Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi

Jens Otten
Institut für Informatik
University of Potsdam, Germany
jeotten@cs.uni-potsdam.de

Geoff Sutcliffe
Department of Computer Science
University of Miami, USA
geoff@cs.miami.edu

Abstract

The TPTP language, developed within the framework of the TPTP library, allows the representation of problems and solutions in first-order and higher-order logic. Whereas the writing of solutions in resolution calculi is well documented and used, an appropriate representation of solutions in tableau or connection calculi using the TPTP syntax has not yet been specified. This paper describes how the TPTP language can be used to represent derivations and solutions in standard tableau, sequent and connection calculi for classical first-order logic.

1 Introduction

The TPTP language specifies syntax and semantics for expressing problems in first-order and higher-order logic. It is used not only within the TPTP library [23], but also within similar problem libraries, e.g., the ILTP library [15]. The TPTP syntax for representing problems is used by a variety of automated theorem proving (ATP) systems based on different proof calculi. The TPTP language also allows representation of solutions, e.g., derivations and models, produced by ATP systems. The writing of derivations in resolution calculi is well documented and specified [25]. At the last CADE system competition, CASC-22 [24], three of the five ATP systems that output proofs in the core FOF division use the TPTP syntax. All three of those systems produce proofs that are based on resolution calculi.

Even though the TPTP syntax is flexible, the presentation of derivations in, e.g., tableau, sequent or connection calculi is not straightforward. Derivations in these calculi differ significantly from derivations in the resolution calculus. Whereas the leaves of a proof in the tableau calculus consists of the axioms of the *calculus*, the leaves of a derivation in the resolution calculus consists of the formulae of the given problem; the axiom of the (formal) resolution calculus is the empty clause [18], which occurs only at the root of a refutation.

This paper describes how the TPTP language can be used to represent derivations and proofs in standard tableau and connection calculi. As the sequent calculus is closely related to the tableau calculus, this can easily be adapted to present derivations in the sequent calculus as well. This is a proposed format, not yet formally established as a TPTP standard; community feedback with suggestions for improvement are welcome. The goal is to produce a format that is compatible with the existing format for representing derivations (reviewed in Section 2.2), so that existing TPTP infrastructure for proof processing, e.g., the GDV proof verifier [21] and the IDV proof visualizer [26], can be used with little or no modification.

2 The TPTP Language

The TPTP language is suitable for representing problems as well as derivations in first-order and higher-order logic. The following description presents its main concepts. A detailed definition is part of the TPTP library [23]; see also [25].

```
%--------------------------------------------------------------------------
% File     : SYN054+1 : TPTP v4.0.1. Released v2.0.0.
% Domain   : Syntactic
% Problem  : Pelletier Problem 24
% Status   : Theorem
% Rating   : 0.00 v2.1.0
%--------------------------------------------------------------------------
fof(pel24_1,axiom, ( ~ ( ? [X] : ( big_s(X) & big_q(X) ) ) )).
fof(pel24_2,axiom, ( ! [X] : ( big_p(X) => ( big_q(X) | big_r(X) ) ) )).
fof(pel24_3,axiom, ( ~ ( ? [X] : big_p(X) ) => ? [Y] : big_q(Y) )).
fof(pel24_4,axiom, ( ! [X] : ( ( big_q(X) | big_r(X) ) => big_s(X) ) )).
fof(pel24,conjecture, ( ? [X] : ( big_p(X) & big_r(X) ) )).
%--------------------------------------------------------------------------
```

Figure 1: The presentation of the TPTP problem `SYN054+1`

2.1 Representing Problems

The top level building blocks for problems using the TPTP syntax are annotated formulae, of the following form:

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

The *language* is one of `thf`, `fof`, or `cnf`, for formulae in typed higher-order, first-order, and clause normal form. Each annotated formula has a unique *name*. The *role* is, e.g., `axiom` or `conjecture`. The *source* describes where the formula came from, e.g., an input file, and *useful_info* is a list of user information. The last two fields are optional.

Example 1. *Pelletier's problem 24 [14] consists of the following subformulae.*

$$\begin{array}{cll} \neg(\exists x(Sx \wedge Qx)) & \textit{Axiom 1} & (1) \\ \forall(Px \Rightarrow (Qx \vee Rx)) & \textit{Axiom 2} & (2) \\ \neg(\exists x Px) \Rightarrow \exists y Qy & \textit{Axiom 3} & (3) \\ \forall x((Qx \vee Rx) \Rightarrow Sx) & \textit{Axiom 4} & (4) \\ \exists x(Px \wedge Rx) & \textit{Conjecture} & (5) \end{array}$$

It stands for the first-order formula $(\textit{Axiom 1} \wedge \textit{Axiom 2} \wedge \textit{Axiom 3} \wedge \textit{Axiom 4}) \Rightarrow \textit{Conjecture}$*. This problem is in the TPTP library under the name* `SYN054+1`*. Its representation using the TPTP syntax is given in Figure 1 (with an abbreviated version of the full TPTP header).*

