Interoperability of Proof Systems with SC-TPTP (Full Version)

Simon Guilloud[1], Julie Cailler[2], Sankalp Gambhir[1], Auguste Poiroux[1], Yann Herklotz[1], Thomas Bourgeat[1], and Viktor Kunčak[1]

[1] EPFL, Lausanne, Switzerland
[2] University of Lorraine, CNRS, Inria, LORIA, Nancy, France

Abstract. We introduce SC-TPTP, an extension of the TPTP derivation format that supports sequent formalism, enabling seamless proof exchange between interactive theorem provers and first-order automated theorem provers. We provide a way to represent non-deductive steps—Skolemization, clausification, and Tseitin normal form—as deductive steps within the format. Building upon the existing support in the Lisa proof assistant and the Goéland theorem prover, SC-TPTP ecosystem is further enhanced with proof output interfaces for Egg and Prover9, as well as proof reconstruction support for HOL Light, Lean, and Rocq.

1 Introduction

Interoperability of proof systems, in particular automated and interactive theorem provers (ATPs and ITPs), involves several related but distinct challenges, including translation of statements across different logical foundations, alignment of concepts between mathematical libraries, and transfer of proofs. Focussing on proof transfer, this paper proposes a standardized format for export and import of proofs in an extension of first-order logic, integrating it into a variety of proof-producing and proof-consuming systems. Such standardized proof formats have multiple uses [58,48], of which we highlight three. First, ATPs deploy complex algorithms and heuristics to solve difficult problems, making them vulnerable to bugs. A standardized proof format and corresponding proof checker enable proof assistants with a small trusted kernel to verify the correctness of automated provers. This validation helps to evaluate ATPs in benchmarks and competitions, ultimately enhancing the trustworthiness of theorem provers.

Another key use is the collection of proofs from different tools, similar to the TSTP solution library [54], enabling comparisons of proof objects across various metrics, facilitating analysis and serving as training material for AI tools such as large language models. These proof collections are most valuable when the proofs are expressed in a unified format. One final use of standardized proofs, which we illustrate in this paper, is the ability for users of interactive theorem provers (ITPs) to invoke automated theorem provers (ATPs) as tactics. These tactics, known as *hammers*, are available in many ITPs [46,23,35], but do not always ensure the translation and validation of the ATP's proof. Some ITPs treat

the ATP as an oracle, bypassing the formal kernel of the ITP and accepting the ATP-proven statement as an axiom, which necessitates trust in both the ATP and the translation process. Other hammers involve running an internal tactic that, while less powerful, is proof-producing, using lemmas discovered by the ATP or filling in intermediate steps in place of the ATP when possible [46]. This approach requires effectively rediscovering the proof, which can fail.

Challenges. As noted in [11], "One of the main difficulties with proof reconstruction is that each prover has its own (often undocumented) proof format and inference rules". For this reason, most existing proof reconstruction implementations focus on one source and one target system and are done most often with SMT solvers, for which there is an ongoing effort to establish proof formats [16,52,39,15,27,3]. A standardized format with well-documented syntax, proof steps, and parameters will help minimize the duplication of work. In general, if n systems each import proofs from m other systems, this requires $n \cdot m$ implementations, including parsing, translation, and proof reconstruction. With one middle ground format, only $n + m$ such implementations are required.

Approach. We propose to use a list of sequents in an extension of first-order logic (FOL) as the foundation for a concrete proof format, SC-TPTP. FOL is widely used by many relevant tools and, as past work on proof transfer and interoperability highlights, the practical challenges are significant even without the added complexity of unifying diverse logical foundations. For our deduction system, we adopt sequent calculus, a well-established formalism with both theoretical and practical benefits. Notably, it can naturally represent proofs from tableaux calculus, resolution calculus, and natural deduction.

We base our proof format on the TPTP derivation format, due to its prevalence in the field, its human-readable syntax, and the number of tools already compatible with it. TPTP (Thousands of Problems for Theorem Provers) [56] is a large collection of problems used to test and evaluate automated theorem provers. It also serves as a specification format [57] for writing these problems and their derivations, where lists of formulas are each deduced from previous ones or from axioms. However, TPTP does not specify which proof steps are allowed, their parameters, or how formulas are derived. To address this, we introduce SC-TPTP, which stands for *Sequent Calculus - TPTP*.

A preliminary proposal for SC-TPTP was presented in [20], introducing its use in interfacing the Lisa proof assistant [30] and the Goéland ATP [21], enabling proof exchange between the two systems. The initial version raised questions about generalizing the format to more proof systems and representing clausification, as many ATPs rely on clausification via Tseitin normal form and Skolemization. Since these two transformations only preserve satisfiability, they differ from sequent calculus rules. In this paper, we present our approach to representing Tseitin normal form and Skolemization within a purely deductive proof, demonstrating proof reconstruction for the resolution-based prover Prover9.

Contributions. Our main contributions are as follows:

- We present a refined version of the SC-TPTP proof format (and extend the underlying TPTP syntax) [20], introducing schematic symbols and structure sharing for terms and formulas.
- We describe how important non-deductive and satisfiability-preserving steps, in particular Tseitin's transform, can be performed in a purely deductive logical system suitable for interactive theorem provers.
- We make available a set of utilities to handle certification of clausification automatically, to ease adoption of the format by more tools.
- We implement SC-TPTP interfaces for additional ITPs including HOL Light, as a representative of the HOL-family of proof assistants, Lean and Rocq (partial support), as representatives of dependently typed proof assistants and extend the support in Lisa, as a representative of systems based on set theory. This provides the users of these proof assistant with the ability to invoke ATPs supporting SC-TPTP as hammers *with proof reconstruction*. Beyond Goéland, we implement SC-TPTP proof production for additional ATPs: Prover9, a resolution-based solver which makes use of our approach to certifying clausification, and egg, a popular e-graph based tool for reasoning with equality.

Our implementations, including a library of utilities to parse, manipulate, and transform SC-TPTP proofs are available at

https://github.com/orgs/SC-TPTP/repositories

The present text is the full version, including appendices, of [29].

2 The SC-TPTP Format

SC-TPTP is based on the TPTP format, where proofs are represented with *derivations*. A derivation is a list of annotated formulas, each of which is equipped with a name, a role, and, in some cases, an indication of how the formula was deduced. Formulas are essentially part of the FOFX grammar in TPTP.

Example 1. Given the valid formula $\exists x.\ \forall y.\ (d(x) \Rightarrow d(y))$, an automated prover might produce the following SC-TPTP proof for the *drinker's paradox* conjecture:

```
fof(phi, let, ?[X]: ![Y]: (d(X) ⇒ d(Y))).
fof(f1, plain, [d(X), d(Y)] ⟶ [d(Y0), d(Y)],
        inference(hyp, [status(thm), 1], [])).
fof(f2, plain, [d(X)] ⟶ [(d(Y) ⇒ d(Y0)), d(Y)],
        inference(rightImplies, [status(thm)], [f1])).
fof(f3, plain, [d(X)] ⟶ [![Y0] : (d(Y) ⇒ d(Y0)), d(Y)],
        inference(rightForall, [status(thm), 0, 'Y0'], [f2])).
fof(f4, plain, [d(X)] ⟶ [$phi, d(Y)],
        inference(rightExists, [status(thm), 0, $fot(Y)], [f3])).
fof(f5, plain, [] ⟶ [$phi, d(X) ⇒ d(Y)],
```

```
        inference(rightImplies, [status(thm), 1], [f4])).
fof(f6, plain, [] ⟶ [$phi, ![Y]: (d(X) ⇒ d(Y))],
        inference(rightForall, [status(thm), 1, 'Y'], [f5])).
fof(f7, plain, [] ⟶ [$phi],
        inference(rightExists, [status(thm), 0, $fot(X)], [f6])).
```

In general, an annotated formula in SC-TPTP follows the pattern `fof(name, role, statement, annotation)`.

- The **name** of a step is an identifier that later proof steps refer to.
- SC-TPTP recognizes five **roles**: `axiom` indicates that the formula is an axiom and can be used as a leaf in the derivation, `conjecture` does not have a logical meaning in the proof and indicates what the proof is supposed to prove, `plain` indicates a valid statement, deduced from previously statements or axioms, `assumption` is used if the step does not have any premises, and `let`, a new role, is used to introduce shorthands, such as `phi` in the first line of the example proof above. Let statements should not be referred to as premises of future steps; instead, they introduce a new defined constant symbol, which can be used in first-order formulas. Any occurrence of `$phi` is a shorthand (without any variable renaming) of the formula it stands for. Let bindings enable structure sharing for proofs, which often need to repeatedly state the same assumptions. Terms can also be bound to identifiers using annotated terms (not present in normal TPTP syntax), e.g. `fot(t, let, f(X, c))`.
- The **statement** is either a formula, or a sequent. A sequent is a pair of sets of formulas of the form $[\phi_1, ...] \longrightarrow [\psi_1, ...]$. A formula is a shortcut for the sequent with an empty left side and one formula on the right side.
- The **annotation** indicates how the formula was derived and is of the form `inference(stepName, [status(thm), p1, ..., pn], [r1, ..., rn])`. Following the SZS ontologies [55], `status(thm)` indicates the logical status of the formula. In SC-TPTP, all derived statements are `thm`. The $\mathtt{p}_i$ are parameters used to efficiently check and transform the proof, and depend on the specific proof step. The $\mathtt{r}_i$'s are the premise of the steps. They always correspond to names of previous annotated formulas.

