The TPTP Format for Clausal Tableaux Proofs

Geoff Sutcliffe[1][0000−0001−9120−3927]✉,
Sean B. Holden[2][0000−0001−7979−1148], and
Mantas Baksys[2][0000−0001−9532−1007]

[1] University of Miami, USA, `geoff@cs.miami.edu`,
[2] University of Cambridge, United Kingdom,
`sbh11@cl.cam.ac.uk,mb2412@cam.ac.uk`

Abstract. This paper describes the (new) TPTP format for recording clausal tableau proofs. The format builds on the existing infrastructure of the TPTP World, in particular the TPTP format for recording derivations. An ATP system that outputs tableaux in this format is described. Existing TPTP World tools for verifying and viewing derivations can be directly extended to verify and view tableaux recorded in this new format.

Keywords: TPTP, tableau, proof

1 Introduction

Automated Theorem Proving (ATP) [22] is concerned with the development and use of software that automates sound reasoning: the derivation of conclusions that follow inevitably from known facts. ATP is at the heart of many computational tasks, including sensitive tasks such as software/hardware verification [6] and system security [3]. ATP systems are often used as components of more complex Artificial Intelligence (AI) systems, which means that the impact of ATP extends into many facets of society. In many of these applications the use of ATP systems is mission critical, in the sense that incorrect results from ATP might have nasty consequences. Facing the demand for error-free results from ATP systems is the reality that ATP systems are complex pieces of software, implementing complex calculi with complex data structures and algorithms [23]. Despite best intentions and efforts, incorrect results are possible. To counter incorrectness, an ATP system can be required to output a proof that serves as a certificate for the system's claim. To ensure that a proof is correct, proof verification can be required, which serves as a certification (but not a certificate) of the proof.

At one top level, proofs can be divided into two types: Hilbert-style proofs that start at axioms and derive theorems [4], and proofs-by-contradiction that negate the conjecture to be proved and derive a contradiction with the axioms [11]. ATP systems that search for Hilbert-style proofs are often called "natural deduction systems", e.g., THINKER [20] and Muscadet [19]. Most contemporary high-performance ATP systems that search for a proof-by-contradiction, called a refutation in this context, use a saturation based approach [23], e.g., E [24] and Vampire [10]. A complementary approach is taken in systems that build tableaux and closed connection matrices [5], e.g., leanCoP [14] and Connect++ [7]. The TPTP World (see Section 2) has an established format for writing Hilbert-style proofs and refutations [40], and has a tool for verifying proofs in that format (see Section 2.1), but has not yet settled on a format for tableaux and connection proofs. One early proposal for a TPTP style format never gained traction [17]. A more recent proposal [16] in the TPTP style provided some inspiration for the new format.

This paper is structured as follows: Section 2 introduces the TPTP World, providing the necessary background to the TPTP language, explains the TPTP format for recording (non-tableau) derivations, and describes the GDV and IDV tools for verifying and viewing derivations. Section 3 describes the new TPTP format for recording clausal tableau proofs, and explains how the format meets the requirements for easy reconstruction of the tableau, and for semantic verification of the tableau inference steps. Section 4 describes an ATP system that can output tableaux in the new format, and explains how the GDV and IDV tools can be directly extended to verify and view tableaux. Section 5 concludes.

2 The TPTP World

The TPTP World [35] is a well-established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems. The TPTP World infrastructure includes the TPTP language [40], the TPTP problem library [31], the TSTP solution library [32], the SZS ontologies [30], the Specialist Problem Classes (SPCs) and problem difficulty ratings [42], SystemOnTPTP [28] and StarExec [27], and the CADE ATP System Competition (CASC) [33]. The problem library is a large collection of Thousands of Problems for Theorem Proving – hence the name. The problem library contains over 25000 problems from over 50 different domains, written in the TPTP language. The problems are categorized into Specialist Problem Classes according to their syntactic and logical status [30]. The TSTP solution library is the result of running numerous ATP systems on that library and collecting their output. The solutions are categorized according to their logical and output form [30]. The TPTP and TSTP libraries, with their categorizations, provide the basis for assigning a difficulty rating to each problem, according to the number of ATP systems that are able to solve it [42].

The most salient feature of the TPTP World for this work is the TPTP language. The TPTP language [34] is one of the keys to the success of the TPTP World. The TPTP language is used for writing both problems and solutions, which enables convenient communication between ATP systems and tools. Originally the TPTP World supported only first-order clause normal form (CNF) [41]. Over the years full first-order form (FOF) [31], typed first-order form (TFF) [39,2], typed extended first-order form (TXF) [38], typed higher-order form (THF) [36,9], and non-classical forms (NTF) [26] have been added.

A general principle of the TPTP language is: "We provide the syntax, you provide the semantics". As such, there is no a priori commitment to any semantics for each of the language forms, although in almost all cases the intended logic and semantics are well known.

