TableauxRocq: A Deep Embedding of Free-Variable Tableaux in Rocq

Johann Rosain
ENS de Lyon, Lyon, France

Julie Cailler
University of Lorraine, CNRS, Inria, LORIA, Nancy, France

Abstract

The free-variable tableau method has been widely used in order to automate proofs in multiple kinds of logics. Many automated theorem provers rely on this approach, either because it is the only available method—*e.g.,* in certain modal logics—or because it facilitates the generation of proof certificates. However, as far as the authors know, its results have never been formalized in a proof assistant. In this paper, we present TableauxRocq, a deep embedding of free-variable first-order tableaux in the Rocq prover. The formalized calculus is proved sound and provides a modular Skolemization system that enables the use of Skolemization-based optimizations. Moreover, we show how TableauxRocq can be used as a certifier for automated theorem provers by adapting the Goeland prover—that can already output Rocq terms—to output proofs in the TableauxRocq format. By using the power of reflection, thereby providing a fully certified proof checker for free, we show that Goeland's exported Rocq terms and TableauxRocq's proof certificates can be checked in a similar time frame without proof optimizations, and that the latter has strictly better performances in presence of Skolemization-related optimizations.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Computing methodologies → Theorem proving algorithms

Keywords and phrases The Rocq Prover, First-Order Tableaux, Automated Reasoning, Interoperability, Proof Translation

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.13

Supplementary Material The TableauxRocq version presented in this paper together with the modified version of Goeland and the scripts to run the experiments are available on Zenodo [52].
Software (Zenodo archive): `https://zenodo.org/records/20205704`
Software (Source Code): `https://github.com/jrosain/TableauxRocq/`
Software (Source Code): `https://github.com/GoelandProver/Goeland`
archived at `swh:1:dir:e18a81ed53b5e6275220097993f326d2504b254f`

Acknowledgements We would like to thank the anonymous reviewers for their valuable and insightful comments on the paper.

1 Introduction

The free-variable tableau method [30] is a proof-search procedure originally designed for first-order logic (FOL). This technique, in contrast to clausification-based approaches [2, 13, 5, 50, 47], allows the search for proofs to operate directly from the base formula, thus providing a more trusted proof output. Indeed, by avoiding complex transformations, these techniques provide more human-readable derivations, producing proofs that are in turn easier to (machine-)check. Free-variable tableaux have then been extended to first-order logic with theory reasoning [10, 28] such as arithmetic [53, 20] or rank-1 polymorphism [19], and to other logics, *e.g.,* modal logics [7], intuitionistic logics [45], and higher-order logic [43]. Due to their design as proof-search procedures, the various tableau calculi include many optimizations, making their soundness and completeness proofs rely on non-trivial arguments.

In practice, these optimizations lead to complex implementations in Automated Theorem Provers (ATPs) [17, 21, 53, 11, 8, 48], which in turn makes it difficult to fully trust their answers. Consequently, recent efforts in the ATPs community have focused on certifying the output of their tools [17, 24, 51, 44, 46, 49], relying mostly on *syntactic* methods, *i.e.,* by translating their proofs into a given format, with potential transformations and elaboration phases.

In this paper, we advocate the use of *semantic* methods based on a deep embedding in proof assistants. Conversely to syntactic methods, semantic ones propose to transfer the development effort from the ATPs' translation mechanism to the certification part by formalizing the underlying theory, thus allowing to readily use proofs generated by ATPs as-is. Semantic methods offer several advantages over syntactic ones. For instance, they make it possible to formally establish the soundness and completeness of *optimized* versions of the calculi, which in turn enable ATPs to output proofs with no further syntactic manipulations. This can result in a significant speedup in both proof generation and proof checking time, as some optimizations—such as Skolemization in first-order logic—are known to cause an explosion in the size of the proof certificates [4]. Moreover, semantic methods shift confidence in the certificate from a *trusted code base* to a *trusted theory base* paradigm. In syntactic approaches, beyond trusting the proof assistant's kernel, one must also trust the correctness of the proof-translation pipeline—a potentially complex mechanism that manipulates proof terms and may introduce errors. In semantic approaches, this additional trust is instead placed in the formalized theory itself: a set of mathematical definitions and lemmas that can be independently inspected and validated by a domain expert, without requiring any understanding of the implementation.

Semantic approaches also allow for the formal development of soundness and completeness proofs for actual proof-search procedures, which is a topic that, while being well studied for propositional logic [32, 14, 31, 41], is more scattered for FOL [54] and as far as the authors know, is missing for tableaux.

Contributions. Our contributions are threefold. First, we develop TableauxRocq, a deep embedding of the first-order free-variable tableau calculus in Rocq, presented in Sec. 2, that is modular w.r.t. the Skolemization method used. We provide a generic specification of a Skolemization step that allows us to prove the soundness of the tableau calculus w.r.t. the usual classical first-order semantics. Second, in Sec. 3, we implement an algorithm that decides whether a proof is a valid tableau proof and prove it sound w.r.t. the specification. Finally, we adapt Goeland [21], a first-order tableau-based theorem prover supporting multiple Skolemization strategies to output proofs in the TableauxRocq format. We also report on initial benchmarks validating our approach in Sec. 4.

Throughout the paper, we provide links to an HTML-rendered version of the Rocq formalization, signaled by a small Rocq logo  . The formalization is available on GitHub.[1] For ease of writing, everything in the code is in ASCII, but both in the HTML version and the paper, we render some notations in UTF-8 characters.

2 Formalized Free-Variable First-Order Tableau Calculus

In this section, we describe the core of the TableauxRocq library. It contains an implementation of the free-variable first-order tableau calculus using a minimal syntax, which makes it possible

[1] At `https://github.com/jrosain/TableauxRocq`.

Listing 1 Rocq Implementation of the Typeclass of Atoms

```
Record Atom :=
  { atom :> Type
  ; eqb_atom :: EqBool atom
  ; set_atom : set atom }
```

to avoid redundant cases in proofs while not limiting expressiveness. As is usually done in syntaxes with binders, we use a specific representation of variables, presented in Sec. 2.1. Based on this representation, we define a minimal syntax and semantics for FOL in Sec. 2.2. We then introduce an axiomatization of Skolemization in Sec. 2.3, which enables to formulate a modular tableau calculus in Sec. 2.4 and prove its soundness in Sec. 2.5.

2.1 A Locally Nameless Representation of Variables

Systems with binders (*e.g.,* $\forall$ and $\exists$ in FOL) are tricky to formalize. In pen-and-paper proofs, definitions such as α-conversion and substitution are often treated informally, relying on the reader's intuition to fill in the gaps. However, directly formalizing these definitions in a proof assistant incurs significant overhead. Hence, a common solution is to use De Bruijn indices [27], a representation in which binders do not explicitly name their variables. Instead, variables are replaced by natural numbers indicating the number of binders between the variable and its corresponding binder. This approach automatically handles α-conversion and allows for a straightforward definition of substitution. Consequently, in dependent-type based provers, most systems with binders are formalized using De Bruijn indices [1, 56, 33, 29, 61, 58].

In TableauxRocq, we adopt a locally nameless representation [25] for managing variables. This approach keeps the advantages of the De Bruijn representation—by implementing bound variables with natural numbers—while providing more flexibility in handling free variables. Because tableau proofs involve a list of formulas rather than a single one, maintaining the De Bruijn indices of free variables synchronized would require substantial effort, which the locally nameless representation allows us to avoid.

In our use case, the type of free variables cannot be treated as a bare type; it needs additional structure. In TableauxRocq, this structure is given by a type that we call `Atom`: a type that has (i) a way to decide equality between its elements and (ii) the ability to collect them in a set-like structure, for example when computing the set of variables appearing in a formula. The implementation of `Atom` is given in Listing 1. Note that, since our aim is to write executable algorithms, an `Atom` satisfies the `EqBool` class—a boolean equality `eqb` that reflects the propositional one—which is equivalent to point (i). The `atom` field, *i.e.,* the *atomic* type, allows us to provide *structures* for *e.g.,* natural numbers or strings, and to write `Check` $\mathbb{N}$ `: Atom`. This makes it possible to pass $\mathbb{N}$ directly to a function expecting an `Atom`, without explicitly providing its `Atom` instance. The token `:>` indicates an implicit coercion from an `Atom` to its carrier type, allowing one to write `x : A` instead of `x : atom A` for `A : Atom`. Similarly, the token `::` automatically declares an instance of the typeclass `EqBool`, enabling one to write `eqb x y` instead of `eqb (eqb_atom A) x y`.

We also provide type classes and definitions for elementary locally nameless operations, allowing for the same notations to be shared across different structures implementing locally nameless variables. In the following, `t` and `u` denote terms and `F` a formula.