Proof Step Levels. The basic steps of sequent calculus, along with substitution of equal terms or equivalent formulas, and the instantiation of free schematic symbols (Appendix B), define the logic of SC-TPTP. These steps are low-level and can be efficiently simulated by any proof system that supports first-order logic. However, outputting strict sequent calculus proofs is a stringent requirement for theorem provers. To facilitate the adoption of the format, SC-TPTP proofs allow various proof steps, which are organized into *levels*.

The first level includes exactly the steps from Appendix B, while subsequent levels are flexible and expected to evolve. The second level contains more advanced proof steps for which an algorithm exists to eliminate them, i.e., processing an SC-TPTP proof and unfolding every occurrence of the proof into level 1 steps.

The third level consists of steps that lack such an implementation but are easily deduced and verified by most proof assistants. With moderate effort, a level 3 step can be converted into a level 2 step. The fourth level contains arbitrary sound deductive steps that cannot be reliably unfolded and may be difficult to implement in a proof-producing form.

These levels offer useful stepping stones to make a solver proof-producing, without needing to implement every proof reconstruction detail, such as congruence closure or negation normal form, which can be time-consuming.

Schematic Symbols. We extend the syntax from [20] by introducing support for schematic formulas, predicates, and functions. Semantically, schematic symbols are halfway between constant symbols and variables: they can be instantiated by terms and formulas but cannot be bound. In higher-order systems, there is no true distinction between schematic symbols and higher-order variables. However, in first-order logic, they are essential for expressing theorem and axiom schemas. For example, in the induction schema of Peano arithmetic, $P(0) \wedge (\forall n.\ P(n) \Rightarrow P(n+1)) \Rightarrow \forall n.\ P(n)$, P is a schematic predicate of arity 1 that can be instantiated by any formula with one free variable. Schematic symbols also allow proving generic theorems useful in first-order logic without relying on specific axioms. For instance, the De Morgan law used to compute negation normal form $\neg(\phi \vee \psi) \Leftrightarrow \neg\phi \wedge \neg\psi$ is generic over the formulas ϕ and ψ, which are formally schematic formulas. Schematic symbols are conservative over strict first-order logic and typically supported by first-order proof assistants, such as in Mizar and Lisa. In Metamath [40], schematic variables are referred to as metavariables, while in Isabelle/FOL, higher-order variables in the meta language can play this role. In SC-TPTP, schematic predicates and functions start with capital letters.

Epsilon Terms. An important addition to SC-TPTP is the support for reasoning with Hilbert's epsilon operator ϵ. This operator is a term level quantifier with the following introduction rule:

$$\frac{\Gamma \vdash \Delta, \phi}{\Gamma \vdash \Delta, \phi[x := (\epsilon x.\phi)]}\ \texttt{rightEpsilon}$$

Informally, $\epsilon x.\ \phi$ denotes an arbitrary element satisfying ϕ, if such an element exists. This addition to sequent calculus is conservative [61], a property known as the Epsilon Theorem. Our motivation for introducing epsilon terms is to allow efficient certification of Skolemization. Constructive deskolemization, the problem of computing a proof of a formula ϕ from a proof of a formula ϕ', where ϕ' is the result of applying Skolemization to ϕ, is hard: all known methods have exponential or non-elementary blow-up [5], depending on whether inner or outer Skolemization is used. This happens even when the proof is cut-free [6].

An epsilon-term $\epsilon x.\phi$ can be seen as a Skolem function $f(y_1, ..., y_n)$ where y_is are all the free variables of ϕ except for x. This enables certification of Skolemization in purely deductive proofs with a linear number of substitution

steps. The downside is that the target system must support epsilon terms, but this is the case in many proof assistants[3]. SC-TPTP proofs use `#` to denote ϵ.

Logic Options to Classify Proofs. SC-TPTP uses annotations in the TPTP header to indicate logical features used in the proof and advanced SC-TPTP features. Such logical annotations (Appendix A), are written similarly to the *Specialist Problem Class* (SPC) annotations in TPTP. A declaration:

```
% Logic    : classical_epsilon_schem_let
```

for example, indicates that the proof uses classical logic (`classical`), permitting sequents with multiple formulas on the right hand side, epsilon choice operator (`epsilon`), schematic predicates or functions (`schem`), and uses defined expressions (`let`) that are defined with the `let` role. Table 1 lists the current set of logical options.

3 Simulating Non-Deductive Proofs

Some common proof strategies used by ATP are not *deductive*, meaning they do not derive a true conclusion from true premises but instead transform the entire problem. Typical examples include *proofs by contradiction*, *Tseitin's transform* and *Skolemization*. Most proof assistants, however, are based on purely deductive logic and cannot accept or prove these steps. When applied correctly, these strategies preserve soundness, and since sequent calculus is complete, any statement proven using non-deductive strategies will also have a proof in strict sequent calculus. However, finding a pure sequent calculus proof can be computationally difficult and time-consuming. Deskolemization, in particular, can blow up the size of the proofs [5,6]. In any case, for all these classes of steps, it is not possible to simply define them as logical steps in SC-TPTP and unfold them locally. Eliminating them from an ATP proof requires modifying the proof globally.

For the sake of conciseness, we briefly illustrate our approach to certifying Tseitin's transform, which makes use of `let` statements and schematic symbols. A detailed explanation of how these non-deductive steps can be performed in both forward and backward proofs in SC-TPTP is available in Appendix E.

Suppose that we have a formula ϕ given as an axiom, and we aim to deduce $\bot$. The structure of the desired proof from the clausified axioms will be as follows:

$$\dfrac{\vdash a_1 \vee ... \vee a_n \qquad ... \qquad \vdash z_1 \vee ... \vee z_n}{\dfrac{\vdots}{\vdash \bot}}$$

Where $a_1 \vee ... \vee a_n$, ..., $z_1 \vee ... \vee z_n$ are the clauses resulting from the Tseitin transform of ϕ. But the clauses are not in general consequences of ϕ, as the transformation only preserves satisfiability.

Suppose ϕ contains $a \wedge b$ as a subterm. We simulate the first step of the Tseitin's transform as follows:

[3] It is called `The` in Mizar, `@` in HOL Light and HOL4, ϵ in Lisa, `Some` in Isabelle and `epsilon` in Rocq and Lean (requires non-constructive axioms).

$$\dfrac{\dfrac{\vdash \phi(a \wedge b)}{A \Leftrightarrow (a \wedge b) \vdash \phi(A)}\ 1. \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash \neg A \vee a}\ 2. \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash \neg A \vee b}\ 3. \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash A \vee \neg a \vee \neg b}\ 4.}{\dfrac{\dfrac{\dfrac{\vdots}{A \Leftrightarrow (a \wedge b) \vdash \bot}}{(a \wedge b) \Leftrightarrow (a \wedge b) \vdash \bot}\ 5.\ \texttt{instPred}}{\vdash \bot}\ 6.\ \texttt{elimIffRefl}}$$

Step 1 is `rightSubstIff`. Steps 2, 3, and 4 are constant-size tautologies. Then the resolution proof follows normally, independently of the $A \Leftrightarrow (a \wedge b)$ on the left-hand side.

Finally, we eliminate the assumption. Step 5 allows instantiating the schematic formula A with an arbitrary formula. To justify equisatisfiability of the Tseitin transform, we need to ensure that A is fresh. This is enforced here by step 5. If A was not a fresh symbol but already appearing in ψ for example, the initial part of the proof would still work, but step 5 would fail. Step 6 finally eliminates the trivial assumption.

Note that every single step can be unfolded into a constant number of level 1 steps, making the whole proof of Tseitin transformation linear. All steps are deductive and can be locally checked, independently of the rest of the proof. The SC-TPTP utilities (Section 4.3) contain tools to automatically perform this transformation on a proof, as well as skolemization, allowing a prover operating on clauses to assume that the input formula is already clausified.

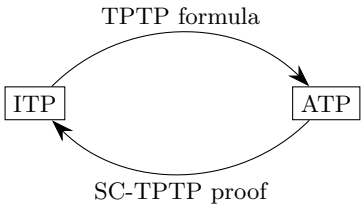

Fig. 1: SC-TPTP for hammers

4 Tools Connected using SC-TPTP

SC-TPTP was already supported by Lisa and Goéland. We add SC-TPTP interfaces to a selection of systems with a variety of logical foundations and principle to demonstrate a broad potential of the approach.

4.1 Automated Theorem Provers Producing SC-TPTP Proofs

The following ATPs are able to produce proofs in the SC-TPTP format, that can be readily verified by one of the ITP in Section 4.2.

Goéland. Goéland [21,19] is an automated theorem prover for first-order logic with equality. It relies on a concurrent tableaux-based proof-search procedure that allows it to conduct a fair branch exploration. The prover can perform

deskolemization [49] and produce machine-checkable proofs in Rocq, LambdaPi and SC-TPTP. App. F provides an example of a proof produced by Goéland.

