SC-TPTP: An Extension of the TPTP Derivation Format for Sequent-Based Calculus

Julie Cailler[1], Simon Guilloud[2]

[1]University of Regensburg, Regensburg, Germany
[2]EPFL, Lausanne, Switzerland

Abstract

Motivated by the transfer of proofs between proof systems, and in particular from first order *automated theorem provers* (ATPs) to *interactive theorem provers* (ITPs), we specify an extension of the TPTP derivation [1] text format to describe proofs in first-order logic: SC-TPTP. To avoid multiplication of standards, our proposed format over-specifies the TPTP derivation format by focusing on sequent formalisms. By doing so, it provides a high level of detail, is faithful to mathematical tradition, and cover multiple existing tools and in particular tableaux-based strategies. We make use of this format to allow the Lisa proof assistant [2] to query the Goéland automated theorem prover [3], and implement a library of tools able to parse, print and check SC-TPTP proofs, export them into Coq files, and rebuild low-level proof steps from advanced ones.

Keywords

TPTP, Proof Format, Automated Theorem Proving, Interactive Theorem Proving

1. Introduction

Transfer of proofs between different proof systems to this day largely remains a challenge. While the relative consistency strength of the foundations of most tools is well known, practical translations presents a variety of technical difficulties. Indeed, not only the syntax of different systems can be very hard (when not impossible) to simulate (for example, λ-abstractions used in type theory-based systems are difficult to represent and reason about in first-order systems, or an classical ATP and an intuitionistic ITP), but even proof systems with similar logical foundations can diverge significantly in terms of the kind and granularity of accepted *proof steps*. In practice, an ATP might use advanced proof steps such as Skolemization, superposition, hyperresolution, or congruence closure, which can be difficult (in terms of implementation and space-time complexity) to simulate with lower-level proof steps typically accepted by proof assistants. Finally, different tools may use different formats to export and store statements and proofs in the first place (e.g. TPTP [1], XML [4], LFSC [5], Lambdapi [6] and other (tool-specific) formats), each requiring dedicated parsers and abstract syntax trees.

Hence, even when the foundations and proof steps are similar, if n systems want to import proofs directly from each other, each is required to implement $n-1$ different import and proof transform algorithms, for a total of $n(n-1)$ implementations. With one common middle ground format, each system only needs to implement one import and one export algorithm from itself to the proof format, for a total of $2n$ implementations. On the other hand, as suggested above,

PAAR'24: 9th Workshop on Practical Aspects of Automated Reasoning, July 2, 2024, Nancy, France

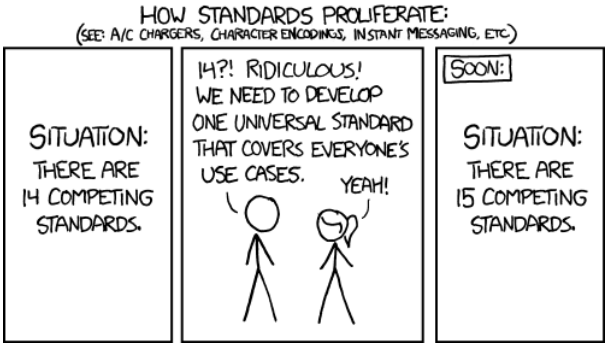

Figure 1: Standards (xkcd — https://xkcd.com/927/). Licensed under CC BY-NC 2.5 License (https://creativecommons.org/licenses/by-nc/2.5/)

unifying the syntax and proofs of arbitrarily many systems with unrelated foundations is overly ambitious and may not be practically feasible. A successful approach needs to convey the right level of abstraction, neither too specific nor too general.

This introduction may remind the reader of the famous xkcd comics "How Standards Proliferate" (Figure 1). Since being counterproductive runs against our objective, we design a proof format as a super specification of the TPTP derivation format, which is already supported by a large collection of systems and tools. TPTP (Thousands of Problems for Theorem Provers) [7] is a large library of problems for testing automated theorem provers. It is also a specification format [1] for said problems and *derivations*, i.e., lists of formulas derived from the specification of the problem or from previously deduced ones. However, there is no standard specification of *how* formulas are derived.

Our motivation for the present proposal is to specifically increase interoperability between tools that use first-order logic. A key goal is to allow proof systems (in particular ITPs) to query ATPs on problems, and obtain proofs in a single format in return. Hence, the present work defines a specification over the existing TPTP format for sequent-calculus style proofs in first-order logic. We fix and specify a set of basic inference steps, as well as their parameters, semantics, and syntax, in the style of sequent calculus. We separate deduction steps in *levels*, from low-level, easy-to-verify standard steps of sequent calculus, to more complex and tool-specific steps, which ideally come with a procedure to transform them into low-level steps. We call the resulting standard SC-TPTP. By keeping as-is TPTP's directives for everything else, including formulas, existing TPTP parsers and printers can be readily used. Our objectives are:

- to allow proof systems (in particular ITPs) using first-order logic to query ATPs for first-order logic (in particular tableaux-based ATPs) on problems, and obtain proofs in a single format in return;
- to provide a single translation exports from multiple systems with sequent calculus-like foundations to another proof system or format (for example Coq);
- to offer transformation algorithms that simplify proofs with possibly complicated and higher level proof steps to proofs with only basic steps;
- to allow answers of ATPs in competitions to be verified unambiguously (for problems in first-order logic).

Contributions Our contributions in the present text are as follows:

- The development of a standard format, SC-TPTP, for sequent-based calculus, with a focus on tableaux-based ATP.
- The implementation of this format into the Lisa proof assistant (import), and the Goéland ATP (export). In particular, this allows Lisa users to call Goéland as a (proof-producing) tactic.
- The development of a publicly available library, providing tools to parse and print SC-TPTP proofs, check their validity in this format, transform proofs to eliminate high-level congruence closure steps, and export them into Coq.

This work is a proof of concept and a proposal to the community, that we hope can help connect various tools. We look forward to the community's input and feedback on how to adapt the features and design choices to be suitable for as many tools as possible.

Related Work Beyond the TPTP format itself [1] and its associated body of work, significant research efforts have been directed toward designing proof formats conducive to tool interoperability. Those efforts started by conducting various investigations in order to delineate the criteria for an ideal proof format, emphasizing the importance of ease of parsing while retaining human readability [8, 9].

One of the original inspirations for designing a common format comes from the SAT community, which encountered notable success with the DRAT format [10]. Work has also been done to provide a proof checker for this format [11].

In the realm of SMT solving, diverse proof formats have emerged, including LFSC [12] used by CVC5 [13] or the Z3 [14] proof format [15], among others. While there is currently no universal proof format for SMT solvers, efforts are made toward this goal [16, 12, 17], resulting for instance in the Alethe [18] proof format, employed by the VeriT solver [19].