2.2 Representing Derivations

A derivation (in the resolution calculus) is a directed acyclic graph whose leaf nodes are formulae from the problem, and whose interior nodes are formulae inferred from parent formulae. A refutation is a derivation that has the root node *false*, representing the empty clause. A derivation written in the TPTP language is a list of annotated formulae, as for problems. For derivations the *source* has one of the forms

`file(`*file_name,file_info*`)`

`inference(`*inference_name,inference_info,parents*`)`

The former is used for formulae taken from the problem file. The latter is used for inferred formulae, in which *inference_name* is the name of the inference rule applied by the ATP system, *inference_info* is a

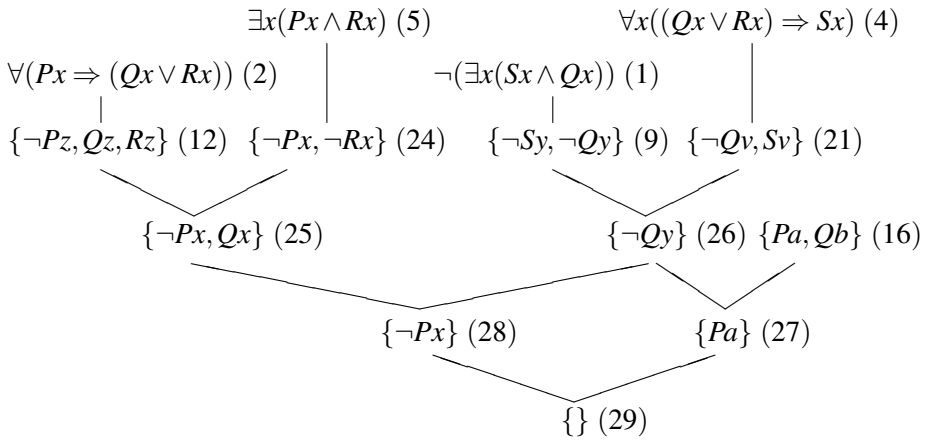


Figure 2: A derivation for SYN054+1 in the resolution calculus

[]ed list of additional information about the inference, and *parents* is a list of the (logical) parents' node names in the derivation. The *inference_info* normally includes a `status()` term that record the semantic relationship between the parents and the inferred formula as an SZS ontology value [22]. Variable bindings applied to a logical parent are captured in `bind`/2 terms following the parent's name.

```
%----------------------------------------------------------------------------------------
fof(1, axiom,~(?[X1]:(big_s(X1)&big_q(X1))),file('SYN054+1.p',pel24_1)).
fof(2, axiom,![X1]:(big_p(X1)=>(big_q(X1)|big_r(X1))),file('SYN054+1.p',pel24_2)).
fof(3, axiom,(~(?[X1]:big_p(X1))=>?[X2]:big_q(X2)),file('SYN054+1.p',pel24_3)).
fof(4, axiom,![X1]:((big_q(X1)|big_r(X1))=>big_s(X1)),file('SYN054+1.p',pel24_4)).
fof(5, conjecture,?[X1]:(big_p(X1)&big_r(X1)),file('SYN054+1.p',pel24)).
fof(6, negated_conjecture,~(?[X1]:(big_p(X1)&big_r(X1))),inference(assume_negation,[],[5])).
fof(7, plain,![X1]:(~(big_s(X1))|~(big_q(X1))),inference(fof_nnf,[],[1])).
fof(8, plain,![X2]:(~(big_s(X2))|~(big_q(X2))),inference(variable_rename,[],[7])).
cnf(9, plain,(~big_q(X1)|~big_s(X1)),inference(split_conjunct,[],[8])).
fof(10,plain,![X1]:(~(big_p(X1))|(big_q(X1)|big_r(X1))),inference(fof_nnf,[],[2])).
fof(11,plain,![X2]:(~(big_p(X2))|(big_q(X2)|big_r(X2))),inference(variable_rename,[],[10])).
cnf(12,plain,(big_r(X1)|big_q(X1)|~big_p(X1)),inference(split_conjunct,[],[11])).
fof(13,plain,(?[X1]:big_p(X1)|?[X2]:big_q(X2)),inference(fof_nnf,[],[3])).
fof(14,plain,(?[X3]:big_p(X3)|?[X4]:big_q(X4)),inference(variable_rename,[],[13])).
fof(15,plain,(big_p(esk1_0)|big_q(esk2_0)),inference(skolemize,[],[14])).
cnf(16,plain,(big_q(esk2_0)|big_p(esk1_0)),inference(split_conjunct,[],[15])).
fof(17,plain,![X1]:((~(big_q(X1))&~(big_r(X1)))|big_s(X1)),inference(fof_nnf,[],[4])).
fof(18,plain,
    ![X2]:((~(big_q(X2))&~(big_r(X2)))|big_s(X2)),inference(variable_rename,[],[17])).
fof(19,plain,
    ![X2]:((~(big_q(X2))|big_s(X2))&(~(big_r(X2))|big_s(X2))),inference(distribute,[],[18])).
cnf(21,plain,(big_s(X1)|~big_q(X1)),inference(split_conjunct,[],[19])).
fof(22,negated_conjecture,![X1]:(~(big_p(X1))|~(big_r(X1))),inference(fof_nnf,[],[6])).
fof(23,negated_conjecture,
    ![X2]:(~(big_p(X2))|~(big_r(X2))),inference(variable_rename,[],[22])).
cnf(24,negated_conjecture,(~big_r(X1)|~big_p(X1)),inference(split_conjunct,[],[23])).

cnf(25,plain,(big_q(X1)|~big_p(X1)),inference(csr,[],[12,24])).
cnf(26,plain,(~big_q(X1)),inference(csr,[],[9,21])).
cnf(27,plain,(big_p(esk1_0)),inference(sr,[],[16,26])).
cnf(28,plain,(~big_p(X1)),inference(sr,[],[25,26])).
cnf(29,plain,($false),inference(sr,[],[27,28])).
%----------------------------------------------------------------------------------------
```