Prover9. Prover9 [38] is an automated theorem prover for first-order logic with equality that implements the resolution and paramodulation calculi, thus relying on clausification. We implement proof producton for Prover9 in two steps. First, we implement it when the input problem is clausal, that is, the conjecture is `$false` and each axiom is a single clause `[a, b, ...]`, naturally represented as the sequent `[]` ⟶ `[a, b, ...]`. The steps of a resolution proofs directly correspond to proof steps of SC-TPTP: the resolution step is similar to a `cut` step, and the instantiation is exactly the `instFun` step. Then, we separately implement certification of clausification, which uses Prover9 as a subroutine and modify the output proof, as described in Section 3. The resulting procedure takes arbitrary first order formulas as input and outputs verifiable SC-TPTP proofs.

egg. egg [60] (short for E-Graphs Good) is not a traditional ATP but a tool implementing high-performance E-graphs [42] and saturation, often used to optimize programs. It can, however, be seen as solving a fragment of first-order logic with equality, where the equivalence relation is the union of ⇔ and =. egg has already been used for theorem proving in the Coquetier project [17].

egg outputs explanations for pairs of equivalent expressions [26,43]. We add TPTP input for egg, which supports quantified equalities and equivalences as assumptions and conjecture, and transforms egg justifications into SC-TPTP proofs.

egg is also able to find the smallest representative in an equivalence class, motivating using SC-TPTP for more than yes/no answers. We implement the experimental `simplify` role for annotated formulas and terms, exclusive to conjectures, which prompts the solver to find a simplified version of the given term or formula, and prove its equivalence to the original. For example:

```
fof(a1, axiom, (! [Xx]: (Xx = sf(sf(sf(sf(sf(Xx)))))))).
fof(a2, axiom, (! [Xx]: (Xx = sf(sf(sf(Xx)))))).
fot(c, simplify, (p(sf(c)))).
...
fof(f15, plain, [] ⟶ [(p(sf(c)) <⇒ p(c))], inference( ... )).
```

This feature is currently supported only by egg. Appendix F provides examples of a proof produced by our egg wrapper, with level 1 and level 2 steps.

4.2 Interactive Theorem Provers Validating SC-TPTP Proofs

Lisa. Lisa [30] is a proof assistant based on first order logic and set theory. Its logical system is an extension of sequent calculus, making proof import straightforward, whereas exporting a problem requires a simple form of monomorphization. We developed a dedicated tactic for several provers, illustrated below. The proof reconstruction implementation is largely shared across these tactics.

```
val divide_mult_shift = Theorem((
        ∀(x, x/t1 ≡≡ x), ∀(x, ∀(y, x/y ≡≡ t1/(y/x))),
        ∀(x, ∀(y, (x/y)*y ≡≡ x))) |- ((t2/t3)*(t3/t2))/t1 ≡≡ t1):
    have(thesis) by Egg

val drinkers2 = Theorem(∃(x, ∀(y, d(x) ⟹ d(y)))):
    have(thesis) by Goeland

val thm = Theorem((∀(x, P(x)) ∨ ∀(y, Q(y))) ⟹ (P(ϕ) ∨ Q(ϕ)) ):
    have(thesis) by Prover9
```

HOL Light. HOL Light [31] is an LCF-style interactive theorem prover based on higher-order logic implemented in OCaml. We contribute an extensible way to generically construct interfaces to external provers for use in HOL Light, and a TPTP parser in OCaml using `ocamllex` and `menhir` supporting the FOFX fragment with the SC-TPTP extensions decribed here. Our implementation provides a function `sctptp_tac`, which, given a way to construct invocations to an external solver, can be instantiated in one line as a reusable tactic. Instantiations `EGG`, `GOELAND`, and `PROVER9` for the respective ATPs are included. Each accepts a list of existing theorems to use as axioms, and a sequent to prove, and returns a `thm` object proving the sequent if a proof was found and reconstructed. The following are examples of HOL Light invocations of Goéland and egg:

```
(* syntax: PROVER [premises] [antecedants] conclusion *)
# let drinkers = GOELAND [] [] `?x:ind. !y. d(x) ⟹ d(y)`;;
val drinkers : thm = |- exists x. forall y. d x ⟹ d y
# let div_mult_shift = EGG [] [
    `!x. x/t1 = x`; `!x y. x/y = t1/(y/x)`; `!x y. (x/y)*y = x`
    ] `((t2/t3)*(t3/t2))/t1 = t1`;;
val div_mult_shift : thm =
  forall v3 v5. v3/v5 = t1/(v5/v3), forall v3 v5. v3/v5 * v5 = v3,
  forall v3. v3/t1 = v3 |- (t2/ t3 * t3/t2) / t1 = t1
```

To transform the original problem (possibly containing higher-order terms) into a first-order SC-TPTP querry, we adapted the monomorphization procedure from the existing implementation of the Meson tactic in HOL Light. All higher-order terms are abstracted into named untyped first-order constants. During proof reconstruction, however, the types for all terms must be recovered. To this end, we implement a type inference and unification procedure similar to that of the simply-typed lambda calculus.

Lean. Lean [25] is an interactive theorem prover based on dependent type theory. In our framework, proof reconstruction is implemented using the standard Lean 4 tactic interface. We developed custom tactics, `egg`, `prover9`, and `goeland`, that automate the entire workflow, including sequent export to TPTP, invocation of the underlying ATP, and subsequent parsing, reification, reconstruction, and checking of the proof within Lean 4, allowing, e.g., the following use.

```
example ( α : Type) [Nonempty α] (d : α → Prop) :
  ∃ y : α, ∀ x : α, (d x → d y) := by
  goeland
```

Our development uses the monomorphization and reification tools from the Lean Auto project [2]. We adapt its mechanism for exporting formulas from the TH0 to the FOF format within the TPTP framework. In the translation of TPTP expressions to Lean expressions, we use type inference to recover information about types of variables. Our proof reconstruction system is implemented using a backward proof strategy within a single global context, allowing us to leverage the Lean 4 tactic proof mode.

Rocq. Similar to the more recent Lean, Rocq is a proof assistant based on a dependently typed calculus. Building upon a proof validating interface between Rocq and egg in the Coquetier project [17], we have developed a prototype that uses SC-TPTP to reconstruct proof statements in the intuitionistic fragment of first-order logic. This interface is currently more limited than one for Lean because it does not make use of monomorphization and supports less level 3 proof steps, but it already provides another path for validating some of the proofs of automated solvers.

4.3 SC-TPTP Utilities and Central Repository

To support the SC-TPTP format, we release a library of tools and utilities to handle SC-TPTP proofs at https://github.com/SC-TPTP/sc-tptp. The library provides a parser and an independent proof checker for SC-TPTP proofs, as well as functions able to unfold level 2 proof steps into level 1 proof, such as a proof-producing implementation of an e-graph able to unfold the `congruence` proof step. It also contains a module able to certify clausification and extend proofs of formulas in CNF provided by ATPs relying on clausification to form a complete proof of the original formula, which is central to SC-TPTP support for Prover9. The library further contains helpers, examples, test cases and documentation. The repository also contains links and forks of tools with SC-TPTP support. The suite of tools is under active development.

5 Related Work

An excellent review of hammers for proof assistants is [11]. In the domain of proof system interoperability but in a very different direction, logical framework such as dedukti [4] and the Lambdapi [1] proof assistant aim at unifying different mathematical foundations to facilitate the sharing of mathematical libraries [12]. SC-TPTP on the other hand is a proof exchange format, that is, a *data* exchange format, and is designed as such. More practically, Dedukti/Lambdapi files are not adequate to represent sequent calculus proof data in such a way that it is easy to export and import. Accessorily, it is also not possible to naturally represent sequents in Dedukti with set semantics. It is also worth noting that

most proof systems (ATPs and ITPs) already have some support for the TPTP format.

In the SMT community, diverse proof formats have emerged: LFSC [53] was originally used by cvc5 [8] and its predecessors. Z3 [24] has its own format [41]. While there is yet no universally accepted format, efforts have been carried toward this goal [10,33]. The Alethe [51] format is based on SMT-LIB [9], the SMT equivalent of the TPTP World, and supported by cvc5 and veriT [18].

The SAT solving community has developed a series of increasingly more optimized and efficient format for certifying proofs of SAT solvers: the DRAT [59,47] format, LRAT [22] and GRAT [37], VeriPB [13,28] and the corresponding VeriPB-CakePB checker [14] These formats are used in the yearly SAT Competition [32] and produced by state-of-the-art SAT solvers. This long chain of improvements suggests that the current version of SC-TPTP will similarly go through various refinements and improvements over the years.

Closer to our approach, extension of the TPTP syntax to the connection calculus [45] was implemented in the leanCoP [44] and Connect++ [34] provers, although intermediate steps do not correspond to logical formulas. Efforts for supporting the connection calculus and Connect++ in SC-TPTP are ongoing.

Independently, the TESC format [7] offers a sequent-based proof format capable of compiling and verifying solutions from Vampire [36] and E [50].

6 Conclusion

We have presented SC-TPTP, a proof format for representing proofs in first-order logic with equality, enabling verification of proofs produced by first-order ATPs and transfer of first-order proofs between proof systems. We demonstrated how clausification can be represented in the format and implemented SC-TPTP interfaces in several types of automated and interactive theorem provers.

While in proof assistants with first-order foundations all statements can easily be fed into first-order ATP, systems based on more complex logics, in particular dependent types, typically require monomorphization to maximize usability. This is a largely independent process implemented by hammers in these systems; we hope in the future to more closely integrate SC-TPTP with existing hammers in ITPs.

Acknowledgements The authors thank reviewers and members of the ATP community who expressed interest and provided feedback. They address particular thanks to Geoff Sutcliffe and Sean Holden.