Closer to our approach, and extension of the TPTP syntax to the connection calculus [20] was implemented into the leanCoP [21] and Connect++ [22] provers. In the meantime, the Theory Extensible Sequent Calculus (TESC) format for First-Order ATPs [23] offers a sequent-based proof format capable of compiling and verifying solutions from Vampire [24] and E [25] by combining TPTP problems with their respective TSTP solutions. Within the TPTP ecosystem, the GDV Verifier [26] proof checker for CNF derivation in the TPTP format warrant mention.

Finally, our paper's overarching theme resides in the realm of proof interoperability, echoing the ethos of systems such as Dedukti and Lambdapi [6].

Organization In Section 2, we provide the necessary background on TPTP, tableaux rules, as well as one-sided and two-sided sequent calculi. We then introduce the proof steps of the SC-TPTP proof standard for (untyped) first order-logic in Section 3, together with their parameters, syntax, and semantic. In Section 4, we present a library of utilities to handle and verify proofs in SC-TPTP. In particular, we implement an export of SC-TPTP proofs to Coq, and a proof-producing egraph [27] to produce detailed proofs corresponding to congruence closure steps. Finally, we describe Section 5, as a use case and proof of concept, how we made the Goéland ATP and the Lisa ITP support SC-TPTP, allowing for direct transfer of proofs from the first to the second.

2. Context

2.1. TPTP & TSTP

TPTP [7] is the reference problem library in the field of automated reasoning. It is made of over 25000 problems (including over 7000 first-order (`FOF` category) problems), ranging from easy to open, that can be used for testing and evaluating ATPs. This library also provides standards for input and output for ATP systems, thanks to the TPTP logic language [1], that is used to specify logical decision problems in various logic ((typed) first-order, higher order, each with and without polymorphism, CNFs, ...). It is a well-adopted input format and benchmark, used for example in CASC, the CADE ATP System Competition.

Sibling to TPTP is the TSTP library, a collection of solutions (derivations, or proofs) produced by ATPs in response to TPTP problems. In derivations, formulas are annotated by an *inference* parameter, itself made of the name of the inference rule, its parameter, and its premises. However, inferences are not specified: every system can output its individual proof steps, which can be arbitrarily complex to reconstruct for a system with different or more basic deduction rules. As an extreme example, an ATP may output a proof step justified with "SMT solver said so", from which it would be very difficult to recover a posteriori a syntactically strict proof, such as a sequent calculus or natural deduction proof or a well-typed proof term. Even when systems output reasonable, low-level steps, there can be many inessential differences in the exact set of rules and their syntax.

2.2. Sequent-Based Calculus

Sequent Calculus is a proof system for (classical) first-order logic (with equality), where statements are represented by sequents. A sequent is a pair of sets of formulas, typically written

$$a_1, ..., a_n \vdash b_1, ..., b_n$$

whose intended semantics is $(a_1 \wedge ... \wedge a_n) \implies (b_1 \vee ... \vee b_n)$.

LK & LJ The original sequent calculus LK [28] for classical logic, and its counterpart LJ [29] for intuitionistic logic, are formals system designed to study natural deduction in first-order logic. Sequent calculus admits a *subformula property*, which gives it its high theoretical and practical relevance. In addition, this calculus also provides a naive (but very inefficient in practice) complete proof search procedure, on which are built more refined proof-search strategies such as the method of analytics tableaux.

Tableaux The tableaux method is a semi-decision procedure for first-order logic. A variety of ATPs are based on the (free-variable) tableaux method, such as Princess [30], Goéland [3], Zenon [31], ZenonModulo [32], LeanTAP [33] and others. One of the advantages of this method is that it follows generally the proof steps of sequent calculus, making it suitable for the reconstruction of efficiently and independently verifiable proofs. The tableaux calculus consists of a set of (refutationally complete) inference rules, divided in four categories: α rules (unary inferences), β rules (binary inferences), γ rules (instantiation rules) and δ rules (Skolemization rules).

GS3 The Gentzen-Schütte calculus (GS3) [34] is a one-sided variant of sequent calculus where formulas are only allowed on the left side. The deduction rules are similar to those of the usual sequent calculus, but the right rules are replaced by left-negation rules. Except for equality, it is easy to observe that the three proof systems are equivalent, with GS3 as a middle ground. For equality reasoning, however, expressing the global closure substitution step using the equality substitution rules is non-trivial, and will be the topic of Section 4.

3. SC-TPTP: Technical Description

We introduce the SC-TPTP format as an over-specification of the TPTP format [1] for derivations, with two level of derivation steps. A derivation is a list of *annotated formulas*, which can be conjectures or axioms as in the specification of TPTP problems, or derived from previously annotated formulas via some inference rule and parameters. Formulas themselves are part of the FOFX grammar of TPTP (Figure 2), which extends FOF by supporting notation for sequents. Until now, the FOFX format was defined but "not yet in use". Technically, TPTP does not allow free variables. However, they are an integral part of sequent calculus[1], and the grammar supports free variables.

```
        <fof_annotated> ::=  fof(<name>, <formula_role>, <fof_formula>, <annotations>).
         <formula_role> ::=  assumption | axiom | conjecture | plain
          <fof_formula> ::=  <fof_logical_formula> | <fof_sequent>
          <fof_sequent> ::=  <fof_formula_tuple> -> <fof_formula_tuple>
                        |    (<fof_sequent>)
    <fof_formula_tuple> ::=  [] | [<fof_formula_tuple_list>]
<fof_formula_tuple_list> ::=  <fof_logic_formula>
                        |    <fof_logic_formula>, <fof_formula_tuple_list>
```

Figure 2: Main elements of the SC-TPTP syntax. `logic_formula` is the FOF syntax for logical formulas. For SC-TPTP derivations, annotations are of the form `inference(rule, [parameters], [premises])`.

In the original presentation of Gentzen, sequents' sides are formally *lists* of formulas, where order and number of duplicates are significant. This semantics requires additional structural rules for contraction and permutation of formulas. However, it is more convenient and efficient (in terms of proof size and number of rules) to consider them as sets. This is the semantic we chose for SC-TPTP. In particular, formulas in sequents can be duplicated, contracted, and reordered at will. This does not make proof checking more complex.

[1] and in general of deductive reasoning, for example in the proof system of HOL Light [35] and in informal mathematics

Rule name	Premises	Rule	Parameters
`hyp`	0	$\dfrac{}{\Gamma, A \vdash A, \Delta}$	`i:Int`: Index of A on the left `j:Int`: Index of A on the right
`leftHyp`	0	$\dfrac{}{\Gamma, A, \neg A \vdash \Delta}$	`i:Int`: Index of A on the left `j:Int`: Index of $\neg A$ on the left.
`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 the cut formula on the right of the first premise

Table 1
Level 1 rules of SC-TPTP, for one-sided and two-sided sequent calculus — structural rules.