- variable opening, denoted `t{n → u}`, which replaces the n^{th} bound variable in `t` with `u`,

$$
\begin{array}{rrcll}
\text{(terms)} & t & ::= & n \mid x \mid f(t_1, \ldots, t_n) & n \in \mathbb{N}, x \in \mathcal{V}, f \in \mathcal{F} \\
\text{(formulas)} & F, G & ::= & \bot \mid P(t_1, \ldots, t_n) \mid \neg F \mid F \vee G \mid \forall F & P \in \mathcal{P}
\end{array}
$$

Figure 1 Minimal Locally Nameless First-Order Syntax

$$
\begin{aligned}
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models n]\!] &\triangleq \rho[n] \\
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{x}]\!] &\triangleq \sigma(\texttt{x}) \\
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{f}(t_1, \ldots, t_n)]\!] &\triangleq \texttt{M(f)}([\![\texttt{M \# } \rho \texttt{ \# } \sigma \models t_1]\!], \ldots, [\![\texttt{M \# } \rho \texttt{ \# } \sigma \models t_n]\!])
\end{aligned}
$$

(a) Interpretation of Terms

$$
\begin{aligned}
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{P}(t_1, \ldots, t_n)]\!] &\triangleq \texttt{M(P)}([\![\texttt{M \# } \rho \texttt{ \# } \sigma \models t_1]\!], \ldots, [\![\texttt{M \# } \rho \texttt{ \# } \sigma \models t_n]\!]) \\
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{Neg } F]\!] &\triangleq \neg [\![\texttt{M \# } \rho \texttt{ \# } \sigma \models F]\!] \\
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{Or } F_1 \ F_2]\!] &\triangleq [\![\texttt{M \# } \rho \texttt{ \# } \sigma \models F_1]\!] \vee [\![\texttt{M \# } \rho \texttt{ \# } \sigma \models F_2]\!] \\
[\![\texttt{M \# } \rho \texttt{ \# } \sigma \models \texttt{All } F]\!] &\triangleq \forall(\texttt{x} : \texttt{M}),\ [\![\texttt{M \# x :: } \rho \texttt{ \# } \sigma \models F]\!]
\end{aligned}
$$

(b) Interpretation of Formulas

Figure 2 Minimal Locally Nameless First-Order Semantics

- bound variables computation (), denoted `bv F` or `bv t` giving the bound variables of `F` or `t`,
- free variables computation (), denoted `fv`, the counterpart of `bv` for free variables,
- local closure (), denoted `isLocallyClosed`, which is true if there are no *bound* variables in a term or in a formula,
- closure (), denoted `isClosed`, which is true if there are no *free* variables in a term or in a formula, and
- substitution (), denoted `x@[σ]`, which maps atoms to locally closed terms.

2.2 A Minimal Syntax and Semantics

TableauxRocq implements the FOL locally nameless syntax of Fig. 1. This syntax is minimal: it includes terms (integers, variables, and functions) and formulas (bottom $\bot$, predicates, negation $\neg$, disjunction $\vee$, and universal quantification $\forall$). In this formalism, the formula $\forall x\,(\forall y\, R(a, Z, x, y))$ is represented in locally nameless style as $\forall(\forall(R(a, Z, 1, 0)))$, where a is a constant, Z is a free variable, and the bound variables are natural numbers indicating how many $\forall$s must be discarded to reach the binder they refer to. Moreover, the sets $\mathcal{V}$, $\mathcal{F}$, and $\mathcal{P}$—which respectively represent free variables, function symbols, and predicate symbols—are implemented as `Atom`s that are (implicit) parameters of the syntax, providing greater flexibility to the user. This design also allows us to show, for example, that terms and formulas have a boolean equality—*i.e.,* we have the typeclass instances `EqBool Term` and `EqBool Form`. Substitution and variable opening can then be straightforwardly defined, making it possible to prove the expected commutation lemmas between these operations.

$$\frac{\neg(\forall F)}{\neg F\{0 \to \texttt{sko}(X_1, \ldots, X_n)\}}\delta$$

Figure 3 General Shape of a Skolemization Rule

Semantics. The semantics implemented in TableauxRocq follows the ideas of the syntax: standard first-order semantics generalized over the types $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{V}$, as shown in Fig. 2. Because the syntax involves both bound and free variables, the interpretation of terms and formulas differs slightly from the usual interpretation functions: we maintain both a bound- and a free-variable environment. The former, which we mainly denote ρ, is represented as a list of elements of the domain, while the latter, that is often denoted by σ or μ, is implemented as a function. To allow for consistent notation across interpretations, TableauxRocq implements the interpretation function as a typeclass and provides instances for terms and formulas.

We denote by $[\![\texttt{M \# } \rho \texttt{ \# } \sigma \vdash \texttt{N}]\!]$ the interpretation of `N` (which can be *e.g.,* a term, a formula, a list of formulas, ...) in the model `M`, with the bound-variable environment ρ and the free-variable environment σ. The usual lemmas about commutation between (i) variable opening and the bound-variable environment (,), and (ii) substitution and the free-variable environment (,) follow naturally.

A formula `F` is *satisfiable* () if there exists a model `M` such that for every free-variable environment σ, $[\![\texttt{M \# } \cdot \texttt{ \# } \sigma \models \texttt{F}]\!]$, and *valid* if for every model `M`, $[\![\texttt{M \# } \cdot \texttt{ \# } \cdot \models \texttt{F}]\!]$. In particular, we denote $\Gamma \models \texttt{F}$ if `Or (Neg Γ) F` is valid (where a context is coerced into a formula by taking the conjunction of its elements).

2.3 Axiomatizing Skolemization

In FOL free-variable tableaux, the Skolemization rule applied to an existentially quantified formula, whose general shape is presented in Fig. 3, has many variations [30, 40, 9, 3, 22, 37]. This rule instantiates the first bound variable of F with the Skolem function returned by the `sko` meta-function, which gives a Skolem symbol and selects a subset of the free variables occurring in the branch.

For instance, in the earliest version of the rule given by Fitting [30], called *outer Skolemization*, `sko` returns a function symbol that is fresh (*i.e.,* not already appearing in the tableau) and parameterized by all free variables of the branch. Later, Hähnle and Schmitt [40] observed that parameterizing only by the free variables occurring in the Skolemized formula suffices to retain soundness. This method is known as *inner Skolemization*. Moreover, they showed that there exists a class of formulas $(\Phi_n)_{n\in\mathbb{N}}$ such that, if $b(n)$ (resp. $b^+(n)$) denotes the number of branches in a proof of Φ_n using outer (resp. inner) Skolemization, then $b^n = O(2^{b^+(n)})$.

To account for the different (un)known Skolemization strategies, we formalize Skolemization as an operation specified by the following data ():

- a Skolemization record (`sko_record`), explained in the last paragraph of the subsection,
- a boolean-valued predicate `is_sko : Term → Form → sko_record → set_atom` $\mathcal{V}$ $\to$ `set_atom` $\mathcal{F}$ $\to \mathbb{B}$, that checks whether a term is a valid Skolemized symbol of a formula given the free variables of the branch and the function symbols of the whole tableau,
- a `symbol` function that returns the function symbol of any valid Skolemized term t, and
- an `args` function that returns the arguments of any valid Skolemized term t.

This data must satisfy four requirements. Since the fourth one will only become relevant later in proofs, we first provide some intuition and defer the detailed explanation to Sec. 2.5.

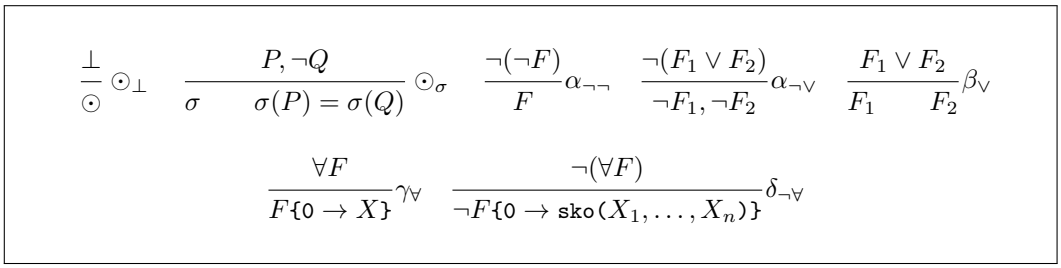

Figure 4 Minimal First-Order Free-Variable Tableau Rules

▶ **Requirement 1** ( ). For every term `t`, if `t` is a valid Skolemized term, then `t = Fun (symbol t) (args t)`.

▶ **Requirement 2** ( ). For every term `t`, if `t` is a valid Skolemized term, then the arguments of `t` must all be free variables.

▶ **Requirement 3** ( ). Every term `t` that is a valid Skolemized term under a context with a set of function symbols `s` is also a valid Skolemized term under a context with any subset of `s` as function symbols.