Figure 3: A derivation for SYN054+1 in the resolution calculus using the TPTP syntax

Example 2. *Figure 2 shows a conversion of some of the axioms and the negated conjecture of problem* `SYN054+1` *from Example 1 to clause normal form, and a subsequent refutation of the clause normal form in the resolution calculus [16]. The leaf nodes are the formulae of the problem. As the root node is the empty clause, the derivation is a proof for problem* `SYN054+1`*. The representation of this derivation using the TPTP syntax is given in Figure 3. It is a slightly simplified version of the original proof output by the EP system [17]. The five inferences of the resolution proof are represented by the nodes 25 to 29. The nodes of the proof in Figure 2 are annotated by the corresponding EP node numbers.*

3 Representing Derivations in the Tableau Calculus

Tableau calculi are well-known proof search calculi for classical and non-classical logics [4, 6]. The axiom and 12 rules of a standard tableau calculus for classical logic are given in Table 1 [20]. It uses *signed formulae* of the form A^T or A^F, in which A is a first-order formula and T/F is its sign (or polarity). The signed formula A^F can be interpreted as $A \Rightarrow \mathit{false}$. The usage of signed formulae allows an elegant and uniform representation of the rules of the tableau calculus. The α-rules add formulae to a branch of a derivation, and the β-rules split a branch of the derivation into two branches. When eliminating a universal quantifier using the γ-rule, all free occurrences of the variable x in A are replaced by the term t. In the δ-rule the variable x is replaced by a Skolem term that consists of a unique Skolem function symbol sk_i and all variables $x_1, \ldots, x_n$ that occur free in A. A formula A is valid if, and only if, there is a derivation of A^F in the tableau calculus.

Table 1: The axiom and the rules of the tableau calculus

Axiom	$\dfrac{A^T \quad A^F}{\bot}$		
α-rules	$\wedge^T \ \dfrac{(A \wedge B)^T}{\begin{array}{c}A^T\\B^T\end{array}}$	$\vee^F \ \dfrac{(A \vee B)^F}{\begin{array}{c}A^F\\B^F\end{array}}$	$\Rightarrow^F \ \dfrac{(A \Rightarrow B)^F}{A^T \wedge B^F}$
	$\neg^T \ \dfrac{(\neg A)^T}{A^F}$	$\neg^F \ \dfrac{(\neg A)^F}{A^T}$	
β-rules	$\wedge^F \ \dfrac{(A \wedge B)^F}{A^F \quad B^F}$	$\vee^T \ \dfrac{(A \vee B)^T}{A^T \quad B^T}$	$\Rightarrow^T \ \dfrac{(A \Rightarrow B)^T}{A^F \quad B^T}$
γ-rules	$\forall^T \ \dfrac{(\forall x A)^T}{A^T[x\backslash t]}$	$\exists^F \ \dfrac{(\exists x A)^F}{A^F[x\backslash t]}$	
δ-rules	$\forall^F \ \dfrac{(\forall x A)^F}{A^F[x\backslash sk_i(x_1,\ldots,x_n)]}$	$\exists^T \ \dfrac{(\exists x A)^T}{A^T[x\backslash sk_i(x_1,\ldots,x_n)]}$	