Problems and solutions are built from *annotated formulae* of the form

language(*name*, *role*, *formula*, *source*, *useful_info*)

The *language*s supported are `cnf` (clause normal form), `fof` (first-order form), `tff` (typed first-order form), and `thf` (typed higher-order form). The *role*, e.g., `axiom`, `lemma`, `conjecture`, defines the use of the formula. In a *formula*, terms and atoms follow Prolog conventions – functions and predicates start with a lowercase letter or are `'single quoted'`, and variables start with an uppercase letter. The language also supports interpreted symbols that either start with a `$`, e.g., the truth constants `$true` and `$false`, or are composed of non-alphabetic characters, e.g., integer/rational/real numbers such as 27, 43/92, -99.66. The logical connectives in the TPTP language are `!>`, `?*`, `@+`, `@-`, `!`, `?`, `~`, `|`, `&`, `=>`, `<=`, `<=>`, and `<~>`, for the mathematical connectives Π, Σ, choice (indefinite description), definite description, $\forall$, $\exists$, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$, and $\oplus$ respectively. Equality and inequality are expressed as the infix operators `=` and `!=`. The *source* and *useful_info* are optional. Figure 1 shows an example problem.

```
%------------------------------------------------------------------------
fof(a1,axiom,         ~ ( ~q(b) & ? [X] : s(X) ) ).
fof(a2,axiom,         ( r & q(b) ) => ! [X] : ~p(X) ).
fof(a3,axiom,         p(c) | ! [Y] : ( ~q(c) & q(Y) ) ).
fof(a4,axiom,         ~q(c) => ~q(b) ).
fof(a5,axiom,         p(c) => r ).
fof(prove,conjecture, ! [X] : ( ~s(X) & ~q(b) & p(c) ) ).
%------------------------------------------------------------------------
```

Fig. 1. An example problem in TPTP format

2.1 The TPTP Format for Derivations

A derivation written in the TPTP language is a list of annotated formulae. The leaves of a derivation typically have the role `axiom` or `conjecture`, and the inferred formulae typically have the role `plain` or `negated_conjecture` (also used for leaves in CNF). The source is either a `file` record for leaves or an `inference` record for inferred formulae. A `file` record contains the problem file name and the corresponding annotated formulae name in the problem file. An `inference` record contains the inference rule name, a list of useful inference information, and a list of the inference parents. The inference parents can be annotated formulae names, and nested `inference` records. Common types of useful inference information are the semantic relationship of the inferred formula to its parents as an SZS ontology value [30] in a `status` record, special information about recognized types of complex inference rules, e.g., Skolemization and splitting, and details of new symbols introduced in the inference. The use of SZS values is core to GDV's approach to verification, described below.

Figure 2 shows the transformation to CNF of the problem in Figure 1, which is used in the refutation discussed here, and the tableau proof in Section 3. Points of note: all the leaf formulae except `a4` and `a5` are copies of formulae in the problem; `a4` and `a5` are easily derived from the corresponding problem formulae; many of the formulae are inferred with SZS status `thm`, i.e., they are logical consequences of their parents; the negated conjecture `c_0_7` is inferred with SZS status `cth` – its negation is a logical consequence of its parent; the Skolemized formula `c_0_11` is inferred with SZS status `esa` – it is equisatisfiable with its parent;

Figure 3 shows the final steps of the refutation found by E 3.2.5 for the problem in Figure 1, following the transformation to CNF shown in Figure 2.[3] Points of note: many of the formulae are inferred with SZS status `thm`, i.e., they are logical consequences of their parents; several of the inferred formulae, e.g., `c_0_23`, have nested `inference` records – the intermediate inferred formulae have not been output.

2.2 TPTP World Tools for Verifying and Viewing Derivations

The GDV derivation verifier [29] verifies proofs in the TPTP format. GDV's input is a TPTP format proof, and optionally (required for complete verification) the problem for which the proof was produced. GDV verifies a proof in four phases: structural verification, leaf verification, rule-specific verification, and inference verification. Many of the checks rely on a "check-by-ATP", which calls a trusted ATP system – either a theorem prover or a model finder (the systems have become trusted through the process described in [37]). Structural verification deals with non-logical aspects of a proof, checking whether the annotated formulae of the proof have the right format and relationships, e.g., the derivation is acyclic. Leaf verification deals with the leaves of the derivation, and their relationship with the problem annotated formulae, e.g., the leaves are copies of problem formulae or can be proved from problem formulae using a check-by-ATP. Rule specific verification deals with inference rules that require special treatment, e.g., Skolemization. Inference verification deals with the various types of inferences that are made by ATP systems in a proof. Examples of checks are: for inference steps with SZS status `thm` check-by-ATP that the inferred formula can be proved from the parent formulae, for inference steps with SZS status `cth` check-by-ATP that the negation of the inferred formula can be proved from the parent formulae. GDV is available online in SystemOnTSTP.[4]

The Interactive Derivation Viewer (IDV) [43] provides a graphical rendering of a proof DAG. It has features that allow the user to examine the proof structure in various ways, including identification of "interesting" steps in the proof [21]. Figure 4 shows the IDV rendering of (the full version of) the refutation in Figure 3. IDV is available online in SystemOnTSTP.[5]

3 The (new) TPTP Format for Clausal Tableau Proofs

There were three primary requirements for the new format for a clausal tableau:

[3] The full refutation can be generated in SystemOnTPTP, available at `tptp.org/cgi-bin/SystemOnTPTP`