▶ **Requirement 4** ( ). For every term `t` and model `M`, if `t` is a valid Skolemized term w.r.t. a formula `Neg (All F)`, then there exists a model `M'` such that (i) for every formula `G`, if `G` is satisfied in `M`, then it is also satisfied in `M'`, and (ii) if `Neg (All F)` is satisfied by `M`, then `Neg F{0 → t}` is satisfied by `M'`.

The first three requirements ensures that the predicate `is_sko` is well-formed. The last one is the key ingredient for proving the soundness of the tableau method.

Skolemization Record ( ). Skolemization records appear starting from the *pre-inner* Skolemization variant [9]. This version of the rule makes the generated Skolem symbol formula-dependent, *i.e.,* if F is a formula to which a δ-rule is applied twice, the same Skolem symbol is produced both times. It is therefore necessary to keep track of the formulas with which Skolem symbols are associated. For example, for the outer and inner Skolemization rules, it suffices to instantiate the Skolemization record as a set, since in these cases only freshness needs to be ensured.

2.4 Tableaux Proofs

The first-order tableau method is a forward-reasoning technique presented as an upside-down proof tree with root at top, as illustrated in Fig. 4. It is composed of unary (α, γ and δ) and binary (β) expansion rules, extending a branch with at most two formulas in the unary case and dividing a branch into two distinct ones in the binary case. $\odot$-rules are closure rules and can be applied on a branch whenever there is either a trivial contradiction, or if there are two formulas P and $\neg Q$ and a substitution σ making $\sigma(P)$ and $\sigma(Q)$ definitionally (*i.e.,* syntactically) equal. A tableau is said *closed* if all its leaves are instances of closure rules. In that case, since a tableau proceeds by *refutation*, the negation of the root formula is universally valid.

Our formalization mimics this forward-reasoning technique in three stages. First, we define a `TableauTree` ( ) as a binary tree whose nodes are labeled with lists of formulas. Intuitively, each label corresponds to the formulas resulting from the application of a tableau

$$\frac{\texttt{is\_branch\_of B T} \quad \texttt{is\_on\_branch (Neg (Neg F)) B T}}{\texttt{T} \triangleright \texttt{expand\_tableau\_branch (Some [F]) None B T}}\ \textsc{AlphaNegNeg}$$

$$\frac{\texttt{is\_branch\_of B T} \quad \texttt{is\_on\_branch (Neg (Or F G)) B T}}{\texttt{T} \triangleright \texttt{expand\_tableau\_branch (Some [Neg F ; Neg G]) None B T}}\ \textsc{AlphaNegOr}$$

$$\frac{\texttt{is\_branch\_of B T} \quad \texttt{is\_on\_branch (Or F G) B T}}{\texttt{T} \triangleright \texttt{expand\_tableau\_branch (Some [F]) (Some [G]) B T}}\ \textsc{BetaOr}$$

$$\frac{\texttt{is\_branch\_of B T} \quad \texttt{is\_on\_branch (All F) B T} \quad \texttt{x} : \mathcal{V}}{\texttt{T} \triangleright \texttt{expand\_tableau\_branch (Some [F\{0} \to \texttt{Free x\}]) None B T}}\ \textsc{GammaAll}$$

$$\frac{\begin{array}{c}\texttt{is\_sko t (Neg (All F)) (symbols T) (fv (get\_context B T))} \\ \texttt{(function\_symbols (get\_all\_formulas T)) = true} \\ \texttt{is\_branch\_of B T} \quad \texttt{is\_on\_branch (Neg (All F)) B T} \\ \texttt{symbs = add\_symbol (symbol t) (Neg (All F)) (symbols T)}\end{array}}{\texttt{T} \triangleright \texttt{expand\_tableau\_branch (Some [F\{0} \to \texttt{t\}]) None B (T, symbs)}}\ \textsc{DeltaNegAll}$$

Figure 5 Expansion Rules

rule. A *branch* () is a list of *branching steps* `Left` or `Right`. We say that `B` is a branch of a tableau tree `T` () if `B` is a path from the root of `T` to a node with no children. Moreover, a formula is on a branch of `T` () if it appears in one of the labels along that branch. We also define the following functions:

- `get_context` (), which accumulates the list of formulas along a branch,
- `get_all_formulas` (), which is the union of `get_context` for all the branches of a tableau,
- `expand_tableau_branch` (), which given a branch `B`, a tableau `T` such that `is_branch_of B T` holds, and two optional list of formulas corresponding to the potential labels of the children of the last node of `B`, adds the non-`None` list of formulas as children of the last node of `B`, and
- `replace_child` (), which replaces the last `TableauTree` of a given path in a tableau tree.

A `Tableau` () is a pair consisting of a `TableauTree` together with a `sko_record`. We define an implicit coercion from a `Tableau` to its underlying `TableauTree`.

The second step of our definition consists in introducing a binary relation between tableaux that effectively implements the tableau expansion rules—*i.e.,* all rules except the closure rules. This relation is denoted `T ▷ T'` and described in Fig. 5. The most notable change w.r.t. the tableau rules of Fig. 4 is Rule DeltaNegAll that expects a term `t` and uses the given Skolemization boolean predicate to check whether `t` is a valid Skolem function in the context. In this case, the Skolem symbol of `t` is added to the `sko_record` in order to be used in subsequent Skolemization steps of the tableau. The other rules are implemented straightforwardly from the corresponding tableau rules. By convention, in unary rules, we extend the left child of a node.

The third step of our definition mirrors the closure rules. We say that a tableau `T` is *closed under the substitution* σ () iff, for every branch `B` of `T`, either `Bot` belongs to the context of `B`, or there are two formulas `F` and `G` in the context of `B` s.t. `F@[`σ`] = Neg G@[`σ`]`.

Using these three elements, we say that the context Γ has a tableau under the substitution σ () iff there exists a `Sequence` (*i.e.,* a list of `Tableaux`) `s` such that (i) the first element of `s` is the tableau with a single node labeled by Γ, (ii) `s[i]` ▷ `s[i+1]` for every `i`, and (iii) the last element of `s` is a closed tableau under the substitution σ.

Note that, if a sequence `s` satisfies `s[i]` ▷ `s[i+1]` for every `i`, we call `s` an *expansion sequence* (). A tableau `T` is said *satisfiable* () if there exists a model `M` such that for every free-variable environment μ, there exists a branch `B` of `T` such that ⟦ `M # [] #` μ ⊨ `get_context B T` ⟧.

2.5 Soundness of the Calculus

In order to use the `hasTableau` predicate for checking proofs or in other developments, a key property we need to show is soundness of the system:

> For every closed context Γ and formula `F`, if there exists σ such that `hasTableau sko (Neg F :: Γ)` σ, then Γ ⊨ `F`.

where `Neg F ::` Γ denotes adding `Neg F` to the list of formulas Γ.

Intuitively, this follows from the fact that, for every tableau expansion rule of Fig. 4, the tableaux *before* and *after* the expansion are equisatisfiable, provided that the Skolemization is valid—*i.e.,* if the interpretation of `sko`$(X_1, \ldots, X_n)$ can be assigned to an arbitrarily chosen value in the model. Since the last tableau of the sequence is *closed*, every branch contains a contradiction and is therefore unsatisfiable.

In fact, full equisatisfiability is not required to establish soundness [30]. It suffices to show that (i) a closed tableau is unsatisfiable and (ii) the tableau expansion rules preserve satisfiability. The first point is straightforward: the substitution σ can be seen as a free-variable environment under which no branch can be satisfied, regardless of the model.

▶ **Lemma 5** (Closed Tableau is not Satisfiable ). For every model `M`, context Γ, substitution σ and sequence `s`, if `s` is an expansion sequence such that the first element of `s` is the tableau with a single node labelled by Γ and the last tableau of `s` is closed, then no `B` branch of the last tableau of `s` is satisfied by `M` under the substitution σ coerced as a free-variable environment.

To demonstrate the second point, we prove that the expansion rules of Fig. 5 are satisfiability preserving.

▶ **Lemma 6** (Expansion Preserves Satisfiability ). For every tableaux `T` and `T′`, if `T` is satisfiable and `T ▷ T'`, then `T'` is satisfiable.

Proof. By case analysis over the expansion rule used. Cases AlphaNegNeg, AlphaNegOr, GammaAll are straightforward applications of the assumption that `T` is satisfiable.

Case BetaOr. The model `M` given by the satisfiability of `T` is the one that satisfies the extended tableau. Without loss of generality, given a free-variable environment μ, we assume that the last expanded branch `B` is the satisfiable one. Then, as ⟦`M # · #` μ ⊨ `Or F G`⟧, either ⟦`M # · #` μ ⊨ `F`⟧, which makes the left child of `B` satisfiable, or ⟦`M # · #` μ ⊨ `G`⟧, which makes the right child of `B` satisfiable.