Rule name	Premises	Rule	Parameters
`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
`leftImp1`	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
`leftImp2`	2	$\dfrac{\Gamma, \neg A \vdash \Delta \qquad \Sigma, B \vdash \Pi}{\Gamma, \Sigma, A \Rightarrow B \vdash \Delta, \Pi}$	`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
`leftEx`	1	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}$	`i:Int`: Index of $\exists x.A$ on the left `y:Var`: Variable in place of x in the premise
`leftAll`	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

Table 2
Level 1 rules of SC-TPTP, for one-sided and two-sided sequent calculus — left introduction rules.

3.1. Derivations

Each step of the derivation is written as an *annotated statement* of the form `fof(`*`name`*`,`*`role`*`,`*`statement`*`,`*`annotation`*`)`, in which:

- *`name`* is an integer or an alphanumeric identifier starting with a lowercase letter. It is used to be referred to by other steps of the derivation.
- *`role`* is either “axiom”, “conjecture” or “plain”. An “axiom” denotes an accepted formula, while a “conjecture” holds for the statement the derivation is supposed to prove, but does not play a logical role. A conjecture should always contain the same formula as the last derived formula in the proof. The TPTP syntax does not impose specific user semantics

Rule name	Premises	Rule	Parameters
`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	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}$	`i:Int`: Index of $A \vee B$ on the right
`rightImp`	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	$\dfrac{\Gamma \vdash A \Rightarrow B, \Delta \qquad \Sigma \vdash B \Rightarrow A, \Pi}{\Gamma, \Sigma \vdash A \Leftrightarrow B, \Delta, \Pi}$	`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
`rightEx`	1	$\dfrac{\Gamma, A[x := t] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}$	`i:Int`: Index of $\exists x.A$ on the right `t:Term`: Term in in place of x in the premise
`rightAll`	1	$\dfrac{\Gamma, A[x := y] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}$	`i:Int`: Index of $\forall x.A$ on the right `y:Var`: Variable in place of x in the premise

Table 3
Level 1 rules of SC-TPTP, for one-sided and two-sided sequent calculus — right introduction rules.

to the "plain" role, and thus we use it to denote inferred steps.

- *`statement`* can be either of a sequent or a formula. Their syntax can be found in https://tptp.org/TPTP/SyntaxBNF.html. For SC-TPTP, a *sequent* statement is made of two sets of formulas, while a *formula* statement is understood as standing for the sequent with an empty left-hand side and whose right-hand side contains exactly this formula.
- *`annotation`* are used to give additional information to the system. If the role of the statement is "axiom" or "conjecture", the annotation has no specific requirement in our format. For "assumption" and "plain" statements, the *`annotation`* must be of the form `inference(stepName, [status(thm), p`$_1$`, ..., p`$_n$`], [r`$_1$`, ..., r`$_n$`])`, in which:
  - *`stepName`* is one of the entry listed in Table 1-Table 6.
  - `status(thm)` indicates the status of the formula (e.g., a consequence of the premise, equisatisfiable with regard to the previous step, a negated conjecture, etc.) following the *SZS ontologies* [36]. In SC-TPTP, all steps are deductive inference and hence we only use the `thm` status.
  - The elements `p`$_i$'s within the parameter list vary in number and shape based on the proof step (indexes, variables, (first-order) terms (`$fot`), etc). They are described in the 4$^{\text{th}}$ column of Table 1-Table 6 and typically indicate how the step is constructed, thus making its correctness easily verifiable, without requiring inference.
  - The elements `r`$_i$'s in the premises list point to the premises of the deduction step. Their number varies between 0 and 2, depending on the proof step, and are referenced in the 2$^{\text{nd}}$ column of Table 1-Table 6.

Example 3.1. The following example illustrates a valid SC-TPTP derivation.

```
fof(c, conjecture, [![X]: P(X)] --> [P(A) & P(B)]).
fof(s1, assumption, [P(A)] --> [P(A)], inference(hyp, [status(thm), 0, 0], [])).
fof(s2, plain, [![X]: P(X)] --> [P(A)],
inference(leftAll, [status(thm), 0, $fot(A)], [s1])).
fof(s3, assumption, [P(B)] --> [P(B)], inference(hyp, [status(thm), 0, 0], [])).
fof(s4,plain, [![X]: P(X)] --> [P(B)],
inference(leftAll, [status(thm), 0, $fot(B)], [s3])).
fof(s5,plain, [![X]: P(X)] --> [P(A) & P(B)],
inference(rightAnd, [status(thm), 0], [s2, s4])).
```

Rule name	Premises	Rule	Parameters
`leftNotAnd`	2	$\frac{\Gamma, \neg A \vdash \Delta \qquad \Sigma, \neg B \vdash \Pi}{\Gamma, \Sigma, \neg(A \wedge B) \vdash \Delta, \Pi}$	`i:Int`: Index of $\neg(A \wedge B)$ on the left
`leftNotOr`	1	$\frac{\Gamma, \neg A, \neg B \vdash \Delta}{\Gamma, \neg(A \vee B) \vdash \Delta}$	`i:Int`: Index of $\neg(A \vee B)$ on the left
`leftNotImp`	1	$\frac{\Gamma, A, \neg B \vdash \Delta}{\Gamma, \neg(A \Rightarrow B) \vdash \Delta}$	`i:Int`: Index of $\neg(A \Rightarrow B)$ on the left
`leftNotIff`	2	$\frac{\Gamma, \neg(A \Rightarrow B) \vdash \Delta \qquad \Sigma, \neg(B \Rightarrow A) \vdash \Pi}{\Gamma, \Sigma, \neg(A \Leftrightarrow B) \vdash \Delta, \Pi}$	`i:Int`: Index of $\neg(A \Leftrightarrow B)$ on the left
`leftNotNot`	1	$\frac{\Gamma, A \vdash \Delta}{\Gamma, \neg\neg A \vdash \Delta}$	`i:Int`: Index of $\neg\neg A$ on the left
`leftNotEx`	1	$\frac{\Gamma, \neg A[x := t] \vdash \Delta}{\Gamma, \neg\exists x.A \vdash \Delta}$	`i:Int`: Index of $\neg\exists x.A$ on the right `t:Term`: Term in place of x in the premise
`leftNotAll`	1	$\frac{\Gamma, \neg A \vdash \Delta}{\Gamma, \neg\forall x.A \vdash \Delta}$	`i:Int`: Index of $\neg\forall x.A$ on the right `y:Var`: Variable in place of x in the premise

Table 4
Level 1 rules of SC-TPTP, for one-sided and two-sided sequent calculus — left not introduction rules. These rules are equivalent to the right rules of Table 3, but more directly match tableaux reasonning.

