$$

$$\frac{\neg(A\to B) \quad \dfrac{\Gamma,[A],[\neg B]}{\bot}\,\mathrm{G}}{D}\,\mathbf{F_c}{\to} \qquad \frac{A\wedge B\to C \quad \dfrac{\Gamma,[A\to(B\to C)]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}{\wedge}$$

$$\frac{A\to B \quad \dfrac{\Gamma}{D\vee A}\,\mathrm{G} \quad \dfrac{\Gamma,[B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}(*) \qquad \frac{A\to B \quad \dfrac{\Gamma}{A}\,\mathrm{G} \quad \dfrac{\Gamma,[B]}{\bot}\,\mathrm{G}}{\bot}\,\mathbf{T}{\to_R}(*)$$

$$\frac{(A\to B)\to C \quad \dfrac{\Gamma,[B\to C]}{D\vee(A\to B)}\,\mathrm{G} \quad \dfrac{\Gamma,[C]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}{\to}$$

$$\frac{(A\to B)\to C \quad \dfrac{\Gamma,[B\to C]}{A\to B}\,\mathrm{G} \quad \dfrac{\Gamma,[C]}{\bot}\,\mathrm{G}}{\bot}\,\mathbf{T}{\to}{\to_R}$$

$$\frac{(\forall\, x.\, A(x))\to B \quad \dfrac{\Gamma,[(\forall\, x.\, A(x))\to B]}{D\vee(\forall\, x.\, A(x))}\,\mathrm{G} \quad \dfrac{\Gamma,[B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}{\to}{\forall}$$

$$\frac{(\forall\, x.\, A(x))\to B \quad \dfrac{\Gamma,[(\forall\, x.\, A(x))\to B]}{\forall\, x.\, A(x)}\,\mathrm{G} \quad \dfrac{\Gamma,[B]}{\bot}\,\mathrm{G}}{\bot}\,\mathbf{T}{\to}{\forall_R}$$

where, in $(*)$, A is an atomic or a negated wff.

TABLE 5. Inference rules of **ILD**, part II.

$$\frac{A}{D \vee A}\ \mathbf{F}\text{-closure} \qquad \frac{A\ \neg A}{D}\ \mathbf{F_c}\text{-closure}$$

TABLE 6. Inference rules of **ILD**, part III.

The first step follows from the lemma:

Lemma 4.2 The inference rules of the **ILD** calculus can be derived in **IL**.

PROOF. For the sake of brevity, we treat only the inference rules related to the conjunction, $\mathbf{T}\wedge$, $\mathbf{F}\wedge$, $\mathbf{F}\wedge_R$ and $\mathbf{F_c}\wedge$; the other cases are similar.

- $\dfrac{A \wedge B \qquad \dfrac{\Gamma, [A], [B]}{D}\,\mathrm{G}}{D}\,\mathbf{T}\wedge$:

  if $\begin{array}{c}\Gamma, A, B\\ \vdots\\ D\end{array}$ is a proof in **IL** of $\Gamma, A, B \vdash D$, then

  $$\Gamma\ \dfrac{\dfrac{A \wedge B}{A}\,\wedge\mathrm{E} \qquad \dfrac{A \wedge B}{B}\,\wedge\mathrm{E}}{\begin{array}{c}\vdots\\ D\end{array}}$$

  is the **IL** proof schema corresponding to the $\mathbf{T}\wedge$ rule.

- $\dfrac{\dfrac{\Gamma}{D \vee A}\,\mathrm{G} \qquad \dfrac{\Gamma}{D \vee B}\,\mathrm{G}}{D \vee (A \wedge B)}\,\mathbf{F}\wedge$:

  if $\begin{array}{c}\Gamma\\ \vdots\\ D \vee A\end{array}$ and $\begin{array}{c}\Gamma\\ \vdots\\ D \vee B\end{array}$ are **IL**-proofs of $\Gamma \vdash D \vee A$ and $\Gamma \vdash D \vee B$, respectively, then the following proof schema in the **IL** calculus simulates the $\mathbf{F}\wedge$ rule:

  $$\dfrac{\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ D \vee A\end{array} \qquad \dfrac{[D]}{D \vee (A \wedge B)}\,\vee\mathrm{I} \qquad \dfrac{\begin{array}{c}\Gamma\\ \vdots\\ D \vee B\end{array} \qquad \dfrac{[D]}{D \vee (A \wedge B)}\,\vee\mathrm{I} \qquad \dfrac{\dfrac{[A]\ [B]}{A \wedge B}\,\wedge\mathrm{I}}{D \vee (A \wedge B)}\,\vee\mathrm{I}}{D \vee (A \wedge B)}\,\vee\mathrm{E}}{D \vee (A \wedge B)}\,\vee\mathrm{E}$$