Case DeltaNegAll. Assuming that there is a model `M` that satisfies `T`, we must construct a model that satisfies `T` extended with `Neg F{0 → t}` on the branch `B`. We define this model to be `M` inside which we replace the interpretation function () by the one given by the Requirement 4 of Sec. 2.3.

▶ **Requirement** (Requirement 4)**.** For every term `t`, formula `F`, function symbols `s` and model `M`, if `t` is a valid Skolemized term, then there exists an interpretation function `f : ℱ → list M → M` such that:

- **(i)** for every free-variable environment μ, if the function symbols of `F` are a subset of `s` and ⟦ `M # · # μ ⊨ Neg (All F)` ⟧ then ⟦ `ReplacementModel M f # · # μ ⊨ Neg F{0 → t}` ⟧, and
- **(ii)** for every formula `G` and free-variable environment μ, if the function symbols of `G` are a subset of `s` and if ⟦ `M # · # μ ⊨ G` ⟧, then ⟦ `ReplacementModel M f # · # μ ⊨ G` ⟧.

Then, given a free-variable environment in `ReplacementModel M f` (i.e., a free-variable environment in `M`), there are two further cases. Either the satisfiable branch is not the one that was expanded, and by Item (ii) it remains satisfiable in the new model; or it is the expanded branch, in which case it is satisfiable by the two properties of Requirement 4. ◀

Moreover, we can show that the Requirement 4 is fulfilled by the Skolemization strategies.

▶ **Lemma 7** ()**.** Outer Skolemization satisfies the four requirements of Skolemization (*c.f.*, Sec. 2.3).

Proof. We focus on the last requirement. First, we remark that given a free-variable environment μ, we can write a function that returns a mere element `c` of `M` such that if ⟦ `M # · # μ ⊨ Neg (All F)` ⟧, then ⟦ `M # [c] # μ ⊨ Neg F` ⟧ (). Indeed, by using the full power of classical logic, we can decide whether ⟦ `M # · # μ ⊨ Neg (All F)` ⟧ is true or false. In the former case, we take the element `c` yielded by the hypothesis. In the latter case, we choose an arbitrary member of `M`.

Therefore, using the choice axiom—which is unavoidable as Skolemization is a function that chooses non constructively a constant value that makes the instantiated formula equisatisfiable w.r.t. the quantified one—we can extract this `c` from the previous function. Then, given `f : ℱ` and `l : list M`, the interpretation function acts as follows:

- if `f` is the Skolem symbol, we update the free-variable assignment to be $[X_i \mapsto c_i]$, where X_i is the i-th argument of the Skolem function and c_i the i-th member of the list `l`, and return the corresponding `c` yielded by the choice function, and
- otherwise, return the result of the interpretation function of `M`.

As the Skolem symbol is fresh, it is clear that any formula satisfied in `M` is also satisfied in the new model. Moreover, `Neg F{0 → t}` is also satisfiable by commutation of variable opening and bound-variable environment and that the interpretation of `t` in the new model is `c`. ◀

Since the Skolem symbol is also fresh, this proof can be reused in the inner Skolemization () case. The justification for pre-inner Skolemization is more involved, relying ultimately on the fact that the Skolem symbol appears only in the formula from which it originates. As this formula is satisfied in the model, the presence of the Skolem symbol does not affect the satisfiability of the other formulas.

A simple corollary of Lemma 6 is that an entire expansion sequence is satisfiable provided that its first tableau is satisfiable.

▶ **Corollary 8** (Expansion Sequence Preserves Satisfiability )**.** For every expansion sequence `s` and tableau `T`, if `s` starts by `T` and `T` is satisfiable, then `s[i]` is satisfiable for every `i`.

▶ Remark 1. Note that, using Corollary 8, it is straightforward to construct a model that satisfies every tableau in the sequence: it suffices to show that inverting the tableau rules preserves satisfiability in the same model (which is easy). For instance, this is the approach chosen in the literature to prove the soundness of pre-inner Skolemization [9].

Finally, we can bring all the components together to show the soundness theorem.

▶ **Theorem 9** (Soundness). For every substitution σ, context Γ and formula `F`, if Γ and `F` are closed and `Neg F :: Γ` has a tableau under the substitution σ, then $\Gamma \models$ `F`.

Proof. Let `M` be a model. Assume that `M` satisfies `Neg F :: Γ` in the empty free-variable environment. As `Neg F :: Γ` is closed, `M` actually satisfies `Neg F :: Γ` in any free-variable environment. Let `T` be the last tableau of the sequence for `Neg F :: Γ`. By Corollary 8, `T` is satisfiable, *i.e.,* there exists a model `M'` s.t., for every free-variable environment, at least one branch of `T` is satisfied. By Lemma 5, we know that the substitution σ coerced to a free-variable environment does not satisfy `T`, which is a contradiction. ◀

3 A Fully Certified Proof Checker

In this section, we present the different parts of the proof-checker algorithm implemented in TableauxRocq. First, as our goal is to use this algorithm with real-world ATPs, we extend TableauxRocq to the full FOL syntax in Sec. 3.1. Next, we describe the design of the proof-checking algorithm in Sec. 3.2. Finally, we prove its soundness in Sec. 3.3.

3.1 Extended First-Order Logic

As first-order ATPs are not restricted to the fragment of FOL used in the previous sections, we added an extended syntax of terms ( ) and formulas ( ) to TableauxRocq. This extended syntax supports full first-order logic, both in terms of syntax and semantics (,), with string-valued variables, function symbols and predicate symbols.

We can then easily define translation functions between `ETerm`s and `Term`s (`translate_ETerm`), and between `EForm`s and `Form`s (`translate_EForm`), by storing bound variables in a list `m` and translating a standard variable to its index in `m` if found, or to a free variable otherwise. With this translation suitably defined, we obtain a correspondence between the fragment and the full semantics.

▶ **Lemma 10** ( ). For every model `M`, extended term `t`, list of strings `bvs`, bound-variable environment ρ and free-variable environment σ, $[\![$ `M # ρ # σ` $\models$ `translate_ETerm bvs t` $]\!]$ is equal to `interpret_eterm M (extended_env bvs ρ σ) t`.

In this lemma, the `extended_env` function maps a string `x` to `ρ[n]` if `x` has index `n` in ρ, or to `σ(x)` if it is not found.

▶ **Lemma 11** ( ). For every model `M`, extended formula `F`, list of strings `bvs`, bound-variable environment ρ and free-variable environment σ, $[\![$ `M # ρ # σ` $\models$ `translate_EForm__aux bvs F` $]\!]$ iff `interpret_eform M (extended_env bvs ρ σ) F`.

Thus, the translation of any formula is valid iff the corresponding base formula is valid in the extended semantics. This allows us to show that the soundness theorem for tableaux carries over to the extended syntax.

▶ **Theorem 12** (Soundness of the Extended Syntax). For every extended formula `F`, Skolemization `sko`, context Γ and substitution σ, if the translation of `Neg F :: Γ` is closed and has a tableau, then Γ implies `F` in the extended semantics.

3.2 Proof-Checking Algorithm

A proof-checking algorithm should be *informative*, *i.e.,* it does not only check whether a proof is valid or not, but also returns the exact error encountered on an invalid proof. Without this, it would be difficult for ATPs developers to fix the reported issues, which are often benign. Hence, to enable the printing of messages, we work in the `Result` monad defined as `Result A` $\triangleq$ `A` $\times$ `list string`, with:

- unit: `ret x := (x, [])`, and
- bind: `r >>= f := let r' := f (fst r) in (fst r', snd r ++ snd r')`,

for which we define a notation that is similar to Haskell's `do` block; *i.e.,*

```
b ← CheckProof sko Γ σ T;
ret (negb b).
```

is syntactic sugar for

```
CheckProof sko Γ σ T >>= fun b ⇒ ret (negb b).
```

In this snippet, the `CheckProof` function () is the implementation of the proof-checking algorithm defined subsequently in this section. It takes as inputs (i) a Skolemization technique, (ii) a context Γ (here, a list of formulas), (iii) a substitution σ, and (iv) a `RuleTree`, which implements the fragment set of tableaux rules (detailed in Listing 2), and returns a `Result` $\mathbb{B}$.