Rule name	Premises	Rule	Parameters
`rightRefl`	0	$\frac{}{\Gamma \vdash t = t, \Delta}$	`i:Int`: Index of $t = t$ on the right.
`rightSubst`	1	$\frac{\Gamma, t = u \vdash P(t), \Delta}{\Gamma, t = u \vdash P(u), \Delta}$	`i:Int`: Index of $t = u$ on the left `P(Z):Var`: Shape of the predicate on the right `Z:Var`: unifiable sub-term in the predicate
`leftSubst`	1	$\frac{\Gamma, t = u, P(t) \vdash \Delta}{\Gamma, t = u, P(u) \vdash \Delta}$	`i:Int`: Index of $t = u$ on the left `P(Z):Term`: Shape of the predicate on the left `Z:Var`: variable indicating where to substitute

Table 5
Level 1 rules of SC-TPTP, for one-sided and two-sided sequent calculus — equality reasoning.

Example 3.2. SC-TPTP proof of the *drinker paradox* [37].

```
fof(c_drinkers_p, conjecture, (? [X] : d(X) => (! [Y] : d(Y)))).

fof(f8, assumption, [~(? [X] : d(X) => (! [Y] : d(Y))),
~(d(X_0) => (! [Y] : d(Y))), d(X_0), ~(! [Y] : d(Y)), ~d(Sko_0),
~(d(Sko_0) => (! [Y] : d(Y))), d(Sko_0)] --> [],
inference(leftHyp, [status(thm), 6, 4], [])).

fof(f7, plain, [~(? [X] : d(X) => (! [Y] : d(Y))),
~(d(X_0) => (! [Y] : d(Y))), d(X_0), ~(! [Y] : d(Y)), ~d(Sko_0),
~(d(Sko_0) => (! [Y] : d(Y)))] --> [], inference(leftNotImp, [status(thm), 5], [f8])).

fof(f6, plain, [~(? [X] : d(X) => (! [Y] : d(Y))),
~(d(X_0) => (! [Y] : d(Y))), d(X_0), ~(! [Y] : d(Y)), ~d(Sko_0)]
--> [], inference(leftNotEx, [status(thm), 0, $fot(Sko_0)], [f7])).

fof(f5, plain, [ ~(? [X] : d(X) => (! [Y] : d(Y))),
~(d(X_0) => (! [Y] : d(Y))), d(X_0), ~(! [Y] : d(Y))] --> [],
inference(leftNotForall, [status(thm), 3, $fot(Sko_0)], [f6])).

fof(f4, plain, [~(? [X] : d(X) => (! [Y] : d(Y))),
~(d(X_0) => (! [Y] : d(Y)))] --> [], inference(leftNotImp, [status(thm), 1], [f5])).

fof(f3, plain, [~(? [X] : d(X) => (! [Y] : d(Y)))] --> [],
inference(leftNotEx, [status(thm), 0, $fot(X_0)], [f4])).

fof(f2, assumption, [(? [X] : d(X) => (! [Y] : d(Y)))] -->
[(? [X] : d(X) => (! [Y] : d(Y)))], inference(hyp, [status(thm), 0, 0], [])).

fof(f1, plain, [] --> [(? [X] : d(X) => (! [Y] : d(Y))),
~(? [X] : d(X) => (! [Y] : d(Y)))], inference(rightNot, [status(thm), 1], [f2])).

fof(f0, plain, [] --> [(? [X] : d(X) => (! [Y] : d(Y)))],
inference(cut, [status(thm), 0], [f1, f3])).
```

3.2. Level 1 Deduction Steps

We define three *levels* for deduction steps. Level 1 steps are exactly the 30 steps represented in Table 1 to Table 5. In this setup, "on the left/right" refers to the left and the right of the conclusion. Those rules are complete for first order logic with equality, and their correctness is simple to check. As they represents low-level proof steps they should also be straightforward to import into any proof system strong enough to accommodate first order logic. They encompass both the traditional two-sided sequent calculus rules, with antecedents and succedants, as well

as the rules for one-sided sequent calculus typically used for tableaux theorem proving. As such, the system is not minimal.

Table 1 presents the first group of rules (`hyp`, `leftHyp`, `leftWeaken`, `rightWeaken`, `cut`), also called *structural rules*. The next group of 8 rules introduced in Table 2 is composed by *left introduction rules*, all of them describing a specific way a symbol can be introduced. The subsequent group in Table 3 encompass *right introduction rules*, dual to the left rules. Then follow *left not introduction rules* in Table 4, which are essentially equivalent to the right introduction rules, but useful for tableaux-style proofs. Finally, `rightRefl`, `leftSubst` and `rightSubst` of Table 5 support equality reasoning.

Parameters and steps correctness

In an SC-TPTP derivation, inferences with level 1 steps come with a set of parameters that makes verification straightforward and more efficient, without requiring inference. For most proof steps, the parameters are only indexes, pointing to the position of the formula targeted by the proof step. For example, consider a derivation with the `rightAnd` rule:

```
fof(ax1, axiom, [] --> [a])
fof(ax2, axiom, [] --> [b])
fof(s1, plain, [] --> [a & b], inference(rightAnd, [status(thm), 0], [ax1, ax2]))
```

The step `s1` is inferred using the rule `rightAnd`. The index 0 in `[0]` indicates that the deduced conjunction is the first formula (on the right-hand side of the conclusion sequent), that is `a & b`. This then indicates that `a` is a formula on the right of the first premise (`ax1`) and `b` a formula on the right of the second premise (`ax2`). In the general case, let $\Gamma_1 \vdash \Delta_1$, $\Gamma_2 \vdash \Delta_2$ and $\Gamma_3 \vdash \Delta_3$ be the sequents of `ax1`, `ax2` and `s1`. To verify that the step is correctly applied, we simply have to check the following conditions:

$$\Gamma_3 == \Gamma_1 \cup \Gamma_2$$
$$\{\texttt{a}\} \cup \Delta_3 == \{\texttt{a \& b}\} \cup \Delta_1$$
$$\{\texttt{b}\} \cup \Delta_3 == \{\texttt{a \& b}\} \cup \Delta_2$$

Where $==$ is equality on sets. Note that the Δ's may contain additional occurrences of `a`, `b` and `a & b`, in which case the step is still valid. All `left`, `right` and `leftNot` propositional steps, as well as `rightRefl` and all structural steps but `cut` work the same way. Note that using hash sets, these tests can be done in time linear in the size of the sequents.