- $\dfrac{\dfrac{\Gamma}{A}\,\mathrm{G} \qquad \dfrac{\Gamma}{B}\,\mathrm{G}}{A \wedge B}\,\mathbf{F}\wedge_R$:

if $\begin{array}{c}\Gamma\\ \vdots\\ A\end{array}$ and $\begin{array}{c}\Gamma\\ \vdots\\ B\end{array}$ are **IL**-proofs of $\Gamma \vdash A$ and $\Gamma \vdash B$, then the the $\mathbf{F}\wedge_R$ rule is simulated in **IL** by the following proof schema:

$$\dfrac{\begin{array}{cc}\Gamma & \Gamma\\ \vdots & \vdots\\ A & B\end{array}}{A \wedge B}\,\wedge\mathrm{I}$$

- $\dfrac{\neg(A \wedge B) \qquad \dfrac{\Gamma, [\neg A]}{\bot}\,\mathrm{G} \qquad \dfrac{\Gamma, [\neg B]}{\bot}\,\mathrm{G}}{D}\,\mathbf{F_c}\wedge$:

if $\begin{array}{c}\Gamma, \neg A\\ \vdots\\ \bot\end{array}$ and $\begin{array}{c}\Gamma, \neg B\\ \vdots\\ \bot\end{array}$ are **IL**-proofs of the sequents $\Gamma, \neg A \vdash \bot$ and $\Gamma, \neg B \vdash \bot$, respectively, then the $\mathbf{F_c}\wedge$ rule is simulated in **IL** by the following proof schema:

$$\dfrac{\Gamma, \dfrac{\dfrac{\dfrac{[A]\ [B]}{A \wedge B}\,\wedge\mathrm{I} \qquad \neg(A \wedge B)}{\bot}\,\to\mathrm{E}}{\neg B}\,\to\mathrm{I}}{\begin{array}{c}\vdots\\ \bot\end{array}}\,\to\mathrm{I}$$

$$\dfrac{\Gamma, \dfrac{\neg A}{\begin{array}{c}\vdots\\ \bot\end{array}}\,\to\mathrm{I}}{D}\,\bot\mathrm{E}$$

since, in the **IL** calculus, the negated wff $\neg A$ is considered as an abbreviation for $A \to \bot$. □

■

Lemma 4.2 naturally suggests to encode the **ILD** calculus in Isabelle as a set of derived inference rules, and to use them as a basis for a tactic which mimics the tableau prover. In fact, that will be an embedding of the tableau prover into Isabelle.

In this respect, as remarked in the introduction, the advantage will be to avoid a check for correctness of the prover, since the soundness of the composition of inference rules is guaranteed by the Isabelle core.

On the other hand, it has been said that an embedding requires more memory space to search for a proof. The evidence of this statement comes from the following calculations; in the hypothesis that both the tableau prover and the embedded prover adopt the same strategy, they produce the same number of subgoals, the former as a sequence of configurations, the latter as a sequence of proof-states.

The memory size of a configuration can be measured: being

$$C \equiv S_1^1\phi_1^1, \ldots, S_{n_1}^1\phi_{n_1}^1 \mid \ldots \mid S_1^m\phi_1^m, \ldots, S_{n_m}^m\phi_{n_m}^m$$

a configuration where S_j^i is a sign, $\mathbf{T}$, $\mathbf{F}$ or $\mathbf{F_c}$, and ϕ_j^i a wff, the size of C, $|C|$ is roughly

$$K_C \sum_{i=1}^{m} \left(K_N \sum_{j=1}^{n_m} \left|\phi_j^i\right| \right) = K \sum_{i=1}^{m} \sum_{j=1}^{n_m} \left|\phi_j^i\right|$$

where $\left|\phi_j^i\right|$ is the memory size of the representation of the wff ϕ_j^i, $K = K_C K_N$, and K_C is the size of the memory block used to describe an element of a list, and finally K_N is the size of the block used to represent an element of a list plus the size to represent a sign.

The constant K_C and K_N can be roughly estimated to be 30 bytes, from empirical measurings. This estimation is rough due to the interference of the garbage collection system[2] of the ML runtime [21].

Moreover, being $d(\phi)$ the depth of the tree representation of a wff, $|\phi| \leq K_F 2^{d(\phi)}$, where K_F is the size of the memory block which represents a symbol in the formula. Hence, the size of a configuration C as above, where d is the maximal depth of a wff occurring in C, and n is the maximum in the set $\{n_1, \ldots, n_m\}$ is less than $nmK_CK_NK_F2^d \approx mn2^{d+1}$ Kbytes.

looking at the internals of ISABELLE, a similar calculation can be performed; for the same configuration, represented as a proof-state, one gets an upper bound whose value is $mnK_CK_NK_F2^{2d} \approx 100mn2^{2d+1}$ Kbytes. Hence, from an empirical evaluation, the tableau representation adopted in the tableau prover is $2^{-d}/100$ smaller, in the worst case, than the corresponding ISABELLE representation. On the average, the ratio seems to be around $2^{-d}/10$, but this estimation may depend on the particular set of test cases[3].