References

1. Deducteam/lambdapi. Deducteam (Sep 2024)
2. Leanprover-community/lean-auto. leanprover-community (Feb 2025)

3. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. pp. 135–150. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25379-9_12
4. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Dedukti: A Logical Framework based on the $λ$$Π$-Calculus Modulo Theory (2023). https://doi.org/10.48550/ARXIV.2311.07185
5. Avigad, J.: Eliminating definitions and Skolem functions in first-order logic. In: Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. pp. 139–146 (Jun 2001). https://doi.org/10.1109/LICS.2001.932490
6. Baaz, M., Hetzl, S., Weller, D.: On the Complexity of Proof Deskolemization. The Journal of Symbolic Logic **77**(2), 669–686 (2012)
7. Baek, S.: A Formally Verified Checker for First-Order Proofs. In: Cohen, L., Kaliszyk, C. (eds.) 12th International Conference on Interactive Theorem Proving (ITP 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 193, pp. 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ITP.2021.6
8. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: Cvc5: A Versatile and Industrial-Strength SMT Solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, vol. 13243, pp. 415–442. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
9. Barrett, C.W., Stump, A., Tinelli, C.: The SMT-LIB Standard Version 2.0 (2010)
10. Besson, F., Fontaine, P., Théry, L.: A Flexible Proof Format for SMT: A Proposal (Aug 2011)
11. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. Journal of Formalized Reasoning **9**(1), 101–148 (Jan 2016). https://doi.org/10.6092/issn.1972-5787/4593
12. Blanqui, F.: Translating HOL-Light proofs to Coq. In: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning. pp. 1–18. https://doi.org/10.29007/6k4x
13. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified Dominance and Symmetry Breaking for Combinatorial Optimisation. Journal of Artificial Intelligence Research **77**, 1539–1589 (Aug 2023). https://doi.org/10.1613/jair.1.14296
14. Bogaerts, B., McCreesh, C., Myreen, M.O., Nordstrom, J., Oertel, A., Tan, Y.K.: Documentation of VeriPB and CakePB for the SAT Competition 2023 (2023)
15. Böhme, S., Fox, A.C.J., Sewell, T., Weber, T.: Reconstruction of Z3's Bit-Vector Proofs in HOL4 and Isabelle/HOL. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. pp. 183–198. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25379-9_15
16. Böhme, S., Weber, T.: Fast LCF-Style Proof Reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving. pp. 179–194. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14052-5_14
17. Bourgeat, T.: Specification and Verification of Sequential Machines in Rule-Based Hardware Languages. Thesis, Massachusetts Institute of Technology (Feb 2023)
18. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: veriT: An Open, Trustable and Efficient SMT-Solver. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. pp. 151–156. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_12

19. Cailler, J.: Designing an Automated Concurrent Tableau-Based Theorem Prover for First-Order Logic. Ph.D. thesis, Université de Montpellier (Dec 2023)
20. Cailler, J., Guilloud, S.: SC-TPTP: An Extension of the TPTP Derivation Format for Sequent-Based Calculus. In: 9th Workshop on Practical Aspects of Automated Reasoning (2024)
21. Cailler, J., Rosain, J., Delahaye, D., Robillard, S., Bouziane, H.L.: Goéland: A Concurrent Tableau-Based Theorem Prover (System Description). In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) Automated Reasoning. pp. 359–368. Lecture Notes in Computer Science, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-10769-6_22
22. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient Certified RAT Verification. In: de Moura, L. (ed.) Automated Deduction – CADE 26. pp. 220–236. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_14
23. Czajka, Ł., Kaliszyk, C.: Hammer for Coq: Automation for Dependent Type Theory. Journal of Automated Reasoning **61**(1), 423–453 (Jun 2018). https://doi.org/10.1007/s10817-018-9458-4
24. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
25. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean Theorem Prover (System Description). In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25, vol. 9195, pp. 378–388. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_26
26. Flatt, O., Coward, S., Willsey, M., Tatlock, Z., Panchekha, P.: Small Proofs from Congruence Closure. In: #PLACEHOLDER_PARENT_METADATA_VALUE#. pp. 75–83. TU Wien Academic Press (Oct 2022). https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_13
27. Fontaine, P., Marion, J.Y., Merz, S., Nieto, L.P., Tiu, A.: Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants. In: Hermanns, H., Palsberg, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 167–181. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11691372_11
28. Gocht, S., Nordström, J.: Certifying Parity Reasoning Efficiently Using Pseudo-Boolean Proofs — Supplemental Material (Sep 2022). https://doi.org/10.5281/zenodo.7083485
29. Guilloud, S., Cailler, J., Gambhir, S., Poiroux, A., Herklotz, Y., Bourgeat, T., Kunčak, V.: Interoperability of Proof Systems with SC-TPTP. In: Automated Deduction – CADE 30. Springer Nature Switzerland, Stuttgart (2025)
30. Guilloud, S., Gambhir, S., Kuncak, V.: LISA – A Modern Proof System. In: 14th Conference on Interactive Theorem Proving. pp. 17:1–17:19. Leibniz International Proceedings in Informatics, Daghstuhl, Bialystok (2023)
31. Harrison, J.: HOL Light: An Overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics, vol. 5674, pp. 60–66. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_4
32. Heule, M.J.H., Iser, M., Järvisalo, M., Suda, M.: Proceedings of SAT Ccompetition 2024: Solver, Benchmark and Proof Checker Descriptions (2024)
33. Hoenicke, J., Schindler, T.I.: A Simple Proof Format for SMT. In: International Workshop on Satisfiability Modulo Theories (2022)

34. Holden, S.B.: Connect++: A New Automated Theorem Prover Based on the Connection Calculus. In: Otten, J., Bibel, W. (eds.) Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi (AReCCa 2023) Affiliated with the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2023), Prague, Czech Republic, September 18, 2023. CEUR Workshop Proceedings, vol. 3613, pp. 95–106. CEUR-WS.org (2023)
35. Kaliszyk, C., Urban, J.: Learning-Assisted Automated Reasoning with Flyspeck. Journal of Automated Reasoning **53**(2), 173–213 (Aug 2014). https://doi.org/10.1007/s10817-014-9303-3
36. Kovács, L., Voronkov, A.: First-Order Theorem Proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 1–35. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
37. Lammich, P.: The GRAT Tool Chain. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing – SAT 2017. pp. 457–463. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_29
38. McCune, W.: Prover9 and Mace4
39. McLaughlin, S., Barrett, C., Ge, Y.: Cooperating Theorem Provers: A Case Study Combining HOL-Light and CVC Lite. Electron. Notes Theor. Comput. Sci. **144**(2), 43–51 (Jan 2006). https://doi.org/10.1016/j.entcs.2005.12.005
40. Megill, N., Wheeler, D.A.: Metamath: a computer language for mathematical proofs. Lulu. com (2019)
41. Moura, L.D., Bjørner, N.S.: Proofs and Refutations, and Z3. In: LPAR Workshops (2008)
42. Nelson, G., Oppen, D.C.: Fast Decision Procedures Based on Congruence Closure. Journal of the ACM **27**(2), 356–364 (Apr 1980). https://doi.org/10.1145/322186.322198
43. Nieuwenhuis, R., Oliveras, A.: Proof-Producing Congruence Closure. In: Giesl, J. (ed.) Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3467, pp. 453–468. Springer (2005). https://doi.org/10.1007/978-3-540-32033-3_33
44. Otten, J., Bibel, W.: leanCoP: Lean connection-based theorem proving. Journal of Symbolic Computation **36**(1), 139–161 (Jul 2003). https://doi.org/10.1016/S0747-7171(03)00037-3
45. Otten, J., Holden, S.B.: A Syntax for Connection Proofs. In: Otten, J., Bibel, W. (eds.) Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi (AReCCa 2023) Affiliated with the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2023), Prague, Czech Republic, September 18, 2023. CEUR Workshop Proceedings, vol. 3613, pp. 84–94. CEUR-WS.org (2023)
46. Paulson, L.C., Susanto, K.W.: Source-Level Proof Reconstruction for Interactive Theorem Proving. In: Schneider, K., Brandt, J. (eds.) Theorem Proving in Higher Order Logics. pp. 232–245. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74591-4_18
47. Rebola-Pardo, A., Cruz-Filipe, L.: Complete and Efficient DRAT Proof Checking. 2018 Formal Methods in Computer Aided Design (FMCAD) pp. 1–9 (Oct 2018). https://doi.org/10.23919/FMCAD.2018.8602993