`Ex` and `All` steps take an additional argument, denoting which subterm or variable is being quantified. Consider:

```
fof(s1,plain, [] --> [(f(X) = f(X))],
      inference(rightRefl, [status(thm), 0], []))

fof(s2,plain, [] --> [?[Y]: (f(X) = Y)],
      inference(rightEx, [status(thm), 0, $fot(f(X))], [s1]))

fof(s3,plain, [] --> [![X]: (?[Y]: (f(X) = Y))],
      inference(rightAll, [status(thm), 0, $fot(X)], [s1]))
```

For `s2`, the parameter 0 indicates that the first formula in the right of the conclusion has been quantified, which is `?[Y]: (f(X) = Y)`. Again in a more general case with arbitrary contexts, let $\Gamma_1 \vdash \Delta_1$, $\Gamma_2 \vdash \Delta_2$ and $\Gamma_3 \vdash \Delta_3$ be the sequents of `s1`, `s2` and `s3`. Now, to check the correctness of `s2`, we must verify:

$$\Gamma_1 == \Gamma_2$$

$$\{\texttt{(f(X) = Y)}[\texttt{Y} := \texttt{f(X)}]\} \cup \Delta_2 == \{\texttt{?[Y]: (f(X) = Y)}\} \cup \Delta_1$$

Where $\phi[X := t]$ denotes the (capture-avoiding) substitution of X by t in ϕ. Note that the equality has to be performed modulo alpha-equivalence. This can be done naively in time $\mathcal{O}(n^2)$, but also efficiently either by using some hash function that is congruent with respect to alpha-equivalence, such as in [38], or more simply by computing a locally nameless normal form for formulas.

For step s3, the same checks need to be done, but additionally, it must be verified that the quantified variable X is not free in the resulting sequent.

The `Cut` step is slightly different: because its main formula does not appear in the conclusion, the index indicates instead the position cut formula in the right-hand side of the first premise (this is arbitrary; we could have pointed instead to the cut formula in the left-hand side of the second premise).

Finally, the `leftSubst` and `rightSubst` rules are a bit more complex. Consider for example the following derivation:

```
fof(a1, axiom, [P(f(a))] --> [])

fof(s1,plain, [P(g(b)), (f(a) = g(b))] --> [],
      inference(leftSubst, [status(thm), 1, $fof(P(Z)), $fot(Z)), [s1]))
```

The parameter 1 points to the equality `f(a) = g(b)`. The second and third parameters explain how the substitution is carried, so for general sequents Δ_1 and Δ_2 as above, the check is:

$$\{\texttt{P(Z)}[\texttt{Z:=g(b)}], \texttt{f(a) = g(b)}\} \cup \Delta_1 == \{\texttt{P(Z)}[\texttt{Z:=g(a)}]\} \cup \Delta_2$$

3.3. Level 2 Deduction Steps

Unlike in level 1, level 2 steps are not entirely fixed and are expected to expand over time. They contain more advanced proof steps, which may be more difficult to verify, but for which there should be an available and implemented algorithm eliminating them from a proof. This mechanism allows Level 1 proofs to be rebuildable from a Level 2 proof. We expect that proofs relying on level 2 proof steps will be common in practice: each tool will keep those they accept natively (or can import easily), and eliminate steps they do not support. At present time, we have implemented three level 2 steps, which were useful to our implementation: left and right simultaneous substitution of equal terms, and congruence closure. These and their parameters are shown in Table 6. Another example of a level 2 candidate (not implemented) is an `NNF` step, which allows deduce a sequent from a premise whose formulas are equivalent, modulo negation normal form.

Simultaneous substitutions are fairly simple. A simultaneous substitution of n formulas can always be unfolded into n application of `leftSubst` or `rightSubst`. Congruence closure is

Rule name	Premises	Rule	Parameters
`NNF`	1	$\frac{\Gamma \vdash \Delta}{\Gamma' \vdash \Delta'}$	No parameters The premise and conclusion are equal up to negation normal form
`congruence`	0	$\frac{}{\Gamma, P(u) \vdash P(t), \Delta}$	No parameter Γ contains a set of equalities such that t and u are congruent (actually more general, see bellow)
`rightSubstMulti`	1	$\frac{\Gamma \vdash P(t_1, ..., t_n), \Delta}{\Gamma \vdash P(u_1, ..., u_n), \Delta}$	[`i`$_1$`, ..., i`$_n$`:Int`]: Index of $t_j = u_j$ on the left `P(Z`$_1$`, ..., Z`$_n$`):Term`: Shape of the formula on the right `Z`$_1$`, ..., Z`$_n$`:Int`]: variables indicating where to substitute
`leftSubstMulti`	1	$\frac{\Gamma, P(t_1, ..., t_n) \vdash \Delta}{\Gamma, P(u_1, ..., u_n) \vdash \Delta}$	`i`$_1$`, ..., i`$_n$`:Int`: Index of $t_j = u_j$ on the left `P(Z`$_1$`, ..., Z`$_n$`):Term`: Shape of the formula on the left `Z`$_1$`, ..., Z`$_n$`:Int`]: variables indicating where to substitute

Table 6
Level 2 rules of SC-TPTP, for one-sided and two-sided sequent calculus.

more technical. A `congruence` step for a sequent $\Gamma \vdash \Delta$ is correct if one of the following cases hold, given all the formulas of the form $s = t$ in Γ:

1. There are two formulas $P(a_1, ..., a_n) \in \Gamma$ and $P(b_1, ..., b_n) \in \Delta$ such that for all i, a_i and b_i are congruent (`hyp` case).
2. There are two formulas $P(a_1, ..., a_n) \in \Gamma$ and $!P(b_1, ..., b_n) \in \Gamma$ such that for all i, a_i and b_i are congruent (`leftHyp` case).
3. There is a formula $a = b \in \Delta$ such that a and b are congruent (`rightRefl` case).

This was immediately useful to us because Goéland uses *Rigid E-Unification* [39, 40] to close branches, which becomes a congruence-like step at proof reconstruction. We implemented a method to eliminate such `congruence` steps in SC-TPTP proofs using e-graphs, as explained in Section 4.2.

4. A Library of Utilities for SC-TPTP

To support the SC-TPTP format, we started the development of a library of tools and utilities to parse, print, verify, and transform SC-TPTP proofs. We chose Scala to implement it because it is a high-level language adapted to the task, it is the language of Lisa and Princess (which we plan to support in the future), and there already exists a complete TPTP parser in Scala, thanks to Alexander Steen[2]. This library of tools is available at https://github.com/SC-TPTP/sc-tptp.

[2]https://github.com/leoprover/scala-tptp-parser

The library contains the syntactic definition of first-order logic, sequents, and deduction steps from Table 1 to Table 6. It also contains a parser (based on the aforementioned one) for SC-TPTP files, as well as a printer and a proof checker, which report incorrect steps. In addition, the library also contains a printer exporting SC-TPTP proofs to Coq proofs. Finally, it contains a tool to eliminate level 2 steps (in particular `congruence`).