[4] Available at `tptp.org/cgi-bin/SystemOnTSTP`

[5] Available at `tptp.org/cgi-bin/SystemOnTSTP`

```
%------------------------------------------------------------------------
fof(a1,axiom,             ~ ( ~q(b) & ? [X] : s(X) ),
    file('PaperFOF.p',a1) ).
fof(a2,axiom,             ( r & q(b) ) => ! [X] : ~p(X),
    file('PaperFOF.p',a2) ).
fof(a3,axiom,             p(c) | ! [Y] : ( ~q(c) & q(Y) ),
    file('PaperFOF.p',a3) ).
fof(a4,axiom,             ~q(c) => ~q(b),
    file('PaperFOF.p',a4) ).
fof(a5,axiom,             p(c) => r,
    file('PaperFOF.p',a5) ).
fof(prove,conjecture,     ! [X] : ( ~s(X) & ~q(b) & p(c) ),
    file('PaperFOF.p',prove) ).

fof(nc1,negated_conjecture, ~ ! [X] : ( ~s(X) & ~q(b) & p(c) ),
    inference(negate,[status(cth)],[prove]) ).
fof(nc2,negated_conjecture, ? [X] : ~ ( ~s(X) & ~q(b) & p(c) ),
    inference(negate,[status(thm)],[nc1]) ).
fof(nc3,negated_conjecture, ~ ( ~s(sK1) & ~q(b) & p(c) ),
    inference(skolemize,[status(esa),new_symbols(skolem,[sK1]),
skolemized(X)],[nc2]) ).

cnf(c1,plain,              ( q(b) | ~s(X) ),
    inference(clausify,[status(thm)],[a1]) ).
cnf(c2,plain,              ( ~q(b) | ~p(X) | ~r ),
    inference(clausify,[status(thm)],[a2]) ).
cnf(c3,plain,              ( p(c) | ~q(c) ),
    inference(clausify,[status(thm)],[a3]) ).
cnf(c4,plain,              ( p(c) | q(Y) ),
    inference(clausify,[status(thm)],[a3]) ).
cnf(c5,plain,              ( q(c) | ~q(b) ),
    inference(clausify,[status(thm)],[a4]) ).
cnf(c6,plain,              ( r | ~p(c) ),
    inference(clausify,[status(thm)],[a5]) ).
cnf(c7,negated_conjecture,  ( s(sK1) | q(b) | ~p(c) ),
    inference(clausify,[status(thm)],[nc3]) ).
%------------------------------------------------------------------------
```

Fig. 2. Clausification for CNF-based proofs of the problem in Figure 1

1. Easy reconstruction of the tableau.
2. Sufficient information for structural verification of the closed tableau.
3. Sufficient information for semantic verification of the inference steps.

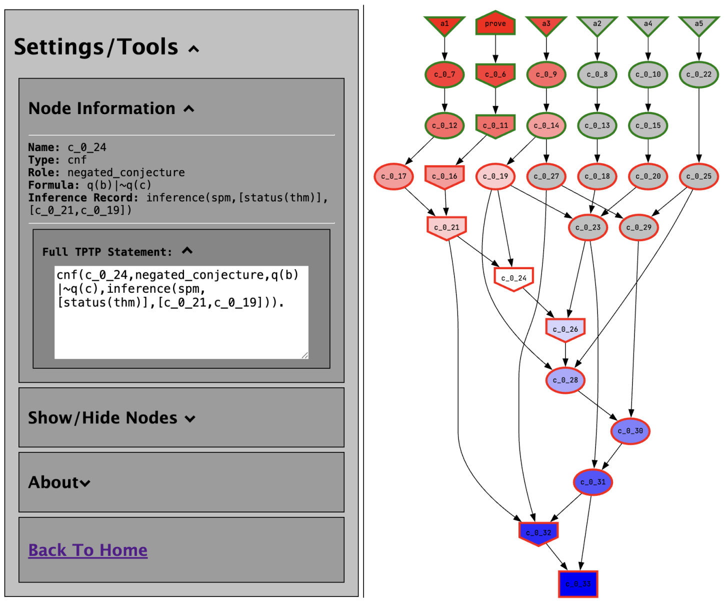
Additional requirements adopted from [16] are:

4. Concise and simple enough for a natural representation of proofs.
5. Readable by humans as well as ATP tools.

```
%------------------------------------------------------------------------
cnf(c_0_23,plain,              ~r | ~q(b),
    inference(csr,[status(thm)],
[inference(spm,[status(thm)],[c_0_18, c_0_19]), c_0_20])).

cnf(c_0_24,negated_conjecture, q(b) | ~q(c),
    inference(spm,[status(thm)],[c_0_21, c_0_19])).

cnf(c_0_25,plain,              r | ~p(c),
    inference(split_conjunct,[status(thm)],[c_0_22])).

cnf(c_0_26,negated_conjecture, ~r | ~q(c),
    inference(spm,[status(thm)],[c_0_23, c_0_24])).

cnf(c_0_27,plain,              q(X1) | p(c),
    inference(split_conjunct,[status(thm)],[c_0_14])).

cnf(c_0_28,plain,              ~q(c),
    inference(csr,[status(thm)],
[inference(spm,[status(thm)],[c_0_25, c_0_19]), c_0_26])).

cnf(c_0_29,plain,              r | q(X1),
    inference(spm,[status(thm)],[c_0_25,c_0_27]) ).

cnf(c_0_30,plain,              r,
    inference(spm,[status(thm)],[c_0_28,c_0_29]) ).

cnf(c_0_31,plain,              ~q(b),
    inference(cn,[status(thm)],
[inference(rw,[status(thm)],[c_0_23,c_0_30])]) ).

cnf(c_0_32,negated_conjecture, q(X1),
    inference(sr,[status(thm)],
[inference(spm,[status(thm)],[c_0_21,c_0_27]),c_0_31]) ).

cnf(c_0_33,plain,              $false,
    inference(cn,[status(thm)],
[inference(rw,[status(thm)],[c_0_31,c_0_32])]),
    [proof] ).
%------------------------------------------------------------------------
```