The translation procedure is applied to a tableau, that is, to a finite sequence of configurations $C_1, \ldots, C_n$. Every configuration C_i is a list of nodes $S_1 \mid \ldots \mid S_n$.

The initial configuration $C_1 = S$ corresponds to a subgoal which has the form of a gap $\dfrac{\Gamma}{D}\,\mathrm{G}$, with $\Gamma = \{\phi \mid \mathbf{T}\phi \in S\} \cup \{\neg\phi \mid \mathbf{F_c}\phi \in S\}$ and $D = \bigvee\{\phi \mid \mathbf{F}\phi \in S\}$.

Any successive configuration $C_{i+1} = S_1 \mid \ldots \mid S_m \mid N_1 \mid \ldots \mid N_k$, generated from $C_i = S_1 \mid \ldots \mid S_m \mid M$ applying the inference rule R whose instance has the form $\dfrac{M}{N_1 \mid \ldots \mid N_k}\,R$, corresponds to a series of subgoals which have the form of gaps: the nodes $S_1, \ldots, S_m$ correspond to the gaps as inductively defined by C_i, whereas the nodes $N_1, \ldots, N_k$ are generated by the application of the inference rule R in the **ILD** calculus to the gap associated with M. The application of the R rule takes place as follows:

1. Being $M = M' \cup \{A\}$ where A is the active swff, $\Gamma = \{\phi \mid \mathbf{T}\phi \in M'\} \cup \{\neg\phi \mid \mathbf{F_c}\phi \in M'\}$, and $D = \bigvee\{\phi \mid \mathbf{F}\phi \in M'\}$;
2. If $\{\phi \mid \mathbf{F}\phi \in M'\} = \emptyset$ then $D = \bot$;

[2] The measurings have been performed on the New Jersey Standard ML system, and the values we are reporting may differ from one implementation to another, but the order of magnitude should be invariant.

[3] Due to the extremely non-smooth nature of the proof-space of a theorem prover, it is very hard to identify a good set of test cases, especially when dealing with non-classical logics. The historical test cases, such as [25], have been considered.

3. If R is $\mathbf{F}\vee$ or $\mathbf{T}\neg$ then $k = 1$ and $N_1 = M$;
4. If R is an $\mathbf{F}$-rule different from $\mathbf{F}\vee$, and $\{\phi \mid \mathbf{F}\phi \in M'\} = \emptyset$, then the rule in the **ILD** calculus is the corresponding restricted version, e.g., $\mathbf{F}\neg_R$ instead of $\mathbf{F}\neg$; otherwise the rule R of **ILD** is applied.

The translation procedure, Tr for short, takes a tableau T as input and produces a deterministic sequence of pairs of tactics F_i `THEN` D_i where F_i succeeds when applied to a goal which has the form described above for the expanded node in the C_i configuration, that is, the set of assumptions is Γ and the conclusion is a disjunction equivalent to D modulo reordering of the disjuncts; in addition, D_i is a complete instance of the resolution tactic of ISABELLE [20] with the **ILD** inference rule that has to be applied according to the previous description.

The result of the application in ISABELLE of the output tactic, that is, the sequential composition of the previously described pairs, is a proof where the open subgoals left after its completion are exactly the ones corresponding to the terminal configuration of the tableau, as proven in the following proposition.

Proposition 4.3 Let T be a tableau; for any gap G generated by the application of rules as described for $\mathrm{Tr}(T)$, there is a node S in the terminal configuration of T such that G is associated with S and

$$G \equiv \frac{\{\phi \mid \mathbf{T}\phi \in S\} \cup \{\neg\phi \mid \mathbf{F_c}\phi \in S\}}{\bigvee\{\phi \mid \mathbf{F}\phi \in S\}}\ \mathrm{G}\ .$$

PROOF. By induction on the structure of the tableau T. The base case is immediate by definition, while the induction step is easily solved by inspecting the rules of **ILD**. □

One may think that the translation procedure Tr generates a proof in the **ILD** calculus which is equivalent to the tableau given as input; this view is correct except for the presence of gaps. But gaps can be removed if a tableau is closed; in fact, the translation procedure is complemented by a gap closure procedure defined as follows.