48. Reger, G., Suda, M.: Checkable Proofs for First-Order Theorem Proving. In: ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements. pp. 55–63. EasyChair EPiC Series in Computing (2017)
49. Rosain, J., Bonichon, R., Cailler, J., Hermant, O.: A Generic Deskolemization Strategy. In: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning. pp. 246–227. https://doi.org/10.29007/g1tm
50. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, Higher, Stronger: E 2.3. In: Fontaine, P. (ed.) Automated Deduction – CADE 27. pp. 495–507. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
51. Schurr, H.J., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: Towards a Generic SMT Proof Format (extended abstract). Electronic Proceedings in Theoretical Computer Science **336**, 49–54 (Jul 2021). https://doi.org/10.4204/EPTCS.336.6
52. Schurr, H.J., Fleury, M., Desharnais, M.: Reliable Reconstruction of Fine-grained Proofs in a Proof Assistant. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction – CADE 28. pp. 450–467. Lecture Notes in Computer Science, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-79876-5_26
53. Stump, A., Oe, D.: Towards an SMT proof format. In: Proceedings of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning. pp. 27–32. SMT '08/BPR '08, Association for Computing Machinery, New York, NY, USA (Jul 2008). https://doi.org/10.1145/1512464.1512470
54. Sutcliffe, G.: TPTP, TSTP, CASC, etc. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) Computer Science – Theory and Applications. pp. 6–22. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74510-5_4
55. Sutcliffe, G.: The SZS Ontologies for Automated Reasoning Software. In: Rudnicki, P., Sutcliffe, G., Konev, B., Schmidt, R.A., Schulz, S. (eds.) Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008)
56. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. Journal of Automated Reasoning **59**(4), 483–502 (Dec 2017). https://doi.org/10.1007/s10817-017-9407-7
57. Sutcliffe, G.: The logic languages of the TPTP world. Logic Journal of the IGPL **31**(6), 1153–1169 (Nov 2023). https://doi.org/10.1093/jigpal/jzac068
58. Weber, T.: Designing Proof Formats: A User's Perspective. In: International Workshop on Proof Exchange for Theorem Proving (Aug 2011)
59. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In: Sinz, C., Egly, U. (eds.) Theory and Applications of Satisfiability Testing – SAT 2014. pp. 422–429. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31
60. Willsey, M., Nandi, C., Wang, Y.R., Flatt, O., Tatlock, Z., Panchekha, P.: Egg: Fast and extensible equality saturation. Proceedings of the ACM on Programming Languages **5**(POPL), 23:1–23:29 (Jan 2021). https://doi.org/10.1145/3434304
61. Zach, R.: Semantics and Proof Theory of the Epsilon Calculus. In: Ghosh, S., Prasad, S. (eds.) Logic and Its Applications. pp. 27–47. Springer, Berlin, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54069-5_4

A Proof Construct Annotations

`classical`	The proof requires is carried in classical logic and contains sequents with multiple formulas on the right-hand side, or uses higher level steps only valid in classical logic.
`epsilon`	The proof makes use of ϵ-terms.
`propext`	Usually, substitution of equivalent formulas (`leftSubstIff` and `rightSubstIff`) is a meta theorem of first-order logic and can be unfolded. However, this is not the case if substitution is carried below epsilon quantifiers. In this case, propositional extensionality is required.
`schem`	The proof uses schematic predicates or functions, which start with a capital letter.
`let`	The proof uses defined expressions starting with `$` that are defined with the `let` role
`fot`	The proof uses top-level annotated terms, with the `let` or `simplify` role.

Table 1: Annotations defined for fine-grained logical properties. We expect more will exist in the future.

B Level 1 Proof Steps of SC-TPTP

Rule name	Premises	Rule	Parameters
`leftFalse`	0	$\dfrac{}{\Gamma, \bot \vdash \Delta}$	`i:Int`: Index of $\bot$ on the left
`rightTrue`	0	$\dfrac{}{\Gamma \vdash \top, \Delta}$	`i:Int`: Index of $\top$ on the right
`hyp`	0	$\dfrac{}{\Gamma, A \vdash A, \Delta}$	`i:Int`: Index of A on the left `j:Int`: Index of A on the right
`leftWeaken`	1	$\dfrac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}$	`i:Int`: Index of A on the left
`rightWeaken`	1	$\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}$	`i:Int`: Index of A on the right
`cut`	2	$\dfrac{\Gamma \vdash A, \Delta \qquad \Sigma, A \vdash \Pi}{\Gamma, \Sigma \vdash \Delta, \Pi}$	`i:Int`: Index of A on the right of the first premise
`leftAnd`	1	$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$	`i:Int`: Index of $A \wedge B$ on the left
`leftOr`	2	$\dfrac{\Gamma, A \vdash \Delta \qquad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \vee B \vdash \Delta, \Pi}$	`i:Int`: Index of $A \vee B$ on the left
`leftImplies`	2	$\dfrac{\Gamma \vdash A, \Delta \qquad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \Rightarrow B \vdash \Delta, \Pi}$	`i:Int`: Index of $A \Rightarrow B$ on the left
`leftIff`	1	$\dfrac{\Gamma, A \Rightarrow B, B \Rightarrow A \vdash \Delta}{\Gamma, A \Leftrightarrow B \vdash \Delta}$	`i:Int`: Index of $A \Leftrightarrow B$ on the left
`leftNot`	1	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}$	`i:Int`: Index of $\neg A$ on the left
`leftExists`	1	$\dfrac{\Gamma, A[x := y] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}$	`i:Int`: Index of $\exists x.A$ on the left `y:String`: Variable in place of x in the premise
`leftForall`	1	$\dfrac{\Gamma, A[x := t] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}$	`i:Int`: Index of $\forall x.A$ on the left `t:Term`: Term in place of x in the premise
`rightAnd`	2	$\dfrac{\Gamma \vdash A, \Delta \qquad \Sigma \vdash B, \Pi}{\Gamma, \Sigma \vdash A \wedge B, \Delta, \Pi}$	`i:Int`: Index of $A \wedge B$ on the right
`rightOr`	1	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}$	`i:Int`: Index of $A \vee B$ on the right
`rightImplies`	1	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}$	`i:Int`: Index of $A \Rightarrow B$ on the right
`rightIff`	2	$\dfrac{\Gamma \vdash A \Rightarrow B, \Delta \qquad \Sigma \vdash B \Rightarrow A, \Pi}{\Gamma \vdash A \Leftrightarrow B, \Delta}$	`i:Int`: Index of $A \Leftrightarrow B$ on the right
`rightNot`	1	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$	`i:Int`: Index of $\neg A$ on the right
`rightExists`	1	$\dfrac{\Gamma \vdash A[x := t], \Delta}{\Gamma \vdash \exists x.A, \Delta}$	`i:Int`: Index of $\exists x.A$ on the right `t:Term`: Term in place of x in the premise
`rightForall`	1	$\dfrac{\Gamma \vdash A[x := y], \Delta}{\Gamma \vdash \forall x.A, \Delta}$	`i:Int`: Index of $\forall x.A$ on the right `y:String`: Variable in place of x in the premise
`rightRefl`	0	$\dfrac{}{\Gamma \vdash t = t, \Delta}$	`i:Int`: Index of $t = t$ on the right

Table 2: Level 1 rules of SC-TPTP, part 1.

Rule name	Premises	Rule	Parameters
`rightSubst`	1	$\frac{\Gamma \vdash P(t), \Delta}{\Gamma, t = u \vdash P(u), \Delta}$	`i:Int`: Index of $t = u$ on the left `backward:Int`: If 1, the substitution is done backward `P(Z):Term`: Shape of the predicate on the right `Z:String`: Variable indicating where the substitution takes place
`leftSubst`	1	$\frac{\Gamma, P(t) \vdash \Delta}{\Gamma, t = u, P(u) \vdash \Delta}$	`i:Int`: Index of $t = u$ on the left `backward:Int`: If 1, the substitution is done backward `P(Z):Term`: Shape of the predicate on the left `Z:String`: Variable indicating where the substitution takes place
`rightSubstIff`	1	$\frac{\Gamma \vdash R(\phi), \Delta}{\Gamma, \phi \Leftrightarrow \psi \vdash R(\psi), \Delta}$	`i:Int`: Index of $\phi \Leftrightarrow \psi$ on the left `backward:Int`: If 1, the substitution is done backward `R(Z):Var`: Shape of the predicate on the right `Z:String`: Variable indicating where in P the substitution takes place
`leftSubstIff`	1	$\frac{\Gamma, R(\phi) \vdash \Delta}{\Gamma, \phi \Leftrightarrow \psi, R(\psi) \vdash \Delta}$	`i:Int`: Index of $\phi \Leftrightarrow \psi$ on the left `backward:Int`: If 1, the substitution is done backward `R(Z):Var`: Shape of the predicate on the right `Z:String`: Schematic formula indicating where in P the substitution takes place
`instFun`	1	$\frac{\Gamma[F_X] \vdash \Delta[F_X]}{\Gamma[F_X := t_X] \vdash \Delta[F_X := t_X]}$	`'F': String`: Schematic function to substitute. `t:Term`: Term, possibly containing $X_1, ..., X_n$, to instantiate F with `Xs: Seq[String]`: Variables parametrizing t. The length gives the arity of F
`instPred`	1	$\frac{\Gamma[P_X] \vdash \Delta[P_X]}{\Gamma[P_X := \phi_X] \vdash \Delta[P_X := \phi_X]}$	`'P': String`: Schematic predicate to substitute. ϕ`: Formula`: Formula, possibly containing $X_1, ..., X_n$, to instantiate F with `Xs: Seq[String]`: Variables parametrizing ϕ. The length gives the arity of P
`rightEpsilon`	1	$\frac{\Gamma \vdash A[x := t], \Delta}{\Gamma \vdash A[x := \epsilon x.A], \Delta}$	`A:Formula`: Formula defining the epsilon-term `X:String`: Variable being substituted in A `t:Term`: Term in place of x in the premise
`leftEpsilon`	1	$\frac{\Gamma, A[x := y] \vdash \Delta}{\Gamma, A[x := \epsilon x.A] \vdash \Delta}$	`i:Int`: Index of $A[x := y]$ on the left of the premise `y:String`: Variable in place of x in the premise

Table 3: Level 1 rules of SC-TPTP, part 2.