Fig. 3. The final steps E 3.2.5's refutation of Figure 1

One early proposal for a TPTP style format [17] met requirement 1, but fell short of requirements 2 and 3 due to the omission of some necessary details. As claimed in [16], that early proposal might have also failed to meet requirements 4 and 5. Another existing TPTP style format is the leanTPTP format output by the leanCoP ATP system [15]. The leanTPTP format definitely meets requirement 4, but omits the


Settings/Tools

Node Information

```
Name: c_0_24
Type: cnf
Role: negated_conjecture
Formula: q(b)|~q(c)
Inference Record: inference(spm,[status(thm)],
[c_0_21,c_0_19])
```

Full TPTP Statement:

```
cnf(c_0_24,negated_conjecture,q(b)
|~q(c),inference(spm,
[status(thm)],[c_0_21,c_0_19])).
```

Fig. 4. IDV rendering of the (full version of the) refutation in Figure 3

information required for requirements 1, 2, and 3. A more recent proposal [16] in the TPTP style provided some inspiration for the new format, but left some important information as implicit, which is made explicit in the new format. The new format aims to meet all the requirements.

Figure 5 shows the clausal tableau for the clauses in Figure 2. Figure 6 shows the TPTP format for recording the tableau. The labels in square brackets in Figure 5 identify the literals in the annotated formulae in Figure 6. The dotted arrow shows the one reduction step, the solid rightward arrows point to the lemmas that are created, and the dashed arrows show where the lemmas are used. Note that lemmas can be used only below the parent node of where they are created. The tableau in Figure 5 has the variables instantiated as they would be when the tableau is closed (but note that in general a tableau need not be ground).

The TPTP format recognizes six inference rules, which are described below. The inference record of each annotated formula records the name of the rule used, the SZS status of the inferred formula wrt its parents, the path from the root to the node above, and the inference parents. The SZS status and parent information makes it possible to use semantic verification of each inference, and the path information makes it easy to reconstruct the tableau. The point at which a variable becomes instantiated can optionally be recorded with a `bind()` record, e.g.,

```
cnf(t4,plain,               p(c) | ~ q(c),
    inference(extension,[status(thm),path([t2:2,t1:1]),bind(X,c)],[c3]) ).
```

notes that the variable `X` in `t2` is bound to `c` (assuming variables have been renamed apart so that it is clear that the `X` is in `t2`).

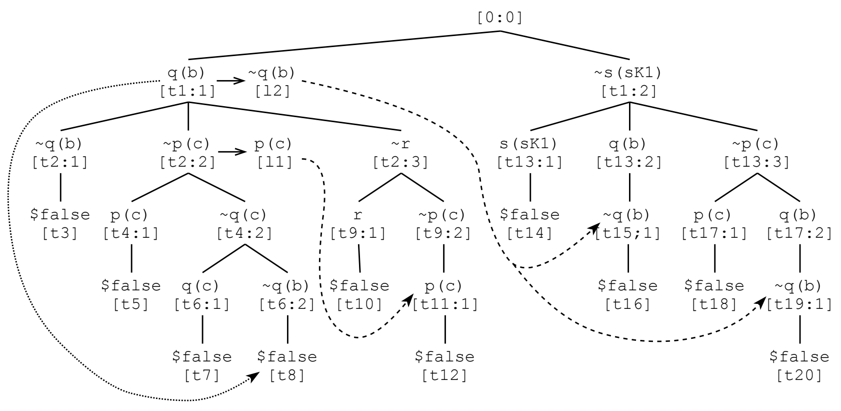


Fig. 5. A tableau for the problem in Figure 1

The Inference Rules

`start`*:* The initial clause below the root node. For example, in Figure 6 `t1` starts the tableau. The path to this point is recorded as `[0:0]`, indicating that the node above is the root node. The logical parent of `t1` is recorded as `[c1]`. The inference has status `thm`, i.e., `t1` is a logical consequence of its parent. `start` can be viewed as a special form of `extension`, described next.

`extension`*:* The standard tableau extension rule. For example, in Figure 6 `t2` extends from the first literal `q(b)` of `t1` to the 1st literal `~q(b)` of `c2`. The path to this point is recorded as `[t1:1,0:0]`. The logical parent of `t2` is recorded as `[c2]`. The inference has status `thm`, i.e., `t2` is a logical consequence of its parent.