Given a configuration $C = S_1 \mid \ldots \mid S_k$ where the node S_i is closed, then the gap associated with S_i is removed according to the closure type of S_i:

1. If $\mathbf{T}A \in S_i$ and $\mathbf{F}A \in S_i$, meaning that S_i is $\mathbf{F}$-closed, then the gap corresponding to S_i has the form
$$\frac{A, A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m}{C_1 \vee \ldots \vee C_l \vee A}\ \mathrm{G}$$
and it is substituted by an instance of the $\mathbf{F}$-closure rule:
$$\frac{A}{C_1 \vee \ldots \vee C_l \vee A}\ \mathbf{F}\text{-}closure\ ;$$
of course, if $l = 0$, the gap is closed by the assumption rule.
2. If $\mathbf{T}A \in S_i$ and $\mathbf{F_c}A \in S_i$, so that S_i is $\mathbf{F_c}$-closed, then the gap corresponding to S_i has the form
$$\frac{A, A_1, \ldots, A_n, \neg A, \neg B_1, \ldots, \neg B_m}{D}\ \mathrm{G}$$

and it is substituted by an instance of the $\mathbf{F_c}$-closure rule:

$$\frac{A \; \neg A}{D} \; \mathbf{F_c}-closure \; .$$

The gap closure procedure is applied to a closed tableau T and produces as output a tactic which closes all the gaps in the terminal configuration of T. The tactic is a sequence of steps of the form F_i `THEN` D_i where F_i is a tactic which succeeds iff the subgoal has the form of the gap associated to the node S_i in the terminal configuration of T, and D_i is the instance of a closure rule of the **ILD** calculus as described above.

An important fact to remark is that both tactics are deterministic; they do not require backtracking[4], but simply they succeeds or fail. In this sense the outcome of the whole translation procedure is really a validation tactic, since it does not involve any form of searching in the proof-space of the logical framework.

Theorem 4.4 Let T be a closed tableau, then, calling D the tactic generated by the translation procedure, and C the tactic generated by the gap closure procedure, the tactic D `THEN` C, when applied to ISABELLE goal which produces T, solves it.

PROOF. The D tactic, as proven in Proposition 4.3, generates a proof in the **ILD** calculus which has as many gaps as the number of nodes in the terminal configuration of T; any step of the C tactic eliminates one gap, and the number of steps correspond to the number of nodes in the terminal configuration of T, henceforth the sequential application of D and C is successful, leaving no open subgoals. □

It is useful to remark that the translation procedure and the gap closure procedure may be thought as translating closed tableaus into **ILD** proofs. In this sense an immediate corollary of the preceding theorem is

Corollary 4.5 **IL**-T $\equiv$ **ILD** $\equiv$ **IL**.

PROOF. The equivalence of **IL**-T and **IL** is immediate since both calculi are sound and complete presentations of the intuitionistic logic, thus they prove the same set of theorems.

In Lemma 4.2, it has been shown that every theorem of **ILD** is a theorem of **IL**, while in Theorem 4.4 it has been shown that every proof in **IL**-T can be translated into an **ILD** proof, thus every theorem of **IL**-T is a theorem of **ILD**.

Hence, by the equivalence of **IL**-T and **IL**, it follows that **IL**-T $\equiv$ **ILD** and **ILD** $\equiv$ **IL**. □

The construction of the complete translation procedure, that is, the composition of the translation and the gap closure procedures, can be easily extended to many other logics [6].

[4] In fact, the `THEN` operator is completely deterministic, as documented in [24].

5 Non-Sound Tableau Calculi

Since, as discussed at length in [16, 15, 17], **IL**-T is duplication free in its propositional fragment, and it admits a decision procedure on the same fragment which is PSPACE-complete [14], there is little space for improving the performances of the prover on this fragment. On the contrary, the predicative part is not, and it cannot be, duplication free; in order to minimize the amount of duplications, the heuristics is to delay the choice of witnesses as much as possible. The extreme solution is to delay this choice until a node in the tableau has to be closed. Following this approach, the **IL**-TDummy calculus is defined [27].

Let Ξ be a denumerable set of symbols, called *dummy variables*, and let Ξ be disjoint from ordinary variables, constants and logical symbols of **IL**-T.

Definition 5.1 The **IL**-*TDummy* calculus is the tableau calculus defined from the rules in Table 2, where the $\mathbf{T}\forall$, $\mathbf{F}\exists$ and $\mathbf{F_c}\exists$ rules are replaced by

$$\frac{S, \mathbf{T}(\forall\, x.\, A(x))}{S, \mathbf{T}A(\alpha), \mathbf{T}(\forall\, x.\, A(x))}\ \mathbf{T}\forall_{\text{Dummy}}$$

$$\frac{S, \mathbf{F}(\exists\, x.\, A(x))}{S, \mathbf{F}A(\alpha)}\ \mathbf{F}\exists_{\text{Dummy}}$$

$$\frac{S, \mathbf{F_c}(\exists\, x.\, A(x))}{S, \mathbf{F_c}A(\alpha), \mathbf{F_c}(\exists\, x.\, A(x))}\ \mathbf{F_c}\exists_{\text{Dummy}}$$

where $\alpha \in \Xi$.