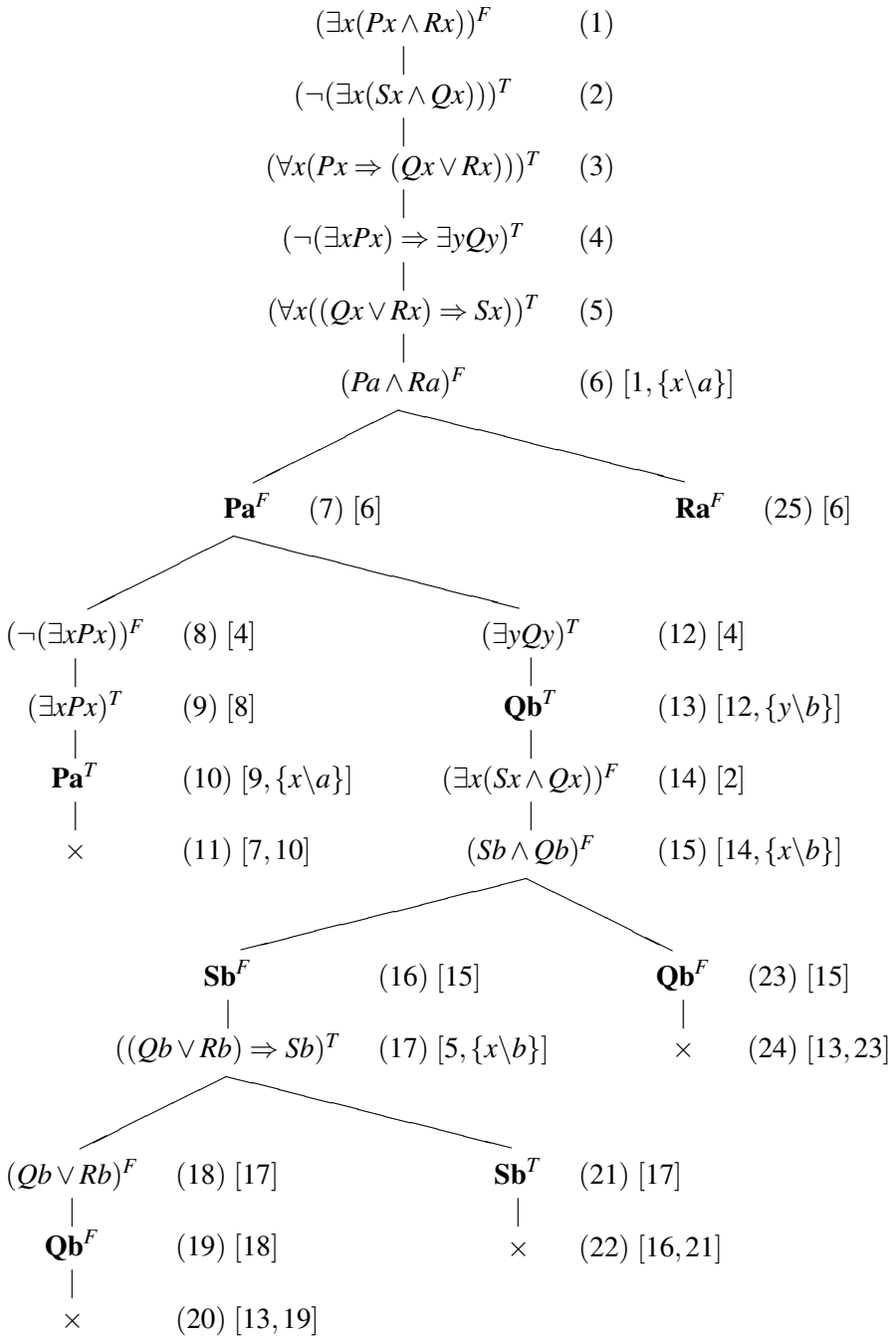


Figure 4: A derivation for `SYN054+1` in the tableau calculus

Example 3. *A derivation of problem* `SYN054+1` *from Example 1 in the tableau calculus is shown in Figure 4. It follows the common representation of standard tableau calculi for classical logic [6]. Each node is annotated in ( )s by its number and in [ ]s by the number of the node whose formula is used as the premise of the inference rule, its* logical parent*. Additionally, a substitution is given when the γ- or δ-rule is applied. The constants a and b are Skolem terms. Branches that are closed by an axiom are marked with $\times$. The derivation in Figure 4 is not a proof, because the rightmost branch (node 25) is not closed by an axiom.*

```
%----------------------------------------------------------------------------------------------------
fof(0, conjecture,?[X]:(big_p(X)&big_r(X)),file('SYN054+1.p',pel24)).
fof(1, negated_conjecture,(~ ?[X]:(big_p(X)&big_r(X)))=>$false,inference(neg_conj,[pparent([0])],[0])).
fof(2, axiom,~ ?[X]:(big_s(X)&big_q(X)),file('SYN054+1.p',pel24_1)).
fof(3, axiom,![X]:(big_p(X)=>(big_q(X)|big_r(X))),file('SYN054+1.p',pel24_2)).
fof(4, axiom,~ ?[X]:big_p(X)=>?[Y]:big_q(Y),file('SYN054+1.p',pel24_3)).
fof(5, axiom,![X]:((big_q(X)|big_r(X))=>big_s(X)),file('SYN054+1.p',pel24_4)).

fof(6, plain,(big_p(X)&big_r(X))=>$false,
       inference(exists_F,[status(thm),pparent([5])],[1:[bind(X,$fot(a))]])).

fof(7, plain,~big_p(a),inference(and_F,[and_F(split,[position(l)]),pparent([6])],[6])).
fof(8, plain,(~ ?[X]:big_p(X))=>$false,
       inference(implies_T,[implies_T(split,[position(ll)]),pparent([7])],[4])).
fof(9, plain,?[X]:big_p(X),inference(neg_F,[status(thm),pparent([8])],[8])).
fof(10,plain,big_p(a),inference(exists_T,[status(thm),pparent([9])],[9])).
fof(11,plain,$false,inference(axiom,[status(thm),pparent([10])],[7,10])).

fof(12,plain,?[Y]:big_q(Y),inference(implies_T,[implies_T(split,[position(lr)]),pparent([7])],[4])).
fof(13,plain,big_q(b),inference(exists_T,[status(thm),pparent([12])],[12:[bind(Y,$fot(b))]])).
fof(14,plain,(?[X]:(big_s(X)&big_q(X)))=>$false,inference(neg_T,[status(thm),pparent([13])],[2])).
fof(15,plain,(big_s(b)&big_q(b))=>$false,
       inference(exists_F,[status(thm),pparent([14])],[14:[bind(X,$fot(b))]])).

fof(16,plain,~big_s(b),inference(and_F,[and_F(split,[position(lrl)]),pparent([15])],[15])).
fof(17,plain,(big_q(b)|big_r(b))=>big_s(b),
       inference(forall_T,[status(thm),pparent([16])],[5:[bind(X,$fot(b))]])).