4.1. Proof Export to Coq

The translation of SC-TPTP proofs to Coq proofs is relatively straightforward, as each sequent calculus step can be translated into a Coq lemma. Our translation relies on a one-to-one mapping between SC-TPTP rules and Coq lemmas, as exemplified below:

Example 4.1. Translation of `rightAnd` in Coq.

```
Lemma rightAnd : forall P Q : Prop, (P) → (Q) → (P ∧ Q).
Proof. intros P Q H. split. auto. auto. Qed.
```

Translation of rules that generate one conclusion is pretty direct. However, as Coq is based on *intuitionistic* logic, it only allows the conclusion to have at most one element. Consequently, rules that create two formulas on the right of the sequent must be rearranged to work with the hypothesis, as in the following example:

Example 4.2. Translation of `rightOr` in Coq.

```
Lemma rightOr : forall P Q : Prop, ∼(∼P ∧ ∼Q) → (P ∨ Q).
Proof. intros P Q H. apply NNPP. intro H1. apply H. split. auto. auto. Qed.
```

This lemma negates the formula, which subsequently allows us to introduce it and generate multiple formulas within the hypothesis. In order to make use of those rules, we need to use `apply` on the corresponding hypothesis before applying *right* rules, and to end with `intro`. For instance, the sequent system *right or* rule would generate `P, Q` in the conclusion of the sequent. Our Coq rule enables that by keeping ¬`P` and ¬`Q` as hypotheses. As such, when needing `P` or `Q`, it suffices to apply ¬`P` or ¬`Q` and then proceed normally.

Moreover, *left* rules require an additional mechanism to be translated into Coq. Indeed, in Coq, two things can be achieved thanks to a formula `A -> B` in the hypothesis: either we have `B` in the conclusion and we can generate `A` in the conclusion, or we have `A` as a hypothesis and we can generate `B` in the hypothesis. However, as our proof is built on an *abductive* way, we want to obtain `A` in hypothesis from a hypothesis `B`. In order to do that, complementarily to the rule itself, we need to define additional lemmas that "invert" the terms in the proof. We thus need to consider a proof with "holes", and wait for Coq to fill them.

Let us illustrate it with the *left implies* rule. The sequent rule states that from two premises (one with ¬`P` and the other one with `Q` as hypothesis), we can infer `P -> Q` as a hypothesis. However, in our proof, we have `P -> Q`, and so we are not able to deduce anything. Luckily, we can define a lemma that inverts the rule in order to make it applicable with the conclusion `P` $->$ `Q`, leaving a hole in the hypothesis, which we will have to prove later by providing a witness for ¬`P` and `Q`. Those inversion steps are suffixed with `_s`.

Example 4.3. Translation of `leftImp` in Coq.

```
Lemma leftImply_s : forall P Q : Prop,
  (~P → False) → (Q → False) → ((P → Q) → False).
Proof. tauto. Qed.

Definition leftImply := fun P Q c hp hq ⇒ leftImply_s P Q hp hq c.
```

Finally, the context of the proof (i.e., constant, predicates, functions, etc, converted into `Parameters`) is retrieved and added at the beginning of the proof.

Example 4.4. Context for the *Drinker's paradox* proof.

```
Parameter sctptp_U : Set. (* universe *)
Parameter sctptp_I : sctptp_U. (* an individual in the universe. *)
Parameter d: sctptp_U → Prop.
Parameter X_0: sctptp_U.
```

All the lemmas used for the translation can be found in `SC-TPTP.v` (https://github.com/SC-TPTP/sc-tptp/tree/main/src). Then, the translation of a full proof can be done by mapping each proof step to its corresponding lemma, using the parameters of the rule.

Example 4.5. Coq proof of the *Drinker's paradox* [37].

```
(* Add SCTPTP.p *)
Parameter sctptp_U : Set. (* universe *)
Parameter sctptp_I : sctptp_U. (* an individual in the universe. *)
Parameter d: sctptp_U → Prop.
Parameter X_0: sctptp_U.

Theorem drinker: ~(~(exists (X: sctptp_U), (d(X) → (forall (Y: sctptp_U), d(Y))))).
Proof.
intro H0.
(* [f3] *) apply H0. exists X_0. apply NNPP. intros H1.
(* [f4] *) apply (leftNotImp _ _  H1).  intros H2 H3.
(* [f5] *) apply H3. intros Sko_0_15. apply NNPP. intros H4.
(* [f6] *) apply H0. exists Sko_0_15. apply NNPP. intros H5.
(* [f7] *) apply (leftNotImp _ _  H5).  intros H6 H7.
(* [f8] *) auto.
Qed.
```

4.2. Unfolding Congruence Steps

As motivated in Section 3.3, to have `congruence` as a level 2 step, we implemented an algorithm unfolding congruence steps into simpler substitution steps. Our approach is based on computing the congruence closure of all subterms of all atomic and negated atomic formulas in the sequent under the equality given left of a sequent. The congruence closure is computed using an e-graph, a dedicated data structure used in automated theorem provers and program optimization. Our implementation of egraph is in particular inspired by [27] and [41].

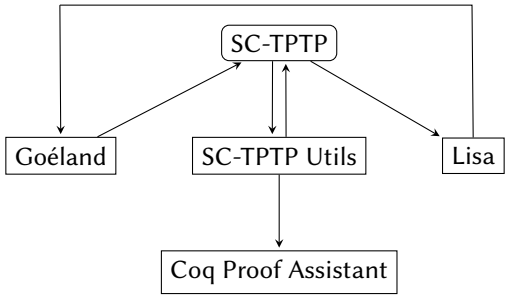

Figure 3: Use cases of SC-TPTP

An e-graph is built on top of a Union-Find data structure, which maintains an equivalence class of terms under a given set of equality (which in an e-graph are either the input equalities or equalities that follow from congruence). To produce SC-TPTP proofs, our Union-Find data structure is equipped with an explain method, as in [41], which when prompted to explain $a = c$ outputs a path $(a, b_1), (b_1, b_2), ..., (b_n, c)$ of equalities. We also record whether an edge comes from an external equality or is a congruence. Congruence edges are then recursively justified by the explain method.

Example 4.6. Consider the following sequent, justified by a `congruence` step:

$$(a = b, b = c, P(f(a))) \vdash \neg P(f(c))$$

The explanation of $f(a) = f(c)$ is simply *congruence*$(f(a), f(c))$, and recursively the explanation of $a = c$ is *external*(a, b), *external*(b, c). For any two congruent terms, the proof of equality is produced with a constant number of steps for each edge in the path between the two terms.