Definition 5.2 A node N in an **IL**-TDummy tableau T is *closed* iff one of the following condition applies:

- $\mathbf{T}A \in N$ and $\mathbf{F}B \in N$ and there is a substitution θ of dummy variables such that $\theta A \equiv \theta B$.
- $\mathbf{T}A \in N$ and $\mathbf{F_c}B \in N$ and there is a substitution θ of dummy variables such that $\theta A \equiv \theta B$.

Definition 5.3 An **IL**-TDummy tableau T is *closed* iff there exists a substitution θ of dummy variables such that every node N in the terminal configuration of T is closed by θ.

It is immediate to verify that every **IL**-T tableau can be converted into an equivalent **IL**-TDummy tableau whose the closing substitution maps every dummy variable introduced by a $\mathbf{T}\forall_{\text{Dummy}}$, $\mathbf{F}\exists_{\text{Dummy}}$ or $\mathbf{F_c}\exists_{\text{Dummy}}$ rule into a corresponding term generated by a $\mathbf{T}\forall$, $\mathbf{F}\exists$ or $\mathbf{F_c}\exists$ rule, respectively. This fact establishes **IL**-TDummy as a variant of the **IL**-T calculus.

The **IL**-TDummy calculus is not sound with respect to the standard Kripke semantics of intuitionistic logic, in fact the sequent $\forall\, x.\, x = x \vdash \exists\, x.\, \forall\, y.\, y = x$ is not true in

1. Being C the terminal configuration of a tableau T, choose a node N in C which is not marked as closed; if this is not possible, then exit with success, being T a closed tableau.
2. Check if N is closed; if the check is positive, mark N as closed and return to step 1.
3. Choose a swff $\mathcal{S}\phi$ in N which can be expanded by an inference rule of the tableau calculus; if this is not possible then backtrack.
4. Remember the state of choices so far ($\mathcal{S}\phi$, N, C) for backtracking.
5. If $\mathcal{S}\phi$ is not of the form $\mathbf{T}\forall$, $\mathbf{F}\exists$, $\mathbf{F_c}\exists$ then apply the inference rule whose active swff is $\mathcal{S}\phi$ to N, and build an extension T' of the tableau T. Then return to step 1.
6. If $\mathcal{S}\phi$ is of the form $\mathbf{T}\forall$, $\mathbf{F}\exists$, $\mathbf{F_c}\exists$
   - in the TabDTac prover, apply the corresponding inference rule where the dummy variable is new.
   - in the TabTac prover, choose a term t and apply the corresponding inference rule with that term as a witness; mark the choice of t for backtracking.
7. Go to step 1.
8. If a backtracking invocation in step 3 cannot be satisfied because there are no alternatives, exit with failure.
9. To ensure termination, do not duplicate swffs in the inference rules which require duplication more than D times, where D is a fixed parameter.

FIG. 2. Procedural description of TabTac and TabDTac.

IL, even if the $=$ relation is not interpreted as the equality. But **IL**-TDummy proves this sequent:

$$\dfrac{\dfrac{\dfrac{\mathbf{F}\exists\, x.\forall\, y.\, y = x, \mathbf{T}\forall\, x.\, x = x}{\mathbf{F}\forall\, y.\, y = \alpha, \mathbf{T}\forall\, x.\, x = x}\,\mathbf{F}\exists_{\text{Dummy}}}{\mathbf{F}z = \alpha, \mathbf{T}\forall\, x.\, x = x}\,\mathbf{F}\forall}{\mathbf{F}z = \alpha, \mathbf{T}\beta = \beta, \mathbf{T}\forall\, x.\, x = x}\,\mathbf{T}\forall_{\text{Dummy}}$$

closed by $\alpha = \beta = z$

The combination of a prover based on the **IL**-TDummy calculus with the translation procedure as described in the previous section leads to a sound tactic, *TabDTac*, which pays the non-sound nature of its prover by being incomplete.

In order to analyze the behaviour of TabDTac with respect to TabTac, it is necessary to fix a strategy that both tactics adopt, that is, a common way to navigate their proof-spaces.

The precise description of the behaviour of the provers in TabTac and TabDTac is depicted in Figure 2. Some remarks are needed:

- both provers terminate on every input; step 9 ensures this property.

- in step 6, the choice for a term t is not arbitrary; it can be either a new variable or a term which already occurs in the node N. This restriction comes from the completeness theorem for **IL**-T.
- both provers adopt the same strategy, that is, the same algorithm to choose a node in a given tableau, and a swff in a given node.
- if a fair strategy is adopted, then the management of backtracking combined with the limit to duplications ensures that both provers are D-complete, that is, if an initial tableau can be refuted using at most D duplicating rules, then it will be refuted by the provers.

The ultimate purpose of this section is to make evident that a prover based on a non-sound calculus can be more efficient on some goals than a prover based on a sound calculus.