fof(18,plain,(big_q(b)|big_r(b))=>$false,
       inference(implies_T,[implies_T(split,[position(lrll)]),pparent([17])],[17]).
fof(19,plain,~big_q(b),inference(or_F,[status(thm),pparent([18])],[18])).
fof(20,plain,$false,inference(axiom,[status(thm),pparent([19])],[13,19])).
fof(21,plain,big_s(b),inference(implies_T,[implies_T(split,[position(lrlr)]),pparent([17])],[17])).
fof(22,plain,$false,inference(axiom,[status(thm),pparent([21])],[16,21])).

fof(23,plain,~big_q(b),inference(and_F,[and_F(split,[position(lrr)]),pparent([15])],[15])).
fof(24,plain,$false,inference(axiom,[status(thm),pparent([23])],[13,23])).
fof(25,plain,~big_r(a),inference(and_F,[and_F(split,[position(r)]),pparent([6])],[6])).
%----------------------------------------------------------------------------------------------------
```

Figure 5: A derivation for `SYN054+1` in the tableau calculus using the TPTP syntax

Even though a derivation in the tableau calculus is still an acyclic directed graph, its structure is different from the structure of a derivation in the resolution calculus. Hence the proof presentation using the TPTP language needs to be adapted. For a tableau, in addition to listing the logical parents of each formula in the parents list, the *physical* parent of each node is recorded in a `pparent()` term in the inference information list. The branching of the tableau is recorded in the same way as splitting inferences are recorded in CNF refutations [25, 21]. The *inference_name* is `axiom` or the name of the applied inference rule, i.e., `and_T`, `or_F`, `implies_F`, `neg_T`, `neg_F`, `and_F`, `or_T`, `implies_T`, `forall_T`, `exists_F`, `forall_F`, or `exists_T`. A formula of the form A^F is represented in the TPTP language by the formula `A=>$false`, if A is a non-atomic formula; otherwise, if A is an atomic formula, it is represented by `~`A. A formula of the form A^T is represented by A.

Example 4. *The derivation of Figure 4 is shown using the TPTP syntax in Figure 5. The non-negated original conjecture is added as node 0. Observe the fact that the physical parent might differ from the logical parent. For example, the physical parent of node 8 is node 7, its formula is* $(\neg(\exists x P x))^F$*, which is obtained by an* implies_T *inference, whose premise is the formula of node 4.*

The proof representation is independent from the specific proof *search* (algorithm). Hence, the proposed format can also be used to represent derivations obtained by, e.g., free-variable tableaux or a proof search using iterative deepening.