C Current Level 2 Proof Steps of SC-TPTP

Rule name	Premises	Rule	Parameters
`congruence`	0	$\dfrac{}{\Gamma, \Delta}$	No parameters Γ contains a set of ground equalities such that P and Q are congruents
`leftHyp`	0	$\dfrac{}{\Gamma, A, \neg A \vdash \Delta}$	`i:Int`: Index of A on the left
`leftNotAnd`	2	$\dfrac{\Gamma, \neg A \vdash \Delta \qquad \Sigma, \neg B \vdash \Pi}{\Gamma, \Sigma, \neg(A \wedge B) \vdash \Delta, \Pi}$	`i:Int`: Index of $\neg(A \wedge B)$ on the left
`leftNotOr`	1	$\dfrac{\Gamma, \neg A, \neg B \vdash \Delta}{\Gamma, \neg(A \vee B) \vdash \Delta}$	`i:Int`: Index of $\neg(A \vee B)$ on the left
`leftNotImplies`	1	$\dfrac{\Gamma, A, \neg B \vdash \Delta}{\Gamma, \neg(A \Rightarrow B) \vdash \Delta}$	`i:Int`: Index of $\neg(A \Rightarrow B)$ on the left
`leftNotIff`	2	$\dfrac{\Gamma, \neg(A \Rightarrow B) \vdash \Delta \qquad \Sigma, \neg(B \Rightarrow A) \vdash \Pi}{\Gamma, \Sigma, \neg(A \Leftrightarrow B) \vdash \Delta, \Pi}$	`i:Int`: Index of $\neg(A \Leftrightarrow B)$ on the left
`leftNotNot`	1	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, \neg\neg A \vdash \Delta}$	`i:Int`: Index of $\neg\neg A$ on the left
`leftNotEx`	1	$\dfrac{\Gamma, \neg A[x := t] \vdash \Delta}{\Gamma, \neg\exists x.A \vdash \Delta}$	`i:Int`: Index of $\neg\exists x.A$ on the left `t:Term`: Term in place of x in the premise
`leftNotAll`	1	$\dfrac{\Gamma, \neg A \vdash \Delta}{\Gamma, \neg\forall x.A \vdash \Delta}$	`i:Int`: Index of $\neg\forall x.A$ on the left `y:String`: Variable in place of x in the premise

Table 4: Current Level 2 rules of SC-TPTP, which can be unfolded into level 1 rules with the SC-TPTP utils.

D Current Level 3 Proof Steps of SC-TPTP

Rule name	Premises	Rule	Parameters
`rightReflIff`	0	$\dfrac{}{\Gamma \vdash A \Leftrightarrow A, \Delta}$	`i:Int`: Index of $A \Leftrightarrow A$ on the right
`rightSubstMulti`	1	$\dfrac{\Gamma \vdash P(\boldsymbol{t}), \Delta}{\Gamma \vdash P(\boldsymbol{u}), \Delta}$	`i_1`, ..., `i_n`: Index of formulas $t_j = u_j$ on the left `P(Z_1, ... , Z_n):Term`: Shape of the formula on the right `Z_1`, ..., `Z_n`: Var: variables indicating where to substitute
`leftSubstMulti`	1	$\dfrac{\Gamma, P(\boldsymbol{t}) \vdash \Delta}{\Gamma, P(\boldsymbol{u}) \vdash \Delta}$	`i_1`, ..., `i_n`: Index of formula $t_j = u_j$ on the left `P(Z_1, ... , Z_n):Term`: Shape of the formula on the left `Z_1`, ..., `Z_n`: Var: variables indicating where to substitute
`rightSubstEqForallLocal`	1	$\dfrac{\Gamma, \forall \boldsymbol{x}.\phi(\boldsymbol{x}) = \psi(\boldsymbol{x}) \vdash R(\phi(\boldsymbol{t})), \Delta}{\Gamma, \forall \boldsymbol{x}.\phi(\boldsymbol{x}) = \psi(\boldsymbol{x}) \vdash R(\psi(\boldsymbol{t})), \Delta}$	`i:Int`: Index of $\forall \boldsymbol{x}.\phi(\boldsymbol{x}) = \psi(\boldsymbol{x})$ on the left `R(Z):Var`: Shape of the predicate on the right `Z:Form`: Variable indicating where in P the substitution takes place
`rightSubstEqForall`	2	$\dfrac{\Gamma \vdash R(\phi(t)), \Delta \qquad \Sigma \vdash \forall x.\phi(x) = \psi(x), \Pi}{\Gamma, \Sigma \vdash R(\psi(t)), \Delta, \Pi}$	`i:Int`: Index of $\forall x.\phi(x) = \psi(x)$ on the right of the second premise `R(Z):Var`: Shape of the predicate on the right `Z:Var`: Variable indicating where in P the substitution takes place
`rightSubstIffForallLocal`	1	$\dfrac{\Gamma, \forall \boldsymbol{x}.\phi(\boldsymbol{x}) \Leftrightarrow \psi(\boldsymbol{x}) \vdash R(\phi(\boldsymbol{t})), \Delta}{\Gamma, \forall \boldsymbol{x}.\phi(\boldsymbol{x}) \Leftrightarrow \psi(\boldsymbol{x}) \vdash R(\psi(\boldsymbol{t})), \Delta}$	`i:Int`: Index of $\forall \boldsymbol{x}.\phi(\boldsymbol{x}) \Leftrightarrow \psi(\boldsymbol{x})$ on the left `R(Z):Var`: Shape of the predicate on the right `Z:FormVar`: Variable indicating where in P the substitution takes place
`rightSubstIffForall`	2	$\dfrac{\Gamma \vdash R(\phi(t)), \Delta \qquad \Sigma \vdash \forall x.\phi(x) \Leftrightarrow \psi(x), \Pi}{\Gamma, \Sigma \vdash R(\psi(t)), \Delta, \Pi}$	`i:Int`: Index of $\forall x.\phi(x) \Leftrightarrow \psi(x)$ on the right of the second premise `R(Z):Var`: Shape of the predicate on the right `Z:Var`: Variable indicating where in P the substitution takes place
`rightNnf`	1	$\dfrac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash \phi', \Delta}$	`i:Int`: Index of ϕ on the right of the premise `j:Int`: Index of ϕ' on the right of the conclusion ϕ and ϕ' have the same negation normal form
`rightPrenex`	1	$\dfrac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash \phi', \Delta}$	`i:Int`: Index of ϕ on the right of the premise `j:Int`: Index of ϕ' on the right of the conclusion ϕ and ϕ' have the same prenex normal form
`clausify`	0	$\dfrac{}{\Gamma, a \iff b \circ c \vdash \Delta}$	`i:Int`: Index of $a \iff b \circ c$ on the left Δ is a clause resulting from the inequality $a \iff b \circ c$
`elimIffRefl`	1	$\dfrac{\Gamma, \forall x_1, ..., x_n.\phi \iff \phi \vdash \Delta}{\Gamma \vdash \Delta}$	`i:Int`: Index of $\phi \iff \phi$ on the left of the premise
`res`	2	$\dfrac{\Gamma \vdash A, \Delta \qquad \Sigma \vdash \neg A, \Pi}{\Gamma, \Sigma \vdash \Delta, \Pi}$	`i:Int`: Index of A on the right of the first premise
`instMult`	1	$\dfrac{\Gamma[F_1, ..., F_n], \vdash \Delta[F_1, ..., F_n]}{\Gamma[G_1, ..., G_n] \vdash \Delta[G_1, ..., G_n]}$	Sequence of triplets of the form: `'F': String, t: Term|Formula, Xs: Seq[String]`. Each triplet has the same construction as arguments of `instFun` and `instPred`, but the substitution is carried simultaneously. Simultaneous substitution of function and predicate schemas, including variables and formula variables.

Table 5: Current Level 3 rules of SC-TPTP, which can be verified.

E Simulating Non-Deductive Proofs

Some common proof strategies used by automated theorem provers are not *deductive*, in the sense that they do not deduce a true conclusion from true premises but rather transform the entire problem. Typical such steps which are often performed by automated theorem provers are *proofs by contradiction*, *Tseitin's transform* and *Skolemization*. Because these strategies, when applied correctly, are sound, and because Sequent Calculus is complete, if a statement has a proof using non-deductive strategies, it has a proof in strict sequent calculus. But this proof may be (computationally) hard to find, and it may be much longer than a proof using it. Deskolemization in particular is known to sometimes blow up the size of the proofs [5,6]. In any case, for all three, it is not simply possible to define them as logical steps in SC-TPTP and unfold them locally. Eliminating them from an ATP proof require modifying the proof in its entirety.

Proof assistants typically have purely deductive systems, and their Kernel typically don't accept such steps. But they are necessary to transform a formula into clausal normal form, which many ATPs rely on. In this section we explain how proofs by contradiction, Tseitin's transform and Skolemization can be simulated deductively.

E.1 Proof by Contradiction with Backward and Forward Proofs

A common strategy to prove a conjecture ϕ is to assume $\neg\phi$ and deduce $\bot$. In TPTP proofs, this is typically denoted as:

```
fof(c1, conjecture, φ).
fof(nc1, negated_conjecture, ~φ).
...
fof(fn, plain, $false).
```

and we want to recover a proof of ϕ.