Listing 2 Rule Tree Definition 

```
Inductive Rule : Type :=
| AlphaNegNeg : Form → Rule | AlphaNegOr : Form → Rule
| BetaOr : Form → Form → Rule
| GammaAll : Form → string → Rule
| DeltaNegAll : Form → Term → Rule.

Inductive RuleTree : Type :=
| Leaf : Option (Form * Form) → RuleTree
| Node : RuleTree → Rule → RuleTree → RuleTree.
```

The actual algorithm () is also parameterized by a set of function symbols corresponding to the constant symbols of the initial context, as well as a Skolemization record of symbols introduced by the `DeltaNegAll` rules, kept in the result. It is defined by induction over the `RuleTree` argument as follows:

- On a `Leaf`, there are two cases:
  - If no pair of formulas is given, we search for a trivial contradiction in Γ. If found, we return `true` with no error messages. Otherwise, we return `false` and inform the user that Γ contains no trivial contradiction.
  - If a pair of formulas is given, we check that both are indeed in Γ and that one of the formula is the negation of the other after applying σ. If any of these conditions fails, we return a message indicating that there are no contradictions in Γ`@[`σ`]`.
- On a `Node`, there are three cases:
  - In the `AlphaNegNeg`, `AlphaNegOr`, and `GammaAll` cases, we check that the given formula is in Γ and call the algorithm recursively on Γ augmented with the formulas resulting from the rule application. If any of these steps fails, we return `false` together with the corresponding message.
  - In the `BetaOr` case, we first check that the target formula is in Γ. We then call the algorithm recursively on Γ augmented with the left sub-formula. If this call returns an

error, we propagate it. Otherwise, we recursively call the algorithm on Γ augmented with the right sub-formula, along with the set of symbols returned by the first call, and return its result.

- In the `DeltaNegAll` case, we check that the target formula is in Γ and that the given term is a valid Skolemized term w.r.t. the given Skolemization technique. We then recursively call the algorithm on Γ augmented with the resulting formula, adding the function symbol of the given term to the Skolemization record, and return its result.

The `CheckProof` function then calls `CheckProof__aux`, with an empty Skolemization record and the set of function symbols of the given list of formulas. Then, it discards the record returned by `CheckProof__aux` in the result.

3.3 Soundness of the Algorithm

Proving the algorithm sound means that, for every input proof validated by `CheckProof`, we must build a proof of `hasTableau` from it. Consequently, this proceeds in three steps: (i) show that a positive answer from a call of `CheckProof` implies that a sequence of tableaux can be extracted, (ii) show that this sequence is an expansion sequence, and (iii) show that the last tableau in this sequence is closed.

The first point can be addressed by extracting a sequence directly from a `RuleTree`. This is the role of the function given in Listing 3.

Listing 3 Translating a Rule Tree Into a Sequence of Tableaux

```
Fixpoint RuleTree_to_Sequence__aux (sko : Skolemization) (B : Branch) (T : Tableau)
    (R : RuleTree) : option Sequence := match R with
  | Leaf _ ⇒ ret [T]
  | Node R₁ r R₂ ⇒
    match r with
    | AlphaNegNeg F ⇒
      Γ ← get_neg_neg F;
      T' ← expand_tableau_branch sko (Some (Ctx.elements Γ)) None B T;
      s ← RuleTree_to_Sequence__aux (B ++ [Left]) T' R₁;
      ret (T :: s)
    | AlphaNegOr F ⇒
      Γ ← get_neg_or F;
      T' ← expand_tableau_branch sko (Some (Ctx.elements Γ)) None B T;
      s ← RuleTree_to_Sequence__aux (B ++ [Left]) T' R₁;
      ret (T :: s)
    | BetaOr F ⇒
      Γs ← get_or F;
      T' ← expand_tableau_branch sko (Some (Ctx.elements (fst Γs)))
             (Some (Ctx.elements (snd Γs))) B T;
      s₁ ← RuleTree_to_Sequence__aux (B ++ [Left]) T' R₁;
      s₂ ← RuleTree_to_Sequence__aux (B ++ [Right]) (last s₁ (mkLeaf sko)) R₂;
      ret (T :: removelast s₁ ++ s₂)
    | GammaAll F x ⇒
      G ← get_all F;
      T' ← expand_tableau_branch sko (Some [G{0 → Free x}]) None B T;
      s ← RuleTree_to_Sequence__aux (B ++ [Left]) T' R₁;
      ret (T :: s)
    | DeltaNegAll F t ⇒
      G ← get_neg_all F;
      f ← get_symbol t;
```

```
        T₀ ← expand_tableau_branch__aux (Some [G{0 → t}]) None B T;
        let T' := {| tree := T₀; symbols := add_symbol f F (symbols T) |} in
        s ← RuleTree_to_Sequence__aux (B ++ [Left]) T' R₁;
        ret (T :: s)
    end
end.
```

This function satisfies a few small properties. First, the extracted sequence is never empty; in fact, the first element of this sequence is always the tableau given as a parameter.

▶ **Lemma 13** (). For every rule tree `R`, branch `B`, tableau `T`, and sequence `s`, if `RuleTree_to_Sequence__aux B T R = Some s`, then the first element of `s` is `T`.

Moreover, the function only acts on the branch `B` and its extensions; that is, given a tableau `T`, the last element of the extracted sequence is the tableau `T` in which the last node of the branch `B` has been replaced by another non-empty tableau.

▶ **Lemma 14** (). For every rule tree `R`, branch `B`, tableau `T` and sequence `s`, if `B` is a branch of `T` and `RuleTree_to_Sequence__aux B T R = Some s`, then there exists a non-empty tableau tree `T'` such that `replace_child B T T' = Some (tree (last s (mkLeaf sko)))`.

In the previous statement, `last` is a function that returns the last element of the sequence `s` if it exists. Otherwise, it returns the tableau given in the second argument, which here is a leaf—*i.e.,* an empty tableau. Both of these properties ensure that `RuleTree_to_Sequence__aux` returns a sequence of tableaux whenever the `CheckProof__aux` function returns a positive answer.

▶ **Lemma 15** (). For every substitution σ, set of function symbols `fs`, rule tree `R`, branch `B`, tableau `T` and Skolemization records `r` and `r'`, if `CheckProof__aux sko fs (get_context B T) σ r R = ret {| status := true; symbs := r' |}` and `B` is a branch of `T`, then there exists a sequence of tableaux `s` such that `RuleTree_to_Sequence__aux B T R = Some s`.

We can further characterize this sequence by the fact that the Skolemization record of the last tableau of the sequence is exactly the record returned by the proof-checking algorithm.

▶ **Lemma 16** (). For every substitution σ, set of function symbols `fs`, rule tree `R`, branch `B`, tableau `T` and Skolemization record `r`, if `B` is a branch of `T`, `CheckProof__aux sko fs (get_context B T) σ (symbs T) R = ret {| status := true; symbs := r |}`, and `RuleTree_to_Sequence__aux B T R = Some s`, then `r = symbols (last s (mkLeaf sko))`.

Recall that the set of function symbols provided to `CheckProof__aux` is the constant set of function symbols from the initial context. We use this set together with the Skolemization record of the given tableau to check whether a term is a valid Skolemization. While this avoids having to keep track of all the function symbols occurring in the tableau, it prevents us from directly satisfying Rule DeltaNegAll, which requires the function symbols of all formulas in the tableau. However, intuitively, no expansion step adds a function symbol to the tableau, except Rule DeltaNegAll, which records the symbol in the Skolemization record. Consequently, the set of function symbols of the whole tableau can be reconstructed from the initial set of function symbols together with the generated Skolem symbols. We say that a tableau `T` *preserves the function symbols* of `s` () if the function symbols of the tableau `T` are included in the union of `s` and the Skolemization record of `T`. Note that we cannot obtain a stronger property, since there is no guarantee that a newly generated

Skolem symbol appears in a formula (*e.g.,* when the quantified variable does not occur in the subformula).

Assuming the preservation of function symbols allows us to show that if the proof-checking algorithm returns a positive answer and `s` is the extracted sequence, then `s[0] ▷ s[1]`.

▶ **Lemma 17** ( ). For every substitution σ, set of function symbols `fs`, rule tree `R`, branch `B`, tableaux `T`, `T'`, Skolemization record `r`, and sequence `s`, if `B` is a branch of `T`, the function symbols `fs` are preserved by `T`, `CheckProof__aux sko fs (get_context B T) σ (symbols T) R = ret {| status := true; symbs := r |}` and `RuleTree_to_Sequence__aux B T R = Some s`, such that `s[1] = T'`, then `T ▷ T'`.

By induction, we recover that under the same assumptions, `s` is an expansion sequence.

▶ **Corollary 18** (Proof Checker Produces an Expansion Sequence). For every substitution σ, rule tree `R`, branch `B`, tableaux `T`, `T'`, Skolemization record `r` and sequence `s`, if `B` is a branch of `T`, the function symbols `fs` are preserved by `T`, `CheckProof__aux sko (get_context B T) σ (symbols T) R = ret {| status := true; symbs := r |}` and `RuleTree_to_Sequence__aux B T R = Some s`, then `s` is an expansion sequence.