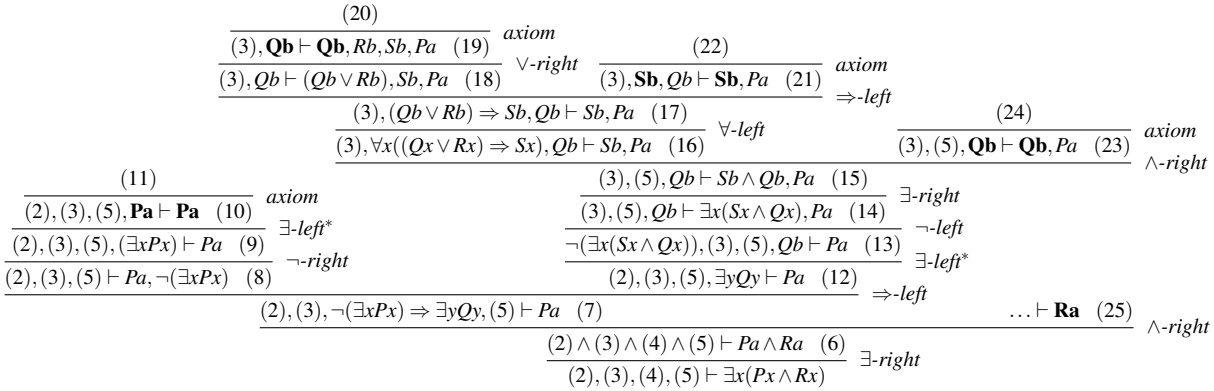


Figure 6: A derivation for SYN054+1 in the sequent calculus

Representing Derivations in the Sequent Calculus. The tableau calculus is closely related to the sequent calculus [5]. Therefore, the TPTP format for tableau derivations can also be used for representing derivations in standard sequent calculi [20]. Formulae of the form A^T occur (only) on the left side of the sequents (the *antecedent*), formulae of the form A^F occur (only) on the right side (the *succedent*). Each inference rule $rule^T$ and $rule^F$ in the tableau calculus corresponds to exactly one rule *rule-left* and *rule-right* in the sequent calculus, respectively.

Example 5. *Figure 6 shows the the sequent derivation that corresponds to the tableau derivation given in Figure 4. However, as the Eigenvariable condition needs to be respected (for the ∃-left* rule), it might be necessary to reorder some inference rules in order to obtain a correct sequent proof.*

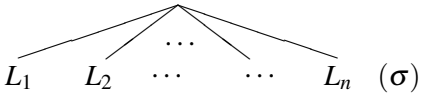
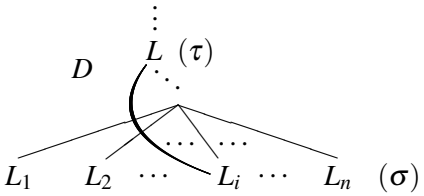
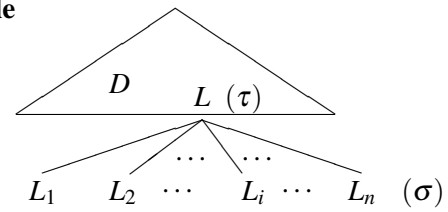
4 Representing Derivations in the Connection Calculus

Connection calculi, e.g. the connection method [2], the connection tableau calculus [8] and the model elimination calculus [9], are established proof search calculi. In principle, derivations in the clausal connection calculus can be seen as derivations in the tableau calculus with a connectedness condition [6]. But to this end many additional inferences need to be inserted. Hence it is advantageous to have a different representation of derivations, in which each inference in the connection calculus is related to exactly one inference in the representation (using the TPTP language).

The main concept of connection calculi is the guidance of the proof search by connections. A *connection* is a set of literals with opposite polarity but identical atomic formulae, i.e., $\{L_1, L_2\}$ is a connection if, and only if, $L_1 = \neg L_2$ or $\neg L_1 = L_2$. The connection calculus has three main inference rules: *start*, *reduction*, and *extension* rule. These rules are depicted in Table 2. For details see [2, 8, 11]. A formula F in disjunctive (conjunctive) clause normal form is valid (unsatisfiable) if, and only if, there is a derivation in the (clausal) connection calculus such that every literal is an element of at least one connection.

Example 6. *A derivation of problem* SYN054+1 *from Example 1 in the clausal connection calculus is shown in Figure 7 and Figure 8. Again, each inference is annotated by its number, the clause number used in the inference, and a substitution. The inferences with numbers 5 and 10 are applications of reduction rules. The branch containing the circled literal, i.e., inference number 7, is closed by an application of the lemma rule (see [11] for details). The derivation is a proof as every literal is element of a connection, hence all branches are closed by at least one connection. The major left branch in Figure 7 is a compact representation of the derivation shown in Figure 4, containing only the bold literals of Figure 4.*

Table 2: The rules of the (clausal) connection calculus

Start rule		is a derivation for a clause $C = \{L_1, \ldots, L_n\}$ and a (term) substitution σ.
Reduction rule		is a derivation, if D (without the thick line) is a derivation, L_i is a (leaf) literal not element of a connection, L is a literal on the path from L_i to the root, and $\{\tau(L), \sigma(L_i)\}$ is a connection.
Extension rule		is a derivation for a clause $C = \{L_1, \ldots, L_n\}$ and a substitution σ, if D is a derivation, L is a (leaf) literal not element of a connection, and $\{\tau(L), \sigma(L_i)\}$ is a connection for some i.

A derivation in the clausal connection calculus using the TPTP language is a list of clausal annotated formulae, as described in Section 2.2. Similar to a tableau, the *parents* is an ordered list in which the first element is the name of the physical parent of the node, and the following element is the name of the logical parent, i.e., the clause of the inference. Again, variable bindings are captured in `bind`/2 terms. Additionally, the number of the selected literal of the physical parent is captured in a `cnf_selected`/1 term. Optionally, such a term can be assigned to the logical parent as well, which would make it easier to identify the connection. The *inference_name* is the name of the applied inference rule, i.e., either `start`, `reduction`, `extension`, or `lemma`.