`connection`*:* Explicitly close the branch of the contradiction of an extension. For example, in Figure 6 `t3` closes the branch of the contradiction between `q(b)` and `~q(b)` in the extension to `t2`. The path to this point is recorded as `[t2:1,t1:1,0:0]`. The logical parents of `t3` are recorded as `[t2:1,t1:1]`, meaning the 1st literal of `t2` and the 1st literal of `t1`. The inference has status `thm`, i.e., `t3` is a logical consequence of its parents. `connection` can be viewed as a degenerate form of `reduction`, described next.

`reduction`*:* The standard tableau reduction rule. For example, in Figure 6 `t8` closes the branch of the contradiction between the 2nd literal `~q(b)` of `t6` and the 1st literal `q(b)` of `t1`. In Figure 5 this is denoted by the dotted arrow from `q(b)` to `t8`. The path to this point is recorded as `[t6:2,t4:2,t2:2,t1:1,0:0]`. The logical parents of `t8` are recorded as `[t6:2,t1:1]`, meaning the 2nd literal of `t6` and the 1st literal of `t1`. The inference has status `thm`, i.e., `t8` is a logical consequence of its parents.

`lemma`*:* The creation of a unit lemma when a branch is closed. For example, in Figure 6 `l1` is the lemma `p(c)` created when the branch rooted at `t2:2` is closed. The lemma is the negation of the 2nd literal `~p(c)` of `t2`. Note that the role of the annotated formula records it as a `lemma`. The path to this point is recorded as `[t2:2,t1:1,0:0]`.

In Figure 5 this is denoted by the solid right arrow from `t2:2` to `l1`. As the node `t1:1` is used in a reduction in closing the branch, the lemma is available only below `t1:1`, recorded as `below(t1:1)`. The logical parent of `l1` is recorded as `[t2:2]`, meaning the 2nd literal `~p(c)` of `t2`. The inference has status `cth`, i.e., the negation of `l1` is a logical consequence of its parent.

`lemma_extension`*:* Use of a lemma to close a branch. For example, in Figure 6 `t11:1` is the lemma `l1`, and `t12` is the `connection` that closes the branch down to `t11:1`. In Figure 5 this is denoted by the dashed arrow from `l1` to `t11:1`. The lemma `l1` can be used here because `9:2` is below `t1:1`. The path to this point is recorded as `[t11:1,t9:2,t2:3,t1:1,0:0]`. The logical parent of `t11` is recorded as `[l1:1]`, meaning the 1st literal `p(c)` of `l1`. The inference has status `thm`, i.e., `t11:1` is a logical consequence of its parent. `lemma_extension` can be viewed as a special form of `extension`.

Additional tableau inference rules, e.g., factorization [8] (another view of lemma generation and use), non-unit lemma generation (also known as bottom-up lemma generation) [1,25], lazy paramodulation [18] (to deal with equality), etc., appear to be easily added to this format.

4 ATP Systems and Tools

Connect++ is an ATP for first-order logic with equality, using the connection calculus to construct proofs. [7]. Connect++ implements most of the search methods and other heuristics developed for `leanCoP` versions 2 and later, including restricted backtracking (and an alternative version of backtracking restriction for extensions), iterative deepening by path length, various approaches to start clause selection, definitional clause conversion, deterministic or random re-ordering, and regularity testing. It also incorporates some further options such as miniscoping and polynomial-time unification. It accepts input in the TPTP `cnf` and `fof` formats. Connect++ incorporates a standard schedule similar to that of `leanCoP`, but also allows arbitrary schedules to be specified in a simple language. It produces certificates for proofs in a very simple format described in [7,16]; these can be verified internally, or output to a file and checked independently by a short Prolog program. From version 0.7.0 Connect++ can also produce closed tableau in the new TPTP format described in this paper. Connect++ is implemented in C++ and freely available under the GNU General Public License (GPL) Version 3.[6]

The extension of GDV to verify tableau proofs required extending the structural verification phase, and a minor change to inference verification. Leaf verification remains unchanged, and the rule-specific steps of derivation verification are naturally inapplicable. Structural verification of a tableau proof requires checking:

- All inference parents exist, in particular that the specified literals exist.
- The `path` records define an acyclic DAG, without contradictions in the paths.
- All paths start at the root `0:0` node.
- The tableau is closed, i.e., every branch ends at a `$false` formula.
- All `lemma_extension` steps use lemmas that are below the specified node in the tableau.