Proof. By induction on the length of the sequence. As it cannot be empty by Lemma 13, we only need to show the property for the inductive case. Because of Lemma 17, all of the cases except the one for the Rule BetaOr are straightforward. For the branching rule, we recover two sequences $\mathtt{s}_1$ and $\mathtt{s}_2$ by induction hypothesis, the first one starting from `T` and the second one starting from the last tableau of $\mathtt{s}_1$. Recall that the sequence yielded here is `T :: removelast` $\mathtt{s}_1$ `++` $\mathtt{s}_2$. To show that this is an expansion sequence, consider an index $i \in \mathbb{N}$:

- If $i < \#|\mathtt{s}_1| - 1$, then if $i = 0$, we directly conclude using Lemma 17. Otherwise, we use the fact that $\mathtt{s}_1$ is an expansion sequence.
- If $i = \#|\mathtt{s}_1| - 1$, then we need to show that the last tableau of `removelast` $\mathtt{s}_1$ reduces to the first element of $\mathtt{s}_2$. By Lemma 13, the first element of $\mathtt{s}_2$ is the last tableau of $\mathtt{s}_1$, and we conclude this case using the fact that $\mathtt{s}_1$ is an expansion sequence.
- Otherwise, we need to show that there is a reduction between two elements of $\mathtt{s}_2$, which is directly given by the induction hypothesis.

◀

Combining the previous statements is enough to show that if the proof-checking algorithm returns a positive answer, then we can extract an expansion sequence from the rule tree. We can finally show that the last element of this sequence is closed.

▶ **Lemma 19** (Proof Checker Validates only Closed Tableaux). For every substitution σ, rule tree `R`, branches `B` and `B'`, tableau `T`, Skolemization record `r`, and sequence `s`, if `B` is a branch of `T`, `CheckProof__aux sko (get_context B T) σ (symbols T) R = ret {| status := true; symbs := r |}` , `B ++ B'` is a branch of the last tableau of `s` and `RuleTree_to_Sequence__aux B T R = Some s`, then the branch `B ++ B'` is closed in the last tableau of `s`.

Since the algorithm is initiated on the empty branch, *i.e.,* an empty list of branching steps, every branch in the final tableau of the sequence is an extension of this initial branch. Consequently, if the sequence begins with a tableau consisting of a single node labeled by Γ, we can establish the closure of the final tableau by using the previous lemma. Furthermore, the algorithm is invoked on a tableau whose only node is labeled by Γ and whose set of symbols is empty, ensuring that it preserves function symbols. From these two facts, we deduce the soundness theorem.

▶ **Theorem 20** (Soundness of the Proof Checker). For every Γ list of formulas, σ substitution and `R` rule tree, if `CheckProof sko` Γ σ `R = ret true` then `hasTableau sko` Γ σ.

This allows us devise a `tableaux` tactic that, given a list of formulas Γ together with a rule tree R and a substitution σ, decide wether (R, σ) is a valid tableau if the root has context Γ. It suffices to call `CheckProof_sound` and to let Rocq compute the result of `CheckProof sko` Γ σ `R`, effectively providing a fully certified tableau proof checker.

4 Experiments

This section presents an experimental evaluation of TableauxRocq. We compare the proof-checking time of TableauxRocq's certificates against that of the Rocq terms produced by Goeland, on both a set of problems from the TPTP library [57] and on a family of problems specifically designed to stress Skolemization.

Experiments were performed with the Goeland theorem prover on the Grid'5000 platform [6], using an Intel Xeon Gold 5220 CPU (Cascade Lake-SP, x86_64, 2.20 GHz, 18 cores, 96 GiB RAM). The modified version of Goeland, the command lines used for all experiments, as well as the complete benchmark set are provided in the related Zenodo archive [52].

Deskolemization. Goeland is a proof-producing ATP able to export proofs in Rocq (referred to as Rocq terms), SC-TPTP [38], Lisa [39], and LambdaPi [42]. In most of these target formats, proof certificates are obtained via an intermediate deskolemization phase [51, 18, 13]. This transformation turns a Goeland proof into a GS3 [59] proof, a sequent-based calculus closely related to tableaux but restricted to outer Skolemization. As a result, proofs produced using non-outer Skolemization (*e.g.,* [40, 9, 3, 22, 37]) must be transformed, introducing additional computational cost and increasing proof size.

TableauxRocq avoids this limitation entirely: since it natively supports arbitrary Skolemization strategies and substitutions, the translation directly follows the structure of the proof without any prior transformation. In our experiments, we exploit this by comparing the two output pipelines on proofs using the same Skolemization method: Goeland Rocq terms (after deskolemization in the inner case) versus its TableauxRocq output.

On the TPTP Library. We used the TPTP library v9.2.1, from which we extracted first-order (FOF) theorems without equality, resulting in 797 problems. All selected problems are first run with Goeland in outer and inner Skolemization, with a time limit of 300 seconds per problem. Of these, 172 (resp. 177) problems were successfully proved in outer (resp. inner) Skolemization and retained for the remainder of the experiment. For each retained problem, Goeland is run again to generate, in each Skolemization mode: (i) Rocq terms—via deskolemization for inner Skolemization, directly for outer—and (ii) TableauxRocq output. For each problem, we measure the Rocq type-checking time, plus the deskolemization time where applicable.

On this benchmark set, using inner Skolemization, the median checking time of Rocq terms is 0.15s, while the median checking time of the deep embedding in TableauxRocq is 0.45s. On these problems, Rocq is typically faster, as its kernel checks proofs by directly executing OCaml code, whereas TableauxRocq proof-checking algorithm runs through Rocq's `vm_compute` tactic. In addition, this benchmark makes very little use of inner Skolemization: 171 out of 177 problems have identical branch counts in both proof formats, leaving no structural advantage. As expected, TableauxRocq incurs a moderate overhead, but both checking times remain practical.

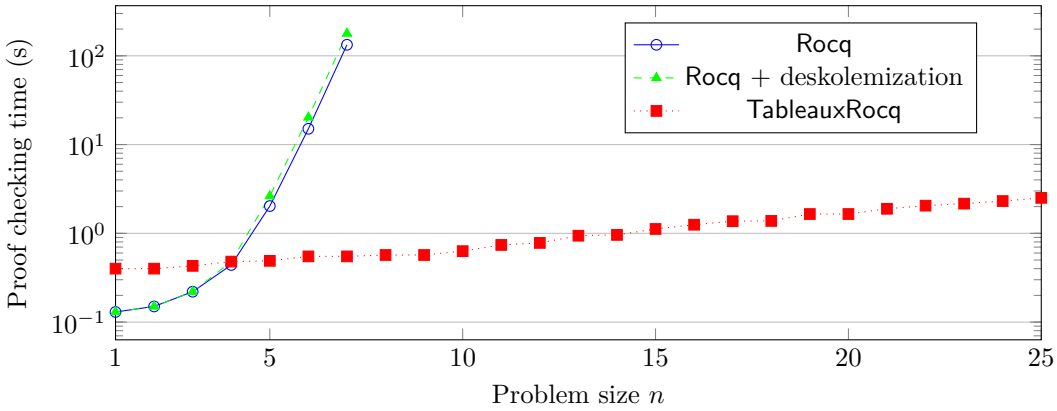

Figure 6 Proof Checking Time for the Skolemization Scaling Benchmarks

The picture changes for the LCL686[2] problem family, which makes significant use of Skolemization. There, TableauxRocq becomes substantially faster as problem size grows: at size 5, Rocq requires 6.2s (+ 0.4s deskolemization) against 3.2s for TableauxRocq; at size 15, the gap widens to 103.6s (+ 9.2s) against 28.5s. This confirms that TableauxRocq's advantage over the Rocq output is most pronounced precisely when Skolemization plays a significant role in the proof.

Using outer Skolemization, Rocq terms require no deskolemization, making this a direct comparison of the two execution models. On this benchmark set, the median checking time of Rocq terms is 0.14s, while the median checking time of the deep embedding in TableauxRocq is 0.42s. Rocq is faster on 169 out of 172 problems, consistently by a factor of roughly 3, illustrating the systematic overhead of `vm_compute` over Rocq's OCaml kernel.

Interestingly, the LCL686 family again sees TableauxRocq become faster at larger sizes (1.2× at size 5, 1.5× at sizes 10 and 15), even without any deskolemization overhead. This shows that the gain of TableauxRocq on complex proofs is not solely due to deskolemization: as proof complexity grows, Rocq per-branch type checking cost increases, whereas TableauxRocq's structural checker scales more efficiently. Comparing with the inner Skolemization results, the gap at size 15 is 101.8s vs 67.1s in outer Skolemization, against 112.8s (including deskolemization) vs 28.5s in inner mode. This shows that deskolemization accounts for roughly half of TableauxRocq advantage in the inner case, with the other half coming from simpler proof term structure.

On the Skolemized Problems. We also perform scaling experiments on a family of problems with the following structure:

$$\Phi_n \triangleq \forall X_1, ..., X_n\,, \bigvee_{i=1}^{n} (\neg P_i(X_i) \wedge \exists Y_i,\ P_i(Y_i))$$

These problems are specifically designed to stress Skolemization. In particular, they typically require extensive reintroduction of Skolem terms in the outer Skolemization mode—and thus in the deskolemization process—whereas the inner Skolemization mode leads to more straightforward proofs.