Example 7. *The derivation of Figure 7 is shown using this TPTP syntax in Figure 9. The output is produced by the most recent version of the* leanCoP *system [12, 10].*

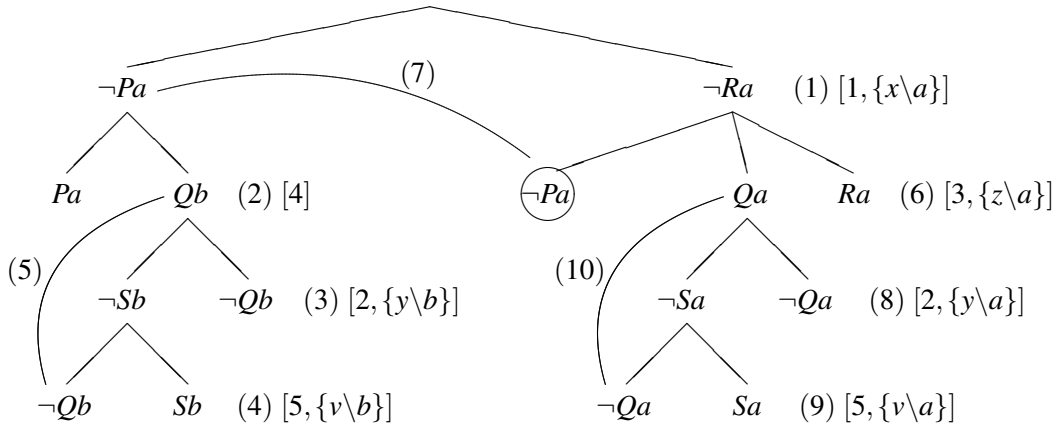


Figure 7: A derivation for `SYN054+1` in the connection calculus (tableau representation)

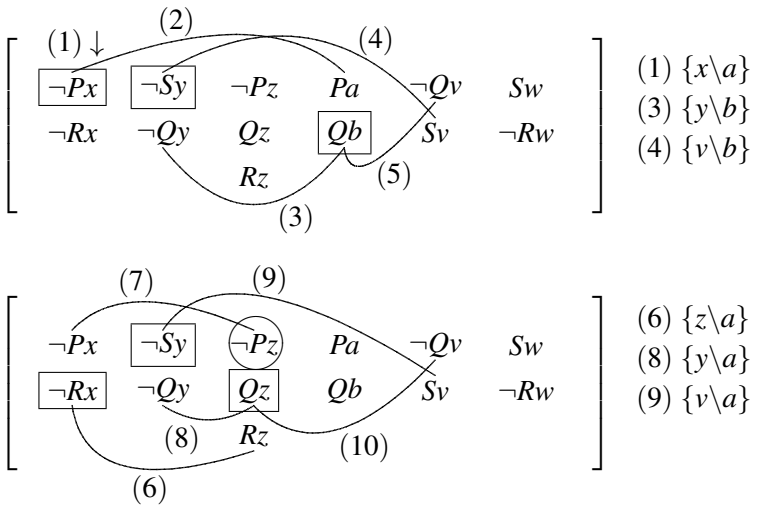


Figure 8: A derivation for SYN054+1 in the connection calculus (matrix representation)

```
%--------------------------------------------------------------------------------------------------
fof(pel24_1,axiom,~ ?[X]:(big_s(X)&big_q(X)),file('SYN054+1.p',pel24_1)).
fof(pel24_2,axiom,![X]:(big_p(X)=>(big_q(X)|big_r(X))),file('SYN054+1.p',pel24_2)).
fof(pel24_3,axiom,~ ?[X]:big_p(X)=>?[Y]:big_q(Y),file('SYN054+1.p',pel24_3)).
fof(pel24_4,axiom,![X]:((big_q(X)|big_r(X))=>big_s(X)),file('SYN054+1.p',pel24_4)).
fof(pel24,conjecture,?[X]:(big_p(X)&big_r(X)),file('SYN054+1.p',pel24)).

fof(f0,negated_conjecture,~ ?[X]:(big_p(X)&big_r(X)),
       inference(negate_conjecture,[status(cth)],[pel24])).
cnf(c1,plain,(~big_p(X)|~big_r(X)),inference(clausify,[status(esa)],[f0])).
cnf(c2,plain,(~big_s(Y)|~big_q(Y)),inference(clausify,[status(esa)],[pel24_1])).
cnf(c3,plain,(~big_p(Z)|big_q(Z)|big_r(Z)),inference(clausify,[status(esa)],[pel24_2])).
cnf(c4,plain,(big_p(a)|big_q(b)),inference(clausify,[status(esa)],[pel24_3])).
cnf(c5,plain,(~big_q(V)|big_s(V)),inference(clausify,[status(esa)],[pel24_4])).

cnf(1,plain,(~big_p(a)|~big_r(a)),
      inference(start,[status(thm)],[c1:[bind(X,$fot(a))]])).
cnf(2,plain,(big_p(a)|big_q(b)),
      inference(extension,[status(thm),pparent([1:[cnf_select([1])]])],[c4])).
cnf(3,plain,(~big_s(b)|~big_q(b)),
      inference(extension,[status(thm),pparent([2:[cnf_select([2])]])],[c2:[bind(Y,$fot(b))]])).
cnf(4,plain,(~big_q(b)|big_s(b)),
      inference(extension,[status(thm),pparent([3:[cnf_select([1])]])],[c5:[bind(V,$fot(b))]])).
cnf(5,plain,$false,
      inference(reduction,[pparent([4:[cnf_select([1])]])],[2])).
cnf(6,plain,(~big_p(a)|big_q(a)|big_r(a)),
      inference(extension,[status(thm).pparent([1:[cnf_select([2])]])],[c3:[bind(Z,$fot(a))]])).
cnf(7,plain,$false,
      inference(lemma,[pparent([6:[cnf_select([1])]])],[1])).
cnf(8,plain,(~big_s(a)|~big_q(a)),
      inference(extension,[status(thm),pparent([6:[cnf_select([2])]])],[c2:[bind(Y,$fot(a))]])).
cnf(9,plain,(~big_q(a)|big_s(a)),
      inference(extension,[status(thm),pparent([8:[cnf_select([1])]])],[c5:[bind(V,$fot(a))]])).
cnf(10,plain,$false,
      inference(reduction,[pparent([9:[cnf_select([1])]])],[6])).
%--------------------------------------------------------------------------------------------------
```