[6] Download from `www.cl.cam.ac.uk/~sbh11/connect++.html`

```
%------------------------------------------------------------------------------------------
cnf(t1,plain,              q(b) | ~s(sK1),
    inference(start,[status(thm),path([0:0])],[c1]) ).

cnf(t2,plain,              ~q(b) | ~p(c) | ~r,
    inference(extension,[status(thm),path([t1:1,0:0])],[c2]) ).

cnf(t3,plain,              $false,
    inference(connection,[status(thm),path([t2:1,t1:1,0:0])],[t2:1,t1:1]) ).

cnf(t4,plain,              p(c) | ~q(c),
    inference(extension,[status(thm),path([t2:2,t1:1,0:0])],[c3]) ).

cnf(t5,plain,              $false,
    inference(connection,[status(thm),path([t4:1,t2:2,t1:1,0:0])],[t4:1,t2:2]) ).

cnf(t6,plain,              q(c) | ~q(b),
    inference(extension,[status(thm),path([t4:2,t2:2,t1:1,0:0])],[c5]) ).

cnf(t7,plain,              $false,
    inference(connection,[status(thm),path([t6:1,t4:2,t2:2,t1:1,0:0])],[t6:1,t4:2]) ).

cnf(t8,plain,              $false,
    inference(reduction,[status(thm),path([t6:2,t4:2,t2:2,t1:1,0:0])],[t6:2,t1:1]) ).

cnf(l1,lemma,              p(c),
    inference(lemma,[status(cth),path([t2:2,t1:1,0:0]),below(t1:1)],[t2:2]) ).

cnf(t9,plain,              r | ~p(c),
    inference(extension,[status(thm),path([t2:3,t1:1,0:0])],[c6]) ).

cnf(t10,plain,             $false,
    inference(connection,[status(thm),path([t9:1,t2:3,t1:1,0:0])],[t9:1,t2:3]) ).

cnf(t11,plain,             p(c),
    inference(lemma_extension,[status(thm),path([t9:2,t2:3,t1:1,0:0])],[l1:1]) ).

cnf(t12,plain,             $false,
    inference(connection,[status(thm),path([t11:1,t9:2,t2:3,t1:1,0:0])],[t9:2,t11:1]) ).

cnf(l2,lemma,              ~q(b),
    inference(lemma,[status(cth),path([t1:1,0:0]),below(0:0)],[t1:1]) ).

cnf(t13,plain,             s(sK1) | q(b) | ~p(c),
    inference(extension,[status(thm),path([t1:2,0:0])],[c7]) ).

cnf(t14,plain,             $false,
    inference(connection,[status(thm),path([t13:1,t1:2,0:0])],[t12:1,t1:2]) ).

cnf(t15,plain,             ~q(b),
    inference(lemma_extension,[status(thm),path([t13:2,t1:2,0:0])],[l2:1]) ).

cnf(t16,plain,             $false,
    inference(connection,[status(thm),path([t15:1,t13:2,t1:2,0:0])],[t15:1,t13:2]) ).

cnf(t17,plain,             p(c) | q(b),
    inference(extension,[status(thm),path([t13:3,t1:2,0:0])],[c4]) ).

cnf(t18,plain,             $false,
    inference(connection,[status(thm),path([t17:1,t13:3,t1:2,0:0])],[t17:1,t13:3]) ).

cnf(t19,plain,             ~q(b),
    inference(lemma_extension,[status(thm),path([t17:2,t13:3,t1:2,0:0])],[l2:1]) ).

cnf(t20,plain,             $false,
    inference(connection,[status(thm),path([t19:1,t17:2,t13:3,t1:2,0:0])],[t19:1,t17:2]) ).
%------------------------------------------------------------------------------------------
```

Fig. 6. A tableau proof in TPTP format, for the problem in Figure 1

Semantic verification is essentially the same as for derivations, with the small additional need to extract the specified literals from inference parents.

The IDV derivation viewer has been adapted to display tableau proofs, as the Interactive Tableau Viewer (ITV). Figure 7 shows the ITV rendering of the tableau in Figure 6. The root node is the top `$true` box. Nodes from `extension` steps are in green-bordered ovals, `$false` nodes from `connection` and `reduction` steps are in red-bordered `$false` boxes, lemmas created in `lemma` steps are in blue-bordered triangles, and lemmas used in `lemma_extension` steps are in green-bordered triangles. In `reduction` and `lemma_extension` steps the relevant ancestor/lemma lights up when the cursor is hovered over the reduced/extension node. ITV is available online in SystemOnTSTP.[7]

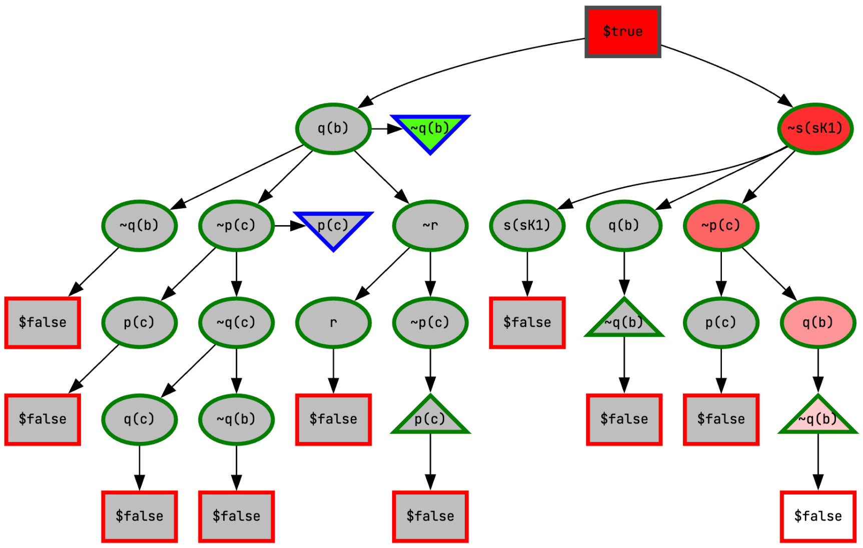


Fig. 7. ITV's rendering of the tableau in Figure 6

5 Conclusion

This paper has described the new TPTP format for recording clausal tableau proofs. The format builds on the existing infrastructure of the TPTP World, in particular the

[7] Available at `tptp.org/cgi-bin/SystemOnTSTP`

TPTP format for recording derivations. The new format meets the requirements listed in Section 3:

1. Easy reconstruction of the tableau - this is can be done directly from the `path`s.
2. Sufficient information for structural verification of the closed tableau - this is provided by the `path`s, the `below` record in lemmas, and the explicit closing of each path with `connection` steps and after `reduction` steps.
3. Sufficient information for semantic verification of the inference steps - this is provided with SZS `status` information.
4. Concise and simple enough for a natural representation of proofs - there is no redundant information or unnecessary detail, and the tableau can be directly reconstructed.
5. Readable by humans as well as ATP tools - the format uses the established TPTP syntax, which is both human and machine readable.