5. Interoperability & Related Tools

This section introduces various tools able to deal with the SC-TPTP format. This format is currently used by the Lisa proof assistant and the Goéland automated theorem prover, as an export format as well as a means of communication between the two tools. A big picture of the SC-TPTP format use case is available in Figure 3.

5.1. Goéland

Goéland [3, 42] is an automated theorem prover for first-order logic with equality. It relies on a concurrent proof-search procedure based on the method of free-variable analytics tableaux that allows it to perform a fair branch exploration. The prover is also able to deal with axiomatisable theories thanks to a module of deduction modulo theory [43], to deal with polymorphic types, and to produce machine-checkable proofs in Coq, Lambdapi and Lisa.

5.2. Lisa

Lisa [2] is a proof assistant based on first-order logic, with set-theoretic foundations. Its proof system is inspired by sequent calculus, with additional built-in proof steps to allow for a more

efficient representation of typical transformation, such as the substitution of equivalent formulas, simultaneous substitution, quantified substitution, and transformation modulo orthologic [44, 45]. All the proof steps from Table 1 to Table 5 are readily convertible to Lisa's Kernel steps.

We implemented a printer, that exports queries about a conjectured sequent as a SC-TPTP problem file (but really a TPTP file, since it contains no proof), and a parser for SC-TPTP proofs directly to Lisa's Kernel proofs. This allowed us to implement a proof tactic in the user interface, directly using Goéland to justify steps prompted by a Lisa user, as in the next example.

Example 5.1. The Goéland tactic in Lisa, proving the drinkers problem.

```
1 val drinkers = Theorem(∃(x, ∀(y, Q(x) ==> Q(y)))) {
2    have(thesis) by Goeland
3 }
```

6. Conclusion

Our goal with SC-TPTP is to provide a common format to allow sequent-based tools to communicate and exchange proofs, but the adoption of such a format entirely relies on the involvement of the community. In order to increase its potential user base, we plan to expand this format in multiple directions.

The first one will be to increase the number of tools able to deal with this format, starting with tableau-based theorem provers such as Princess [46] or Zenon [31, 32]. We also want to support the Connection Calculus [47, 48], related to Tableaux and used by the Connect++ ATP [22] and unify with existing work to export proofs from connection calculi to TPTP. Longer term, we are interested in generalizing our tool to resolution, but this requires additional steps in order to make resolution proofs readily machine-checkable.

Our proof system can be expanded in many ways, which we hope to explore in the future. An important extension would be the support for typed first order logic. In order to do this, our format needs to be extended to fit with TFF [49]. The addition of theory reasoning is also of interest. Theory rules should be possible to add as high-level proof steps, that could be either exported as-is in a proof assistant (if this later is able to deal with the theory), or unfolded in the same way as equality reasoning. Deskolemization [50, 51] is also a strong candidate step. We plan to extend our proof-producing module to export proofs to other proof assistants, such as Lambdapi [6], Isabelle [52] or Lean [53].

While developping SC-TPTP, we have also encountered limitations within the TPTP syntax. As part of our commitment to continuous improvement, we are eager to offer suggestions for refinement. These suggestions may include introducing an *exists unique* quantifier, permitting n-ary conjunctions and disjunctions, or proposing a specialized character for expressing identifiers in TPTP files, allowing to bind and reuse formulas (and other syntactic expressions). By addressing these limitations, we aim to fortify the foundation of our framework for the collective benefit of the community.

Acknoledgment This publication is based upon work from COST Action EuroProofNet, supported by COST (European Cooperation in Science and Technology, www.cost.eu)

References

[1] G. Sutcliffe, The logic languages of the TPTP world, Logic Journal of the IGPL 31 (2023) 1153–1169. doi:10.1093/jigpal/jzac068.

[2] S. Guilloud, S. Gambhir, V. Kuncak, LISA – A Modern Proof System, in: 14th Conference on Interactive Theorem Proving, Leibniz International Proceedings in Informatics, Daghstuhl, Bialystok, 2023, pp. 17:1–17:19.

[3] J. Cailler, J. Rosain, D. Delahaye, S. Robillard, H. L. Bouziane, Goéland: A Concurrent Tableau-Based Theorem Prover (System Description), in: J. Blanchette, L. Kovács, D. Pattinson (Eds.), Automated Reasoning, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 359–368. doi:10.1007/978-3-031-10769-6_22.

[4] M. Kohlhase, F. Rabe, Experiences from Exporting Major Proof Assistant Libraries, Journal of Automated Reasoning 65 (2021) 1265–1298. doi:10.1007/s10817-021-09604-0.

[5] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, Cvc5: A Versatile and Industrial-Strength SMT Solver, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, volume 13243, Springer International Publishing, Cham, 2022, pp. 415–442. doi:10.1007/978-3-030-99524-9_24.

[6] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, R. Saillard, Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory (2016).

[7] G. Sutcliffe, The TPTP Problem Library and Associated Infrastructure, Journal of Automated Reasoning 59 (2017) 483–502. doi:10.1007/s10817-017-9407-7.

[8] S. Böhme, T. Weber, Designing proof formats: A user's perspective—experience report, in: First International Workshop on Proof eXchange for Theorem Proving-PxTP 2011, 2011.

[9] G. Reger, M. Suda, Checkable proofs for first-order theorem proving., in: ARCADE@ CADE, 2017, pp. 55–63.

[10] N. Wetzler, M. J. Heule, W. A. Hunt Jr, Drat-trim: Efficient checking and trimming using expressive clausal proofs, in: International Conference on Theory and Applications of Satisfiability Testing, Springer, 2014, pp. 422–429.

[11] L. Cruz-Filipe, M. J. Heule, W. A. Hunt, M. Kaufmann, P. Schneider-Kamp, Efficient certified rat verification, in: Automated Deduction–CADE 26: 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6–11, 2017, Proceedings, Springer, 2017, pp. 220–236.

[12] A. Stump, D. Oe, 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, 2008, pp. 27–32.

[13] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, et al., cvc5: A versatile and industrial-strength smt solver, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2022, pp. 415–442.

[14] L. De Moura, N. Bjørner, Z3: An efficient smt solver, in: International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.