The implementations of TabTac and TabDTac make use of a fair strategy, defined as follows:

- in step 1, the first node not marked as closed in the terminal configuration of T is chosen.
- in step 3, the first swff which may be expanded, is chosen.
- in step 5, the expansion of $\mathcal{S}\phi$ replaces the swff in the newly generated nodes.
- in step 6, the replacement of a swff takes place as in step 5; in the case of TabTac the choice of a witness term t is fixed as the first term occurring in the node; if no term exists, then a new variable is generated. By backtracking, every term occurring in the node will eventually be chosen, and, at last, a new variable will be the witness for t.

Actually, the proposed strategy is fair, in fact, it corresponds to a depth-first search in the proof-space with a limit on duplications which prevents the possibility to construct branches of infinite depth. For our purposes, we fix the duplication limit D to 0, that is no duplication are allowed.

If we consider the sequent $\forall\, x.\, p(x, f(x)) \vdash \forall\, x.\, \exists\, y.\, p(x, y)$, it is obviously true in **IL**, and so it must be provable in **IL**-T and **IL**-TDummy.

The TabTac prover is able to prove that sequent without duplications; the search for a tableau which proves that sequent goes as follows:

1. initial tableau

$$\mathbf{F}\forall\, x.\, \exists\, y.\, p(x, y), \mathbf{T}\forall\, x.\, p(x, f(x)) \quad .$$

2. expand the first swff

$$\frac{\mathbf{F}\forall\, x.\, \exists\, y.\, p(x, y), \mathbf{T}\forall\, x.\, p(x, f(x))}{\mathbf{F}\exists\, y.\, p(a, y), \mathbf{T}\forall\, x.\, p(x, f(x))}$$

   where a is e new variable.

3. again, expand the first swff in the only node of the terminal configuration

$$\frac{\dfrac{\mathbf{F}\forall\, x.\, \exists\, y.\, p(x, y), \mathbf{T}\forall\, x.\, p(x, f(x))}{\mathbf{F}\exists\, y.\, p(a, y), \mathbf{T}\forall\, x.\, p(x, f(x))}}{\mathbf{F}p(a, a), \mathbf{T}\forall\, x.\, p(x, f(x))}$$

the witness for y is generated according to the rule described above: the first term occurring in the node.

4. Since $\mathbf{F}p(a,a)$ is atomic, the other swff in the first node of the terminal configuration is expanded

$$\dfrac{\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x,f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}p(a,a), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}p(a,a), \mathbf{T}p(a,f(a))}$$

the witness for x is chosen to be a, since this is the first term occurring in the node.

5. The tableau is not closed and it cannot be expanded anymore, thus the prover backtracks; the last choice has been the witness for x in step 4; the first alternative is to use a new variable b as a witness:

$$\dfrac{\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x,f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}p(a,a), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}p(a,a), \mathbf{T}p(b,f(b))}$$

6. Again, the tableau is not closed and it cannot be expanded anymore, thus the prover backtracks; the last choice has been the witness for x in step 4 and in step 5; there are no alternatives on this choice, thus another backtrack instance occurs. The last choice has been the expansion of $\mathbf{F}\exists\, y.\, p(a,y)$ in step 3, and the alternative is to expand the other swff in that node:

$$\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x,f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}p(a,f(a))}$$

7. The first swff in the only node of the terminal configuration can be expanded:

$$\dfrac{\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x,f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}p(a,f(a))}}{\mathbf{F}p(a,a), \mathbf{T}p(a,f(a))}$$

where the first term occurring in the node has been chosen as a witness for y.

8. The tableau is not closed and it cannot be expanded anymore, thus the prover backtracks; the last choice has been the witness for y in step 7; the first available alternative is to use $f(a)$ as a witness:

$$\dfrac{\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x,f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x,f(x))}}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}p(a,f(a))}}{\mathbf{F}p(a,f(a)), \mathbf{T}p(a,f(a))}$$

9. The tableau is closed, so the prover exits with success.

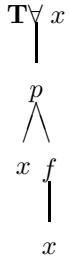
In order to measure the cost of developing the previous proof-table, we consider a series of parameters which identify both the time and the space required:

- number of backtracking steps;
- number of generated configurations;
- number of formulas which are constructed in the process;
- number of memory cells required to represents the formulas.

The number of backtracking steps, and the number of generated configurations are measures of the time required to search for a proof: in particular, the number of backtracking steps provides a rough idea on how time is wasted because of wrong choices, while the number of configurations measures the quantity of inference steps performed by the prover.

If we assume that the prover does not duplicate a configuration when it performs an inference step, the number of generated formulas is a simple measure for the memory space required to run the prover. A more refined measure is the number of cells, that is, the amount of memory elements required to represent a formula assuming a tree representation.

Hence, for example, $\mathbf{T}\forall\, x.\, p(x, f(x))$ requires 5 cells:



Although very rough, the number of cells takes into account the cost in space of representing complex formulas.