The new format has been adopted in the Connect++ ATP system, and the existing GDV and IDV tools for verifying and viewing derivations have been adapted to verifying and viewing tableaux.

Future work includes completing the extension of GDV, and adding highlighting features to ITV as explained in Section 4. In the larger picture, the format can be extended to record non-clausal tableaux [12,13].

Acknowledgments

– For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission.
– ITV was implemented at the University of Miami by Daniel Li and Esteban Morales, building on the IDV code of Jack McKeown.

References

1. Astrachan, O., Stickel, M.: Caching and Lemmaizing in Model Elimination Theorem Provers. In: Kapur, D. (ed.) Proceedings of the 11th International Conference on Automated Deduction. pp. 224–238. No. 607 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1992)
2. Blanchette, J., Paskevich, A.: TFF1: The TPTP Typed First-order Form with Rank-1 Polymorphism. In: Bonacina, M. (ed.) Proceedings of the 24th International Conference on Automated Deduction. pp. 414–420. No. 7898 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2013)
3. Cook, B.: Formal Reasoning About the Security of Amazon Web Services. In: Chockler, H., Weissenbacher, G. (eds.) Proceedings of the 30th International Conference on Computer Aided Verification. pp. 38–47. No. 10981 in Lecture Notes in Computer Science, Springer-Verlag (2018)
4. Enderton, H.: A Mathematical Introduction to Logic. Academic Press (1972)
5. Furbach, U., Beckert, B., Hähnle, R., Letz, R., Baumgartner, P., Egly, U., Bible, W., Brüning, S., Otten, J., Rath, T., Schaub, T.: Tableau and Connection Calculi. In: Bibel, W., Schmitt, P. (eds.) Automated Deduction - A Basis for Applications, Volume 1: Foundations - Calculi and Methods, pp. 3–179. Kluwer (1998)

6. Hähnle, R., Huisman, M.: Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In: Steffen, B., Woeginger, G. (eds.) Computing and Software Science: State of the Art and Perspectives, pp. 345–373. No. 10000 in Lecture Notes in Computer Science, Springer-Verlag (2019)
7. Holden, S.: 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. pp. 95–106. No. 3613 in CEUR Workshop Proceedings (2023)
8. Johnson, C.: Factorization and Circuit in the Connection Method. Journal of the ACM **40**(3), 536–557 (1993)
9. Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism. In: Fontaine, P., Schulz, S., Urban, J. (eds.) Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning. pp. 41–55. No. 1635 in CEUR Workshop Proceedings (2016)
10. Kovacs, L., Voronkov, A.: First-Order Theorem Proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Proceedings of the 25th International Conference on Computer Aided Verification. pp. 1–35. No. 8044 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2013)
11. Mendelson, E.: Introduction to Mathematical Logic. Wadsworth and Brooks/Cole, 3 edn. (1987)
12. Otten, J.: Non-clausal Connection Calculi for Non-classical Logics. In: Nalon, C., Schmidt, R. (eds.) Proceedings of the 26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 209–227. No. 10501 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2017)
13. Otten, J.: Proof Search Optimizations for Non-clausal Connection Calculi. In: Konev, B., Rümmer, P., Urban, J. (eds.) Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning. pp. 49–57. No. 2162 in CEUR Workshop Proceedings (2018)
14. Otten, J.: 20 Years of leanCoP - An Overview of the Provers. In: Otten, J., Bibel, W. (eds.) Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi. pp. 4–22. No. 3613 in CEUR Workshop Proceedings (2023)
15. Otten, J., Bibel, W.: leanCoP: Lean Connection-based Theorem Proving. Journal of Symbolic Computation **36**(1-2), 139–161 (2003)
16. Otten, J., Holden, S.: A Syntax for Connection Proofs. In: Otten, J., Bibel, W. (eds.) Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi. pp. 84–94. No. 3613 in CEUR Workshop Proceedings (2023)
17. Otten, J., Sutcliffe, G.: Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi. In: Konev, B., Schmidt, R., Schulz, S. (eds.) Proceedings of the Workshop on Practical Aspects of Automated Reasoning. pp. 90–100 (2010)
18. Paskevich, A.: Connection Tableaux with Lazy Paramodulation. Journal of Automated Reasoning **40**(2-3), 179–194 (2008)
19. Pastre, D.: Muscadet 2.3 : A Knowledge-based Theorem Prover based on Natural Deduction. In: Gore, R., Leitsch, A., Nipkow, T. (eds.) Proceedings of the International Joint Conference on Automated Reasoning. pp. 685–689. No. 2083 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2001)
20. Pelletier, F.: Automated Natural Deduction in THINKER. Studia Logica **60**, 3–43 (1998)