More generally, suppose a set of axioms $\phi_1, ..., \phi_n$ and conjecture ϕ are given. We may assume $\neg\phi$ and proceed forward:

$$\dfrac{\dfrac{\dfrac{\vdash \psi_1 \quad \ldots \quad \vdash \psi_n \quad \dfrac{}{\neg\phi \vdash \neg\phi}\,\texttt{hyp}}{\vdots}}{\neg\phi \vdash}}{\vdash \neg\neg\phi}\,\texttt{rightNot}}{\vdash \phi}\,\texttt{simp}$$

As a formula and its universal closure are interdeducible, we assume without loss of generality and for simplicity that ϕ has no free variable. $\vdots$ denotes the proof of $\bot$ from the axioms and $\neg\phi$. It can proceed entirely with the $\neg\phi$ formula present on the left, without effect on any proof step.

Example 2. **Conjecture** $\phi := \exists x.\forall y.\ p(x) \Rightarrow p(y)$
Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\phi \vdash \neg\exists x.\forall y.\ p(x) \Rightarrow p(y)}\,\texttt{hyp}}{\neg\phi \vdash \forall x.\exists y.\ p(x) \wedge \neg p(y)}\,\texttt{sameNNF}}{\neg\phi \vdash \exists y.\ p(x) \wedge \neg p(y)}\,\texttt{instForall} \qquad \dfrac{\dfrac{}{\neg\phi, p(x) \wedge \neg p(y) \vdash p(x) \wedge \neg p(y)}\,\texttt{hyp}}{\neg\phi, p(x) \wedge \neg p(y) \vdash}\,\texttt{simp}}{\neg\phi, \exists y.\ p(x) \wedge \neg p(y) \vdash}\,\texttt{leftExists}}{\dfrac{\dfrac{\neg\phi \vdash}{\vdash \neg\neg\phi}\,\texttt{rightNot}}{\vdash \phi}\,\texttt{simp}}\,\texttt{cut}$$

In Sequent Calculus, this kind of forward destructive proofs are somewhat cumbersome. All lvl 1 steps only introduce logical operators. To eliminate them one typically need a `hyp`, an introduction rule, and a `cut`. For example, the `instForall` unfolds into lvl 1 steps as follows:

$$\dfrac{\neg\phi \vdash \forall x.\exists y.\ p(x) \wedge \neg p(y) \qquad \dfrac{\dfrac{}{\exists y.\ p(x) \wedge \neg p(y) \vdash \exists y.\ p(x) \wedge \neg p(y)}\,\texttt{hyp}}{\forall x.\exists y.\ p(x) \wedge \neg p(y) \vdash \exists y.\ p(x) \wedge \neg p(y)}\,\texttt{leftForall}}{\neg\phi \vdash \exists y.\ p(x) \wedge \neg p(y)}\,\texttt{cut}$$

While such forward proofs work well for some proof system such as resolution, there is another strategy for the backward proofs, highlighting the "proof search" aspect of the procedure is simpler and more efficient for proof methods in non-clausal form, such as the tableaux method.

Backward proofs by contradiction To prove ϕ, it is sufficient to prove $\neg\phi \Rightarrow \bot$. In sequent calculus, this gives:

$$\dfrac{\dfrac{\dfrac{}{\phi \vdash \phi}\,\texttt{hyp}}{\vdash \neg\phi, \phi}\,\texttt{leftNot} \qquad \dfrac{\mathcal{A}\uparrow}{\neg\phi \vdash}}{\vdash \phi}\,\texttt{cut}$$

where $\mathcal{A}\uparrow$ is a proof of $\bot$ from $\neg\phi$, intuitively printed backward. When looked at from bottom to top, the left sequent calculus rules look like regular deduction. For example, the `leftAll` rule allow arbitrary instantiation, the `leftExists` rule force existential variables to be instantiated with fresh symbols and the `leftOr` rules branches. Every branch needs to be close by a proof of $\bot$, corresponding to the assumption $\bot \vdash$, for the proof to be correct.

Example 3. **Conjecture** $\exists x.\forall y.\ p(x) => p(y)$

Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p(x), \neg\ p(x) \vdash}}{p(x) \wedge \neg p(x) \vdash}\,\texttt{leftAnd}}{\exists y.\ p(x) \wedge \neg p(y) \vdash}\,\texttt{leftForall}}{\forall x.\exists y.\ p(x) \wedge \neg p(y) \vdash}\,\texttt{leftExists}}{\neg\exists x.\forall y.\ p(x) \Rightarrow p(y) \vdash}\,\texttt{sameNNF}}{\vdash \neg\neg\exists x.\forall y.\ p(x) \Rightarrow p(y)}\,\texttt{rightNot}}{\vdash \exists x.\forall y.\ p(x) \Rightarrow p(y)}\,\texttt{simp}$$

E.2 Clausification

Clausification is the transformation of a formula into a conjunction of disjunction of literals. There are two steps in this transformation; Skolemization and the Tseitin normal form. Both are not deductive steps; they return equisatisfiable formulas rather than logical consequences of the input. Tseitin transform can theoretically be replaced by a conjunctive normal form using distributivity, but at an exponential cost. For Skolemization, a proof of a skolemized formula can be up to non-elementarily smaller than a proof of the non-skolemized formula. Hence for practical use it is necessary to be able to represent these non-deductive operations efficiently.

Tseitin transform on axioms and negated conjecture Suppose we have a formula ϕ given as an axiom, and we aim to deduce $\bot$. The structure of the desired proof from the clausified axioms will be as follows:

$$\dfrac{\vdash a_1 \vee ... \vee a_n \qquad ... \qquad \vdash z_1 \vee ... \vee z_n}{\dfrac{\vdots}{\vdash \bot}}$$

Where $a_1 \vee ... \vee a_n$, ..., $z_1 \vee ... \vee z_n$ are the clauses resulting from the Tseitin transform of ϕ. But the clauses are not in general consequences of ϕ, as the transformation only preserves satisfiability.

Suppose ϕ contains $a \wedge b$ as subterm. We can simulate the first step of the Tseitin's transform as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{A \Leftrightarrow (a \wedge b) \vdash \psi}}{(a \wedge b) \Leftrightarrow (a \wedge b) \vdash \psi}\ 5.\ \texttt{instPred}}{\vdash \psi}\ 6.\ \texttt{elimIffRefl}}{\dfrac{\vdash \phi(a \wedge b)}{A \Leftrightarrow (a \wedge b) \vdash \phi(A)}\ 1.} \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash \neg A \vee a}\ 2. \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash \neg A \vee b}\ 3. \quad \dfrac{}{A \Leftrightarrow (a \wedge b) \vdash A \vee \neg a \vee \neg b}\ 4.}$$

Step 1 is `rightSubstIff`. Step 2, 3 and 4 are constant-size tautologies. Then the resolution proof follows normally, independently of the $A \iff (a \wedge b)$ on the left-hand side.

Finally, we eliminate the assumption. Step 5 allows instantiating the schematic formula A with an arbitrary formula[4]. To justify equisatisfiability, we need to ensure that A is fresh. Intuitively, this is enforced here by step 5. If A was not a fresh symbol but already appearing in ψ for example, the initial part of the proof would still work, but step 5 would fail.

Note that every single step can be unfolded into a constant number of level 1 steps, making the whole proof of Tseitin transformation linear. All steps are deductive and can be locally checked, independently of the rest of the proof.

[4] In higher-order logic, A would simply be a variable of type boolean.

Tseitin transform with negated-conjecure, propositional, backward The Tseitin transform allows to reduce deciding the satisfiability (or validity) of a formula to the satisfiability (or validity) of a different formula in CNF (or DNF). Let $\phi := \neg\psi$ be a conjectured formula we aim to show valid by showing that its negation is unsat. Hence we are trying to prove the sequent

$$\phi \vdash$$

Let ϕ' be the Tseitin transform of ϕ. We want to simulate the following step:

$$\dfrac{\dfrac{\vdots}{\phi' \vdash}}{\phi \vdash}\ \text{Tseitin}$$

For concreteness, let say that ϕ contains $a \wedge b$ as a subformula. In our approach, one step of the Tseitin transform is performed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\neg X \vee a), (\neg X \vee b), (X \vee \neg a \vee \neg b), \phi(X) \vdash}{X \Leftrightarrow (a \wedge b), \phi(X) \vdash}\ \text{4. Unfold } \Leftrightarrow \text{ into 3 clauses}}{X \Leftrightarrow (a \wedge b), \phi(a \wedge b) \vdash}\ \text{3. Substitution of equivalent formulas}}{(a \wedge b) \Leftrightarrow (a \wedge b), \phi(a \wedge b) \vdash}\ \text{2. Instantiate } X}{\phi(a \wedge b) \vdash}\ \text{1. Simplification}$$

Step 1 is a propositional tautology, easily represented with low level rules.

Prenex normal form, miniscoping and negation normal form Both Prenex Normal Form and its opposite miniscoping, as well as Negation Normal Form are typically equivalence-preserving and can easily be proven by repeated application of rewrite rules for each logical symbols:

$$\neg(\phi \vee \psi) \Leftrightarrow (\neg\phi \wedge \neg\psi)$$

$$\neg(\phi \wedge \psi) \Leftrightarrow (\neg\phi \vee \neg\psi)$$

$$\neg\neg\phi \Leftrightarrow \phi$$

$$\neg\exists x.\phi \Leftrightarrow \forall x.\neg\phi$$

$$\phi \vee \exists x.\psi \Leftrightarrow \exists x.\phi \vee \psi, \text{ with } x \text{ not free in } \psi$$

$$\phi \wedge \exists x.\psi \Leftrightarrow \exists x.\phi \wedge \psi, \text{ with } x \text{ not free in } \psi$$

Example 4.