We have to remark that these space measures are not adequate to model the real space requirements of a prover because they do not consider the cost of *splitting rules* such as $\mathbf{T}\vee$, possibly requiring a duplication of the context of a tableau node, and because they do not consider the cost of maintaining a structure to support backtracking.

In any case, since the example we proposed does not use splitting rules and the backtracking structure does not radically change the amount of space required to compute, we feel that the proposed measures are a nice compromise, at least in the peculiar limits of our example.

In th proposed example, the performances of TabTac are summarized below:

- number of backtracking steps = 3;
- number of generated configurations = 8;
- number of swffs = 9;
- number of cells = 36.

Considering the behaviour of TabDTac on the same example, one gets:

1. initial tableau

$$\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x, f(x)) \quad .$$

2. expand the first swff

$$\frac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x, f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x, f(x))}$$

   where a is a new variable.
3. again, expand the first swff in the only node of the terminal configuration

$$\frac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x, f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x, f(x))}}{\mathbf{F}p(a,\alpha), \mathbf{T}\forall\, x.\, p(x, f(x))}$$

   where α is a new dummy variable.
4. The only expandable swff in the terminal configuration is the last one:

$$\frac{\dfrac{\dfrac{\mathbf{F}\forall\, x.\,\exists\, y.\, p(x,y), \mathbf{T}\forall\, x.\, p(x, f(x))}{\mathbf{F}\exists\, y.\, p(a,y), \mathbf{T}\forall\, x.\, p(x, f(x))}}{\mathbf{F}p(a,\alpha), \mathbf{T}\forall\, x.\, p(x, f(x))}}{\mathbf{F}p(a,\alpha), \mathbf{T}p(\beta, f(\beta))}$$

   where β is a new dummy variable.
5. In the check for closure we get that the tableau is closed by the substitution which assigns $\beta = a$, $\alpha = f(a)$.

Adopting the same measures as for TabTac, we get:

- number of backtracking steps = 0;
- number of generated configurations = 4;
- number of swffs = 5;
- number of cells = 21.

If we consider the cost of applying the closing substitution to the whole tableau, we get an extra amount of space which can be roughly measured by 2 new swffs for a total of 8 new cells.

Hence, it is evident that the TabDTac prover seems to be 2 times faster than TabTac and it requires 80% of the space used by the TabTac prover. Of course, these numbers do not take into account the time costs of running the unification algorithm to check the closure of a node, nor they take into account the cost in time of applying the closing substitution to the whole tableau. The last cost is required since, the translation procedure needs as input a tableau in the **IL** language, where no dummy variables are present.

A precise measure of the times for these operations is hard to obtain, but an empirical measure shows that TabDTac is 20% faster than TabTac and so, since the work needed by the translation algorithm is the same, being the final proof-tables identical, the TabDTac prover is 20% faster than the TabTac prover on the given example.

The example clearly shows that the heuristics which inspired TabDTac, i.e. *delay the choice of witnesses as much as possible*, pays in terms of performances, but when this fact happens and when the trade-off between reducing the amount of search versus running an unification algorithm is advantageous are still open questions.

6 Conclusions

The proposed approach, validation by translation, can be summarized in an abstract description. Calling $\mathcal{L}$ a calculus, implemented by a logical framework, and calling $\mathcal{P}$ another calculus, implemented by a prover in that environment, the technique described in this work validates the answer of the prover by means of an uniform map from proofs of $\mathcal{P}$ into proofs of $\mathcal{L}$.

Independently from the correctness of the prover, and from the correctness of the translation procedure, and even from the soundness of the $\mathcal{P}$ calculus, the result of the previous operation can be checked in $\mathcal{L}$. If the test is positive, the goal G is proven in $\mathcal{L}$, and, thus, the prover succeeds, otherwise the prover fails and the goal G is left open.

In any case, the final answer of the prover is correct, even if the prover may contain faults in the implementation or in the design. Therefore, as explicited and analyzed at length in the paper, the answer of the prover gets validated independently from its implementation.

Whenever our approach is applied, a formal development similar to the one presented in Section 4 has to be derived; as already remarked, the same formal setting is necessary to apply the embedding technique. On the other hand, whenever a logic $\mathcal{P}$ is embedded in the framework logic $\mathcal{L}$, a map τ between proofs has to be established, and this function τ constitutes the formal description of the translation procedure of our approach.

Hence, the validation by translation technique is as expressive as the embedding technique.

The convenience of one technique upon the other has to be judged on efficiency, and, in Section 4 it has been proven beyond any doubt that the validation by translation approach may be superior. It is easy to find analogous cases where the embedding technique is much more efficient, e.g., considering **ILD** as the framework logic, and **IL**-T as the prover logic. Normally, in our experience [7, 6],it is easy to verify what approach is more convenient when the formal setting has been developed, that is, when the map from the internal logic to the environment has been defined and proven correct.