[15] L. M. de Moura, N. S. Bjørner, Proofs and refutations, and z3., in: LPAR Workshops, volume 418, Doha, Qatar, 2008, pp. 123–132.
[16] F. Besson, P. Fontaine, L. Théry, A flexible proof format for smt: A proposal, in: First International Workshop on Proof eXchange for Theorem Proving-PxTP 2011, 2011.
[17] J. Hoenicke, T. Schindler, A simple proof format for smt., in: SMT, 2022, pp. 54–70.
[18] H.-J. Schurr, M. Fleury, H. Barbosa, P. Fontaine, Alethe: Towards a generic smt proof format, arXiv preprint arXiv:2107.02354 (2021).
[19] T. Bouton, D. Caminha B. de Oliveira, D. Déharbe, P. Fontaine, verit: an open, trustable and efficient smt-solver, in: International Conference on Automated Deduction, Springer, 2009, pp. 151–156.
[20] J. Otten, S. Holden, A syntax for connection proofs (2023).
[21] J. Otten, W. Bibel, leancop: lean connection-based theorem proving, Journal of Symbolic Computation 36 (2003) 139–161.
[22] S. B. Holden, Connect++: A New Automated Theorem Prover Based on the Connection Calculus, in: J. Otten, W. Bibel (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, volume 3613 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 95–106.
[23] S. Baek, The tesc proof format for first-order atps, 2020.
[24] L. Kovács, A. Voronkov, First-order theorem proving and vampire, in: International Conference on Computer Aided Verification, Springer, 2013, pp. 1–35.
[25] S. Schulz, S. Cruanes, P. Vukmirović, Faster, higher, stronger: E 2.3, in: Automated Deduction–CADE 27: 27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings 27, Springer, 2019, pp. 495–507.
[26] G. Sutcliffe, Semantic derivation verification: Techniques and implementation, International Journal on Artificial Intelligence Tools 15 (2006) 1053–1070.
[27] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, P. Panchekha, Egg: Fast and extensible equality saturation, Proceedings of the ACM on Programming Languages 5 (2021) 23:1–23:29. doi:10.1145/3434304.
[28] G. Gentzen, Untersuchungen über das logische Schließen I, Mathematische Zeitschrift 39 (1935) 176–210.
[29] G. Gentzen, Untersuchungen über das logische schließen. ii., Mathematische zeitschrift 39 (1935).
[30] P. Rümmer, A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic, in: I. Cervesato, H. Veith, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 274–289. doi:10.1007/978-3-540-89439-1_20.
[31] R. Bonichon, D. Delahaye, D. Doligez, Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs, in: N. Dershowitz, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, volume 4790, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 151–165. doi:10.1007/978-3-540-75560-9_13.
[32] D. Delahaye, D. Doligez, F. Gilbert, P. Halmagrand, O. Hermant, Zenon Modulo: When Achilles Outruns the Tortoise Using Deduction Modulo, in: D. Hutchison, T. Kanade,

J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, K. McMillan, A. Middeldorp, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, volume 8312, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 274–290. doi:10.1007/978-3-642-45221-5_20.

[33] B. Beckert, J. Posegga, LeanTAP: Lean tableau-based theorem proving, in: A. Bundy (Ed.), Automated Deduction — CADE-12, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1994, pp. 793–797. doi:10.1007/3-540-58156-1_62.

[34] A. S. Troelstra, H. Schwichtenberg, Basic Proof Theory, Cambridge Tracts in Theoretical Computer Science, 2 ed., Cambridge University Press, Cambridge, 2000. doi:10.1017/CBO9781139168717.

[35] J. Harrison, HOL Light: An Overview, in: S. Berghofer, T. Nipkow, C. Urban, M. Wenzel (Eds.), Theorem Proving in Higher Order Logics, volume 5674, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 60–66. doi:10.1007/978-3-642-03359-9_4.

[36] G. Sutcliffe, The szs ontologies for automated reasoning software., in: LPAR Workshops, volume 418, Citeseer, 2008.

[37] R. M. Smullyan, What is the name of this book? The riddle of Dracula and other logical puzzles, Englewood Cliffs, N.J. : Prentice-Hall, 1978.

[38] K. Maziarz, T. Ellis, A. Lawrence, A. Fitzgibbon, S. P. Jones, Hashing Modulo Alpha-Equivalence (2021) 17.

[39] J. H. Gallier, S. Raatz, W. Snyder, Theorem Proving Using Rigid E-Unification Equational Matings, in: Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987, IEEE Computer Society, 1987, pp. 338–346.

[40] A. Degtyarev, A. Voronkov, What you always wanted to know about rigid e-unification, Journal of Automated Reasoning 20 (1998) 47–80.

[41] R. Nieuwenhuis, A. Oliveras, Proof-Producing Congruence Closure, in: J. Giesl (Ed.), Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings, volume 3467 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 453–468. doi:10.1007/978-3-540-32033-3_33.

[42] J. Cailler, Designing an Automated Concurrent Tableau-Based Theorem Prover for First-Order Logic, Ph.D. thesis, Université de Montpellier, 2023.

[43] G. Dowek, T. Hardin, C. Kirchner, Theorem proving modulo, Journal of Automated Reasoning 31 (2003) 33–72.

[44] S. Guilloud, M. Bucev, D. Milovancevic, V. Kuncak, Formula Normalizations in Verification, in: 35th International Conference on Computer Aided Verification, Lecture Notes in Computer Science, Springer, Paris, 2023, pp. 398–422.

[45] S. Guilloud, V. Kuncak, Orthologic with Axioms, 2023. doi:10.48550/arXiv.2307.07569. arXiv:2307.07569.

[46] P. Rümmer, A constraint sequent calculus for first-order logic with linear integer arithmetic, in: Proceedings, 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, volume 5330 of *LNCS*, Springer, 2008, pp. 274–289.

[47] J. Otten, Connection Calculi for Automated Theorem Proving in Classical and Non-Classical Logics, Ph.D. thesis, University of Potsdam, 2013.

[48] C. Kaliszyk, Efficient Low-Level Connection Tableaux, in: H. De Nivelle (Ed.), Auto-

mated Reasoning with Analytic Tableaux and Related Methods, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2015, pp. 102–111. doi:10.1007/978-3-319-24312-2_8.

[49] J. C. Blanchette, A. Paskevich, Tff1: The tptp typed first-order form with rank-1 polymorphism, in: Automated Deduction–CADE-24: 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings 24, Springer, 2013, pp. 414–420.

[50] R. Bonichon, O. Hermant, A Syntactic Soundness Proof for Free-Variable Tableaux with on-the-fly Skolemization, 2013.

[51] J. Rosain, R. Bonichon, J. Cailler, O. Hermant, A generic deskolemization strategy, in: Logic for Programming, Artificial Intelligence, and Reasoning: 25th International Conference, LPAR-25, Balaclava, Mauritius, May 26-31, 2024. Proceedings 25, Springer, 2024.

[52] T. Nipkow, M. Wenzel, L. C. Paulson, Isabelle/HOL: a proof assistant for higher-order logic, Springer, 2002.

[53] J. Avigad, L. De Moura, S. Kong, Theorem proving in lean, Release 3 (2015) 1–4.