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\vdash \exists x.\forall y.p(x) \Rightarrow \forall y.p(y)}}{\forall x.(\forall y.p(x) \Rightarrow p(y)) \Leftrightarrow (p(x) \Rightarrow \forall y.p(y)) \vdash \exists x.p(x) \Rightarrow \forall y.p(y)}\ \texttt{substIff} \qquad \dfrac{\dfrac{\vdash (\phi \Rightarrow \forall y.\psi(y)) \Leftrightarrow (\forall y.\phi \Rightarrow \psi(y))}{\vdash (p(x) \Rightarrow \forall y.p(y)) \Leftrightarrow (\forall y.p(x) \Rightarrow p(y))}\ \texttt{inst}}{\vdash \forall x.(p(x) \Rightarrow \forall y.p(y)) \Leftrightarrow (\forall y.p(x) \Rightarrow p(y))}\ \texttt{rightForall}}{\vdash \exists x.p(x) \Rightarrow \forall y.p(y)}\ \text{Cut}$$

Skolemization Consider a formula in prenex normal form with a single alternation of quantifiers:

$$\phi := \forall x.\exists y.\psi(x, y)$$

where ϕ is quantifier-free. Its skolemized form is the formula

$$\phi^{\mathsf{sko}} := \forall x.\psi(x, f(x))$$

with f a fresh function symbol. Then, $\phi \vdash \bot$ has a proof if and only if ϕ^{sko} has a proof. Similarly, the empty sequent has a proof from ϕ if and only if it has a proof from ϕ^{sko}. However, in general the proof of the original formula is complicated to compute and may in general be much larger than the proof of the original formula (exponentially or even non-elementary so). Moreover, the transformation is non-local: it does not simply unfold into low level proof steps, leaving the rest of the proof intact, but require transforming it as a whole.

We can formalize Skolemization while making it locally checkable and purely deductive, by introducing Hilbert's ε choice operator. ε is a term-level binder intuitively selecting an element satisfying a predicate, if such an element exist, and an arbitrary term otherwise. Its defining property is as follows:

$$\exists x.P(x) \Leftrightarrow P(\varepsilon x.P(x)) \ \texttt{substEpsilon}$$

When applied to ϕ, we obtain the following proof:

Example 5. **Axioms** $\vdash \forall x.\exists y.\psi(x, y)$
Conjecture $\vdash$
Proof

$\texttt{let}(f(x) := \varepsilon y.\psi(x, y))$

$$\dfrac{\dfrac{\dfrac{\mathcal{A}}{\forall x.\psi(x, f(x))}}{\forall x.\psi(x, \varepsilon y.\psi(x, y))}}{\forall x.\exists y.\psi(x, y)}\ \texttt{substEpsilon}$$

Observe that the subproof $\mathcal{A}$ can ignore the nature of ε and treat it as a black box: an uninterpreted function symbol containing the same free variables as the ε expression. Then $\mathcal{A}$ is exactly a proof of ϕ^{sko}.

F Example Proofs

Listing 1.1: Example proof from egg: Level 2

```
%--------------------------------------------------------------------
% Status   : Theorem
% SPC      : FOF_UNK_RFO_SEQ
% Solver   : egg v0.9.5
%          : egg-sc-tptp v0.1.0
```

```
% Logic    : schem
%------------------------------------------------------------------------
fof(div_one, axiom, ! [X]: d(X, t1) = X).
fof(cancel_den, axiom, ! [X, Y]: (m(d(X, Y), Y) = X)).
fof(invert_div, axiom, ! [X, Y]: d(X, Y) = d(t1, d(Y, X))).
fof(c, conjecture, d(m(d(t2, t3), d(t3, t2)), t1) = t1).


fof(f0, plain, [] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = d(m(d(t2, t3), d(t3, t2)), t1)],
    inference(rightRefl, [ ... ], [])).
fof(f1, plain, [] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t2, t3), d(t3, t2))],
    inference(rightSubstEqForall, [ ... ], [div_one, f0])).
fof(f2, plain, [] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t1, d(t3, t2)), d(t3, t2))],
    inference(rightSubstEqForall, [ ... ], [invert_div, f1])).
fof(f3, plain, [] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = t1],
    inference(rightSubstEqForall, [ ... ], [cancel_den, f2])).
```

Listing 1.2: egg proof from the same problem: Level 1

```
fof(div_one, axiom, ! [X]: d(X, t1) = X).
fof(cancel_den, axiom, ! [X, Y]: (m(d(X, Y), Y) = X)).
fof(invert_div, axiom, ! [X, Y]: d(X, Y) = d(t1, d(Y, X))).
fof(c, conjecture, d(m(d(t2, t3), d(t3, t2)), t1) = t1).

fof(f0, plain, [] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = d(m(d(t2, t3), d(t3, t2)), t1)],
    inference(rightRefl, [...], [])).
fof(f1, plain, [
    d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t2, t3), d(t3, t2))] --> [
    d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t2, t3), d(t3, t2))],
    inference(rightSubst, [...], [f0])).
fof(f2, plain, [![X] : d(X, t1) = X] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t2, t3), d(t3, t2))],
    inference(leftForall, [...], [f1])).
fof(f3, plain, [] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t2, t3), d(t3, t2))],
    inference(cut, [...], [div_one, f2])).
fof(f4, plain, [d(t2, t3) = d(t1, d(t3, t2))] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t1, d(t3, t2)), d(t3, t2))],
    inference(rightSubst, [...], [f3])).
fof(f5, plain, [![Y] : d(t2, Y) = d(t1, d(Y, t2))] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t1, d(t3, t2)), d(t3, t2))],
    inference(leftForall, [...], [f4])).
fof(f6, plain, [![X, Y] : d(X, Y) = d(t1, d(Y, X))] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t1, d(t3, t2)), d(t3, t2))],
    inference(leftForall, [...], [f5])).
fof(f7, plain, [] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = m(d(t1, d(t3, t2)), d(t3, t2))],
    inference(cut, [...], [invert_div, f6])).
fof(f8, plain, [m(d(t1, d(t3, t2)), d(t3, t2)) = t1] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = t1],
    inference(rightSubst, [...], [f7])).
fof(f9, plain, [![Y] : m(d(t1, Y), Y) = t1] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = t1],
    inference(leftForall, [...], [f8])).
fof(f10, plain, [![X, Y] : m(d(X, Y), Y) = X] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = t1],
    inference(leftForall, [...], [f9])).
fof(f11, plain, [] -->
    [d(m(d(t2, t3), d(t3, t2)), t1) = t1],
    inference(cut, [...], [cancel_denominator, f10])).
```

Listing 1.3: Example proof from goeland

```
% SZS output start Proof for lisa.maths.Tests.buveurs.p


fof(drinkers, conjecture, (? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))))
    .

fof(f9, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6))))), ~((d(X4_8) ⇒ d(Sko_0))), d(X4_8),
     ~(d(Sko_0)), ~((! [Y6] : ((d(Sko_0) ⇒ d(Y6))))), ~((d(Sko_0)
    ⇒ d(Sko_1))), d(Sko_0), ~(d(Sko_1))] ⟶ [], inference(leftHyp,
     [status(thm), 4], [])).

fof(f8, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6))))), ~((d(X4_8) ⇒ d(Sko_0))), d(X4_8),
     ~(d(Sko_0)), ~((! [Y6] : ((d(Sko_0) ⇒ d(Y6))))), ~((d(Sko_0)
    ⇒ d(Sko_1)))] ⟶ [], inference(leftNotImplies, [status(thm),
    6], [f9])).

fof(f7, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6))))), ~((d(X4_8) ⇒ d(Sko_0))), d(X4_8),
     ~(d(Sko_0)), ~((! [Y6] : ((d(Sko_0) ⇒ d(Y6)))))] ⟶ [],
    inference(leftNotAll, [status(thm), 5, 'Sko_1'], [f8])).

fof(f6, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6))))), ~((d(X4_8) ⇒ d(Sko_0))), d(X4_8),
     ~(d(Sko_0))] ⟶ [], inference(leftNotEx, [status(thm), 0, $fot(
    Sko_0)], [f7])).

fof(f5, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6))))), ~((d(X4_8) ⇒ d(Sko_0)))] ⟶ [],
    inference(leftNotImplies, [status(thm), 2], [f6])).

fof(f4, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))), ~((! [
    Y6] : ((d(X4_8) ⇒ d(Y6)))))] ⟶ [], inference(leftNotAll, [
    status(thm), 1, 'Sko_0'], [f5])).

fof(f3, plain, [~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))))] ⟶
    [], inference(leftNotEx, [status(thm), 0, $fot(X4_8)], [f4])).

fof(f2, plain, [(? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6))))))] ⟶ [(?
    [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6))))))], inference(hyp, [status(
    thm), 0], [])).
```

```
fof(f1, plain, [] ⟶ [(? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))),
    ~((? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6))))))], inference(
    rightNot, [status(thm), 1], [f2])).

fof(f0, plain, [] ⟶ [(? [X4] : ((! [Y6] : ((d(X4) ⇒ d(Y6)))))],
    inference(cut, [status(thm), 1], [f1, f3])).

% SZS output end Proof for lisa.maths.Tests.buveurs.p
```