The same kind of discussion holds for non-sound calculi, but another, more subtle phenomenon appears; we proved in Section 5 that a non-sound variant of a calculus may produce valid proofs in a small fraction of the time required by a sound prover. The definition of a proper class of formulas where this phenomenon takes place poses

serious difficulties.

In this respect, the results of Section 5 justify a research effort on the identification of classes of theorems preserved by non-sound variants of a fixed calculus.

As a side effect of the previously remarked phenomenon, it is possible to use the validation by translation approach to try to increase performances of a logical framework by incorporating non-sound provers.

Concluding, we proposed an auto-validating technique to develop theorem provers in a logical framework, and we proved that the technique is of interest by comparing it with the widely used embedding approach.

There are some open questions, and, precisely, when the validation by translation approach is convenient with respect to the embedding technique, and when a non-sound calculus let a prover to derive valid theorems in less time than the corresponding sound calculus.

Both questions have a common character: they require an *a priori* answers; in fact, reasoning *a posteriori*, it is already implicit in the present work the way to answer to both questions, at least in the limits of the case study we proposed.

References

[1] A. Avellone, M. Benini, and U. Moscato. Tactics for translation of tableau in natural deduction. In *Automated Reasoning with Analytic Tableaux and Related Methods: Position Papers*, 1999.

[2] A. Avellone, M. Benini, and U. Moscato. Translating tableaux into natural deduction: Applications to theorem proving. Technical Report 240-99, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, 1999.

[3] A. Avellone, M. Ferrari, and P. Miglioli. Duplication-free tableau calculi together with cut-free and contraction-free sequent calculi for the interpolable propositional intermediate logics. *Journal of Logic and Computation*, 1997.

[4] S. Agerholm. Mechanizing program verification in HOL. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *Proceedings of the ACM/IEEE 1991 International Workshop on Higher-Order Logic Theorem Proving System and its Applications*, pages 208–222, Davis, CA, 1992. IEEE.

[5] A. Avellone, P. Miglioli, U. Moscato, and M. Ornaghi. Generalized tableau systems for intermediate propositional logics. In D. Galmiche, editor, *Proceedings of the 6th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 43–61. Springer Verlag, 1997.

[6] M. Benini. *Verification and Analysis of Programs in a Constructive Environment*. PhD thesis, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, January 2000.

[7] M. Benini, D. Nowotka, and C. Pulley. Computer arithmetic: Logic, calculation and rewriting. In D.M. Gabbay and M. De Rijke, editors, *Frontiers of Combining Systems 2*, Series in Logic and Computation, pages 77–93. Research Studies Press, 1998.

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

[9] M.J.C. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. Milne and P. Subrahmanyam, editors, *Formal Aspects of VLSI Design*. North Holland, 1986.

[10] M.J.C. Gordon. Mechanizing programming logics in higher order logic. In G. Birtwistle and P. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 387–439. Springer Verlag, 1989.

[11] J. Harrison. Binary Decision Diagrams as a HOL derived rule. *The Computer Journal*, 38:162–170, 1995.

[12] J. Harrison. Stalmarck's algorithm as a HOL derived rule. In *Proceedings of 9th International Conference on Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 221–234. Springer Verlag, 1996.

[13] G. Huet and G. Plotkin, editors. *Logical Frameworks*. Cambridge University Press, 1991.

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

[15] P. Miglioli, U. Moscato, and M. Ornaghi. How to avoid duplications in refutation systems for intuitionistic logic and Kuroda logic. In K. Broda, M. D'Agostino, R. Goré, R. Johnson, and S. Reeves, editors, *Theorem Proving with Analytic Tableaux and Related Methods: 3rd International Workshop, Abingdon, U.K.*, pages 169–187, May 1994.

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

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

[18] L.C. Paulson. *Logic and Computation: Interactive Proof with Cambridge LCF*. Cambridge University Press, 1987.

[19] L.C. Paulson. Designing a theorem prover. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 415–475. Oxford University Press, 1992.

[20] L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[21] L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996.

[22] L.C. Paulson. Generic automatic proof tools. In R. Veroff, editor, *Automated Reasoning and its Applications*, chapter 3. The MIT Press, 1997. Also, Report No 396, Computer Laboratory, Cambridge University.

[23] L.C. Paulson. Isabelle's object-logics. Technical report, Computer Laboratory, Cambridge University, May 1997.

[24] L.C. Paulson. Isabelle's reference manual. Technical Report 283, Computer Laboratory, Cambridge University, May 1997.

[25] F.J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986.

[26] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[27] M. Redaelli. Sistemi di refutazione per logiche intermedie e modali. Master's thesis, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, 1994.

[28] H. Schwichtenberg and A.S. Troelstra. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.

[29] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer Verlag, 1973.

Received 27.4.1999, Reveised 20.12.2000