[2] LC686+1.001.p, LC686+1.005.p, LC686+1.010.p, LC686+1.015.p and LC686+1.020.p.

We generate instances for $1 \leq n \leq 25$ and measure the proof-checking time for both translations, as well as the deskolemization time for the Rocq output, with a time limit of 300 seconds per run. Results are shown in Fig. 6 in logarithmic scale. While inner Skolemization successfully produced proofs for all instances up to $n = 25$, the deskolemization algorithm could only handle up to $n = 7$, as the exponential blowup in proof size caused it to exceed the time limit. Indeed, the deskolemization time grows with proof size, remaining reasonable up to $n = 5$ (0.6s), but rising to 5.2s at $n = 6$ and exceeding 45s at $n = 7$, severely impacting proof generation.

Moreover, as we can see in Fig. 6, the checking time of the (deskolemized) Rocq terms starts to exhibit exponential growth, while the TableauxRocq output displays a linear increase. For small instances ($n \leq 3$), Rocq terms are faster to type-check; the two approaches become comparable at $n = 4$ (0.44s + 0.04s deskolemization for Rocq terms, 0.49s for TableauxRocq). From $n = 5$ onward, TableauxRocq becomes significantly faster: at $n = 7$, Rocq terms require 133.1s (+ 45.1s deskolemization) to type-check, against only 0.55s for TableauxRocq. The curve further shows that TableauxRocq scales well beyond that point, requiring only 2.78s at $n = 25$. Note that our experiments were limited by Goeland's proof generation time, not by TableauxRocq's proof-checking time.

Overall Observations. It is important to note that the two checking pipelines rely on different execution models, which introduces a systematic performance penalty for TableauxRocq, regardless of proof structure. Despite this, TableauxRocq achieves comparable checking performance on the TPTP benchmarks, and significantly better scalability on Skolemization-based problems. Beyond performance considerations, TableauxRocq is suitable for prover integration. The tableau-based format allows proofs to be naturally expressed in a form that closely matches the structure of tableaux proofs, making it easier for ATPs to output such certificates. Moreover, provers using optimized Skolemization strategies do not need to implement a dedicated deskolemization phase: the benefits of optimized Skolemization during proof search outweigh the (slight) additional checking cost introduced by the TableauxRocq translation.

As a side note, while running experiments, some of the proofs generated by Goeland in the TableauxRocq format were rejected by the checker. Upon investigation, these rejections exhibited actual bugs in the Goeland proof output, illustrating how TableauxRocq's certified checker can serve as a debugging tool for ATPs.

5 Related Work

As far as the authors know, free-variable tableaux systems have not been formalized before. In the literature, efforts have mostly focused on sequent-style calculi, which are useful to study proof-theoretic properties, or on resolution-based systems.

Formalization of Tableaux Calculi. Formal study of tableaux calculi are quite rare. In first-order logic, From, Schlichtkrull and Villadsen [36] relie on the ground version of the calculus to prove properties of a sequent-style system. In a more general setting, Blanchette, Popescu and Traytel [15] propose an abstract proof of completeness in Isabelle, which can be instantiated in many proof systems for first-order logic, in particular with ground tableaux. Tableaux have also been studied in a hybrid logic by From [35], but still without free variables.

Other Proof Systems. The Rocq library of undecidability proofs [33] includes a natural deduction system—proved sound—where it is shown that one cannot decide the validity of a formula. Clausified systems have also been formalized, mostly in Isabelle. For instance, the resolution procedure has been formally proved sound and complete by Schlichtkrull [54]. In the same vein, Waldmann et al. [60] gave an abstract specification of ATPs using optimized saturation calculi and prove their completeness. Building on that, Bergeron, Krasnopol and Tourret [12] formalized clause splitting, a more advanced technique for saturation-based theorem provers. In another register, Schlichtkrull, Blanchette and Traytel [55] developed a resolution-based fully certified ATP, while Fleury, Blanchette and Lammich [32] provided a verified SAT solver in Imperative HOL.

6 Conclusion and Future Work

We have introduced the foundations of the TableauxRocq library, a deep embedding of free-variable tableaux in Rocq that also serves as a certified proof checker for tableaux ATPs. TableauxRocq proves that the tableau method is sound for first-order logic, and allows for a seamless proof output from ATPs. However, TableauxRocq is currently restricted to first-order logic without equality, leaving many avenues for future work.

A first axis is to improve the supported logic. In particular, we aim to add equational reasoning, as most TPTP problems involve equality and are therefore not currently certifiable by TableauxRocq. This would enable broader experimental evaluation of TableauxRocq's proof-checking performance. We also plan to integrate additional Skolemization strategies beyond inner and outer Skolemization, and to study whether a modular framework *à la* Cantone and Nicolosi-Asmundo [23] can be incorporated in order to ease the development of new Skolemization methods. More generally, as noted in the introduction, many tableaux provers are not based on first-order logic. Consequently, we want to extend TableauxRocq to support multiple logics, possibly using a *Coq à la Carte* [34] approach to enable sharing different parts of the proofs in multiple logics. Finally, we aim to show completeness of the system.

A second direction concerns the proof checker itself. One of our next goals is to setup an automated extraction of TableauxRocq's proof checker that parses a TPTP-style proof, namely SC-TPTP [38]. This serves two purposes. First, it would make it possible to provide a standalone executable containing the fully certified proof checker. Second, we would not have to rely on Rocq's `vm_compute` tactic, enabling more accurate measurements of TableauxRocq's proof-checking performance and a fair(er) comparison w.r.t. the outputs directly targeting Rocq terms. Ultimately, this could support the development of a hammer-like [26, 16] tactic `tableaux` in Rocq, enabling automated calls to external tableau provers together with certified proof reconstruction.

Declaration of AI Use

We declare that no GenAI (LLMs) was used in the writing of this paper, in the development of the customized Goeland version and in the development of the TableauxRocq library.

References

1 Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-löf à la coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified*

Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024, pages 230–245. ACM, 2024. doi:10.1145/3636501.3636951.

2 Peter B Andrews. Theorem proving via general matings. *Journal of the ACM (JACM)*, 28(2):193–214, 1981.

3 Matthias Baaz and Christian G. Fermüller. Non-elementary Speedups between Different Versions of Tableaux. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *TABLEAUX '95*, volume 918 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 1995.

4 Matthias Baaz, Stefan Hetzl, and Daniel Weller. On the Complexity of Proof Deskolemization. *J. Symb. Log.*, 77(2):669–686, 2012.

5 Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994. URL: https://doi.org/10.1093/logcom/4.3.217, doi:10.1093/LOGCOM/4.3.217.

6 Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013. doi:10.1007/978-3-319-04519-1_1.

7 Bernhard Beckert and Rajeev Goré. Free-variable tableaux for propositional modal logics. *Stud Logica*, 69(1):59–96, 2001. doi:10.1023/A:1013886427723.

8 Bernhard Beckert, Reiner Hähnle, Peter Oel, and Martin Sulzmann. The tableau-based theorem prover 3 tap version 4.0. In *International Conference on Automated Deduction*, pages 303–307. Springer, 1996.

9 Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. The even more liberalized δ-rule in free variable semantic tableaux. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory*, pages 108–119, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

10 Bernhard Beckert and Christian Pape. Incremental theory reasoning methods for semantic tableaux. In *International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 93–109. Springer, 1996.

11 Bernhard Beckert and Joachim Posegga. leantap: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.

12 Ghilain Bergeron, Florent Krasnopol, and Sophie Tourret. Formalizing Splitting in Isabelle/HOL. In Yannick Forster and Chantal Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2025.22, doi:10.4230/LIPIcs.ITP.2025.22.

13 Wolfgang Bibel. *Automated theorem proving*. Vieweg, 1982.

14 Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A verified sat solver framework with learn, forget, restart, and incrementality: Jc blanchette et al. *Journal of automated reasoning*, 61(1):333–365, 2018.

15 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Abstract completeness. *Arch. Formal Proofs*, 2014, 2014. URL: https://www.isa-afp.org/entries/Abstract_Completeness.shtml.

16 Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement day. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2010. doi:10.1007/978-3-642-14203-1_9.

17 Richard Bonichon, David Delahaye, and Damien Doligez. Zenon: An extensible automated theorem prover producing checkable proofs. In *Logic for Programming, Artificial Intelligence and Reasoning*, pages 151–165. Springer, 2007.

18 Richard Bonichon and Olivier Hermant. A Syntactic Soundness Proof for Free-Variable Tableaux with on-the-fly Skolemization. 2013.

19 Guillaume Burel, Guillaume Bury, Raphaël Cauderlier, David Delahaye, Pierre Halmagrand, and Olivier Hermant. First-order automated reasoning with theories: when deduction modulo theory meets practice. *Journal of Automated Reasoning*, 64(6):1001–1050, 2020.