21. Puzis, Y., Gao, Y., Sutcliffe, G.: Automated Generation of Interesting Theorems. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference. pp. 49–54. AAAI Press (2006)
22. Robinson, A., Voronkov, A.: Handbook of Automated Reasoning. Elsevier Science (2001)
23. Schulz, S.: Algorithms and Data Structures for First-Order Equational Deduction. In: Benzmüller, C., Fischer, B., Sutcliffe, G. (eds.) Proceedings of the 6th International Workshop on the Implementation of Logics. pp. 1–6. No. 212 in CEUR Workshop Proceedings (2006)
24. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, Higher, Stronger: E 2.3. In: Fontaine, P. (ed.) Proceedings of the 27th International Conference on Automated Deduction. pp. 495–507. No. 11716 in Lecture Notes in Computer Science, Springer-Verlag (2019)
25. Schumann, J.: DELTA - A Bottom-up Preprocessor for Top-Down Theorem Provers. In: Bundy, A. (ed.) Proceedings of the 12th International Conference on Automated Deduction. pp. 774–777. No. 814 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1994)
26. Steen, A., Fuenmayor, D., Gleißner, T., Sutcliffe, G., Benzmüller, C.: Automated Reasoning in Non-classical Logics in the TPTP World. In: Konev, B., Schon, C., Steen, A. (eds.) Proceedings of the 8th Workshop on Practical Aspects of Automated Reasoning. p. Online. No. 3201 in CEUR Workshop Proceedings (2022)
27. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a Cross-Community Infrastructure for Logic Solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Proceedings of the 7th International Joint Conference on Automated Reasoning. pp. 367–373. No. 8562 in Lecture Notes in Artificial Intelligence (2014)
28. Sutcliffe, G.: SystemOnTPTP. In: McAllester, D. (ed.) Proceedings of the 17th International Conference on Automated Deduction. pp. 406–410. No. 1831 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2000)
29. Sutcliffe, G.: Semantic Derivation Verification: Techniques and Implementation. International Journal on Artificial Intelligence Tools **15**(6), 1053–1070 (2006). https://doi.org/{10.1142/S0218213006003119}
30. Sutcliffe, G.: The SZS Ontologies for Automated Reasoning Software. In: Sutcliffe, G., Rudnicki, P., Schmidt, R., Konev, B., Schulz, S. (eds.) Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics. pp. 38–49. No. 418 in CEUR Workshop Proceedings (2008)
31. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning **43**(4), 337–362 (2009)
32. Sutcliffe, G.: The TPTP World - Infrastructure for Automated Reasoning. In: Clarke, E., Voronkov, A. (eds.) Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. pp. 1–12. No. 6355 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2010)
33. Sutcliffe, G.: The CADE ATP System Competition - CASC. AI Magazine **37**(2), 99–101 (2016)
34. Sutcliffe, G.: The Logic Languages of the TPTP World. Logic Journal of the IGPL **31**(6), 1153–1169 (2023)
35. Sutcliffe, G.: Stepping Stones in the TPTP World. In: Benzmüller, C., Heule, M., Schmidt, R. (eds.) Proceedings of the 12th International Joint Conference on Automated Reasoning. pp. 30–50. No. 14739 in Lecture Notes in Artificial Intelligence (2024)

36. Sutcliffe, G., Benzmüller, C.: Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. Journal of Formalized Reasoning **3**(1), 1–27 (2010)
37. Sutcliffe, G., Blanqui, F., Burel, G.: Proof Verification with GDV and LambdaPi - It's a Matter of Trust. In: Biskri, I., Talbert, D. (eds.) Proceedings of the 38th International FLAIRS Conference (2025). `https://doi.org/{10.32473/flairs.38.1.138642}`
38. Sutcliffe, G., Kotelnikov, E.: TFX: The TPTP Extended Typed First-order Form. In: Konev, B., Urban, J., Schulz, S. (eds.) Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning. pp. 72–87. No. 2162 in CEUR Workshop Proceedings (2018)
39. Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP Typed First-order Form with Arithmetic. In: Bjørner, N., Voronkov, A. (eds.) Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. pp. 406–419. No. 7180 in Lecture Notes in Artificial Intelligence, Springer-Verlag (2012)
40. Sutcliffe, G., Schulz, S., Claessen, K., Van Gelder, A.: Using the TPTP Language for Writing Derivations and Finite Interpretations. In: Furbach, U., Shankar, N. (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning. pp. 67–81. No. 4130 in Lecture Notes in Artificial Intelligence, Springer (2006)
41. Sutcliffe, G., Suttner, C.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning **21**(2), 177–203 (1998)
42. Sutcliffe, G., Suttner, C.: Evaluating General Purpose Automated Theorem Proving Systems. Artificial Intelligence **131**(1-2), 39–54 (2001). `https://doi.org/{10.1016/S0004-3702(01)00113-8}`
43. Trac, S., Puzis, Y., Sutcliffe, G.: An Interactive Derivation Viewer. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of the 7th Workshop on User Interfaces for Theorem Provers. Electronic Notes in Theoretical Computer Science, vol. 174, pp. 109–123 (2007)