Figure 9: A derivation for SYN054+1 in the connection calculus using the TPTP syntax

5 Conclusion

A proposal for representing standard tableau, sequent and connection calculi in the TPTP language has been presented. Even though derivations in these calculi differ significantly from those in the resolution calculus, the existing TPTP syntax is flexible enough to represent derivations in these calculi as well. A common standard for presentation of derivations and proofs will increase the interoperability between ATP systems, ATP tools, and application software (see, e.g., [19]).

Future work includes the development of tools to translate connection proofs into sequent proofs, which are often used in interactive proof editors, such as Coq [1], NuPRL [3] or PVS [13].

A possible extension of the current work includes the use of the TPTP language to represent derivations in tableau and connection calculi for non-classical logics, e.g., intuitionistic and modal logics [7, 27]. These calculi often use additional annotations, e.g., a prefix that is assigned to each formula. The TPTP syntax might need to be carefully extended in order to allow the presentation of derivations in these calculi as well.

References

[1] Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[2] W. Bibel. *Automated Theorem Proving*. Vieweg and Sohn, 1987.

[3] R. Constable, S. Allen, H. Bromly, W. Cleaveland, J. Cremer, R. Harper, D. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. Sasaki, and S. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.

[4] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.

[5] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 36:176–210, 405–431, 1935.

[6] R. Hähnle. Tableaux and Related Methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier Science, 2001.

[7] C. Kreitz and J. Otten. Connection-based Theorem Proving in Classical and Non-classical Logics. *Journal of Universal Computer Science*, 5:88–112, 1999.

[8] R. Letz and G. Stenz. Model Elimination and Connection Tableau Procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 2015–2114. Elsevier Science, 2001.

[9] D.W. Loveland. Mechanical Theorem Proving by Model Elimination. *Journal of the ACM*, 15(2):236–251, 1968.

[10] J. Otten. leanCoP 2.0 and ileancop 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 283–291, 2008.

[11] J. Otten. Restricting Backtracking in Connection Calculi. *AI Communications*, 23(2-3):159–182, 2010.

[12] J. Otten and W. Bibel. leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation*, 36(1-2):139–161, 2003.

[13] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 411–414. Springer-Verlag, 1996.

[14] F.J. Pelletier. Seventy-five Problems for Testing Automatic Theorem Provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

[15] T. Raths, J. Otten, and C. Kreitz. The ILTP Problem Library for Intuitionistic Logic - Release v1.1. *Journal of Automated Reasoning*, 38(1-2):261–271, 2007.

[16] J.A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[17] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[18] J. Schumann. *Automated Theorem Proving in Software Engineering*. Springer-Verlag, 2002.

[19] J. Siekmann, C. Benzmüller, and S. Autexier. Computer Supported Mathematics with OMEGA. *Journal of Applied Logic*, 4(4):533–559, 2006.

[20] R.M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

[21] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[22] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

[23] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[24] G. Sutcliffe. The CADE-22 Automated Theorem Proving System Competition - CASC-22. *AI Communications*, 23(1):47–60, 2010.

[25] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81, 2006.

[26] S. Trac, Y. Puzis, and G. Sutcliffe. An Interactive Derivation Viewer. In S. Autexier and C. Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.

[27] A. Waaler. Connections in Nonclassical Logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1487–1578. Elsevier Science, 2001.