20 Guillaume Bury and David Delahaye. Integrating simplex with tableaux. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 86–101. Springer, 2015.

21 Julie Cailler, Johann Rosain, David Delahaye, Simon Robillard, and Hinde Lilia Bouziane. Goéland: a concurrent tableau-based theorem prover (system description). In *IJCAR 2022-11th International Joint Conference on Automated Reasoning*, volume 13385, pages 359–368, 2022.

22 Domenico Cantone and Marianna Nicolosi Asmundo. A Further and Effective Liberalization of the δ-Rule in Free Variable Semantic Tableaux. In Ricardo Caferra and Gernot Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics, Selected Papers*, volume 1761 of *Lecture Notes in Computer Science*, pages 109–125. Springer, 1998.

23 Domenico Cantone and Marianna Nicolosi-Asmundo. A sound framework for δ-rule variants in free-variable semantic tableaux. *Journal of Automated Reasoning*, 38:31–56, 2007.

24 Raphaël Cauderlier and Pierre Halmagrand. Checking zenon modulo proofs in dedukti. 2015.

25 Arthur Charguéraud. The locally nameless representation. *J. Autom. Reason.*, 49(3):363–408, 2012. URL: `https://doi.org/10.1007/s10817-011-9225-2`, `doi:10.1007/S10817-011-9225-2`.

26 Lukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *J. Autom. Reason.*, 61(1-4):423–453, 2018. URL: `https://doi.org/10.1007/s10817-018-9458-4`, `doi:10.1007/S10817-018-9458-4`.

27 N. G. de Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. volume 34, pages 381–392. North-Holland, 1972.

28 David Delahaye, Damien Doligez, Frédéric Gilbert, Pierre Halmagrand, and Olivier Hermant. Zenon modulo: When achilles outruns the tortoise using deduction modulo. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 274–290. Springer, 2013.

29 Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. `doi:10.1145/3498715`.

30 M. Fitting. *First Order Logic and Automated Theorem Proving*. Springer-Verlag, 2nd edition, 1996.

31 Mathias Fleury. Formalization of logical calculi in isabelle/hol. 2020.

32 Mathias Fleury, Jasmin Christian Blanchette, and Peter Lammich. A verified sat solver with watched literals using imperative hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 158–171, 2018.

33 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A coq library of undecidable problems. In *Proceedings of the Sixth International Workshop on Coq for Programming Languages (CoqPL 2020)*, New Orleans, United States, 2020. Association for Computing Machinery. URL: `https://hal.science/hal-02944217`, `doi:10.1017/S0960129597002302`.

34 Yannick Forster and Kathrin Stark. Coq à la carte: a practical approach to modular syntax with binders. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM*

SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020, pages 186–200. ACM, 2020. doi:10.1145/3372885.3373817.

35 Asta Halkjær From. Synthetic completeness for a terminating seligman-style tableau system. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, University of Turin, Italy, March 2-5, 2020*, volume 188 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.TYPES.2020.5, doi:10.4230/LIPICS.TYPES.2020.5.

36 Asta Halkjær From, Anders Schlichtkrull, and Jørgen Villadsen. A sequent calculus for first-order logic formalized in isabelle/hol. *J. Log. Comput.*, 33(4):818–836, 2023. URL: https://doi.org/10.1093/logcom/exad013, doi:10.1093/LOGCOM/EXAD013.

37 Martin Giese and Wolfgang Ahrendt. Hilbert's ϵ-terms in automated theorem proving. In Neil V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 171–185, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

38 Simon Guilloud, Julie Cailler, Sankalp Gambhir, Auguste Poiroux, Yann Herklotz, Thomas Bourgeat, and Viktor Kunčak. Interoperability of proof systems with sc-tptp. In *Automated Deduction CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings*, page 325340, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-3-031-99984-0_18.

39 Simon Guilloud, Sankalp Gambhir, and Viktor Kunčak. Lisa–a modern proof system. In *14th International Conference on Interactive Theorem Proving*, 2023.

40 Reiner Hähnle and Peter H. Schmitt. The Liberalized δ-Rule in Free Variable Semantic Tableaux. *J. Autom. Reason.*, 13(2):211–221, 1994.

41 Steffen Hölldobler, Norbert Manthey, Tobias Philipp, and Peter Steinke. Generic cdcl-a formalization of modern propositional satisfiability solvers. *POS@ SAT*, 27:89–102, 2014.

42 Gabriel Hondet and Frédéric Blanqui. The new rewriting engine of dedukti (system description). volume 167, pages 35:1–35:16. Schloss Dagstuhl Leibniz-Zentrum für Informatik, 2020. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2020.35, doi:10.4230/LIPICS.FSCD.2020.35.

43 Michael Kohlhase. Higher-order tableaux. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods, 4th International Workshop, TABLEAUX '95, Schloß Rheinfels, St. Goar, Germany, May 7-10, 1995, Proceedings*, volume 918 of *Lecture Notes in Computer Science*, pages 294–309. Springer, 1995. doi:10.1007/3-540-59338-1_43.

44 Anja Petković Komel, Michael Rawson, and Martin Suda. Case study: Verified vampire proofs in the lambdapi-calculus modulo. *arXiv preprint arXiv:2503.15541*, 2025.

45 Boris Konev and Alexander V. Lyaletski. Tableau method with free variables for intuitionistic logic. In Mieczyslaw A. Klopotek, Slawomir T. Wierzchon, and Krzysztof Trojanowski, editors, *Intelligent Information Processing and Web Mining - Proceedings of the International IIS: IIPWM'06 Conference held in Ustrón, Poland, June 19-22, 2006*, volume 35 of *Advances in Soft Computing*, pages 153–162. Springer, 2006. doi:10.1007/3-540-33521-8_15.

46 Hanna Lachnitt, Mathias Fleury, Haniel Barbosa, Jibiana Jakpor, Bruno Andreotti, Andrew Reynolds, Hans-Jörg Schurr, Clark Barrett, and Cesare Tinelli. Improving the smt proof reconstruction pipeline in isabelle/hol. In *16th International Conference on Interactive Theorem Proving (ITP 2025)*, pages 26–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2025.

47 Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. *Handbook of automated reasoning*, 1(7):371–443, 2001.

48 Franz Oppacher and E Suen. Harp: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4(1):69–100, 1988.

49 Michael Rawson, Andrei Voronkov, Johannes Schoisswohl, and Anja Petković Komel. Ground truth: Checking vampire proofs via satisfiability modulo theories. In *International Conference on Automated Deduction*, pages 136–149, 2025.

50 John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

51 Johann Rosain, Richard Bonichon, Julie Cailler, and Olivier Hermant. A generic deskolemization strategy. In Nikolaj Björner, Marijn Heule, and Andrei Voronkov, editors, *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 100 of *EPiC Series in Computing*, pages 246–263. EasyChair, 2024. URL: https://easychair.org/publications/paper/VgpS, doi:10.29007/g1tm.

52 Johann Rosain and Julie Cailler. Tableauxrocq: A deep embedding of free-variable tableaux in rocq — supplementary material. May 2026. doi:10.5281/zenodo.20205704.

53 Philipp Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In *Logic for Programming, Artificial Intelligence and Reasoning*, pages 274–289. Springer, 2008.

54 Anders Schlichtkrull. Formalization of the resolution calculus for first-order logic. *J. Autom. Reason.*, 61(1-4):455–484, 2018. URL: https://doi.org/10.1007/s10817-017-9447-z, doi:10.1007/S10817-017-9447-Z.

55 Anders Schlichtkrull, Jasmin Christian Blanchette, and Dmitriy Traytel. A verified prover based on ordered resolution. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 152–165. ACM, 2019.

56 Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Botsch Nielsen, Nicolas Tabareau, and Théo Winterhalter. Correct and complete type checking and certified erasure for coq, in coq. *J. ACM*, 72(1):8:1–8:74, 2025. doi:10.1145/3706056.

57 G. Sutcliffe. Stepping Stones in the TPTP World. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in Lecture Notes in Artificial Intelligence, pages 30–50, 2024.

58 Dawit Legesse Tirore, Jesper Bengtson, and Marco Carbone. A sound and complete projection for global types. *J. Autom. Reason.*, 69(2):14, 2025. URL: https://doi.org/10.1007/s10817-025-09726-9, doi:10.1007/S10817-025-09726-9.

59 Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic proof theory, Second Edition*, volume 43 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 2000.

60 Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. *J. Autom. Reason.*, 66(4):499–539, 2022. URL: https://doi.org/10.1007/s10817-022-09621-7, doi:10.1007/S10817-022-09621-7.

61 Théo Winterhalter. Dependent ghosts have a reflection for free. *Proc. ACM Program. Lang.*, 8(ICFP):630–658, 2024. doi:10.1145/3674647.