Natural deduction in a paracomplete setting

A. E. Bolotov, V. O. Shangin[1]

abstract. In this paper we present the automated proof search technique in natural deduction paracomplete logic. Here, for some statements we do not have evidence to conclude if they are true or false, as it happens in the classical framework. As a consequence, for example, formulae of the type $p \vee \neg p$, are not valid. In this paper we formulate the natural deduction system for paracomplete logic PComp, explain its main concepts, define proof searching techniques and the searching algorithm providing examples proofs.

Keywords: paraconsistent logic, natural deduction, proof search

1 Introduction

The Natural Deduction systems for a long time have been mostly subject of teaching curriculum while the computer science community paid much more attention to other deductive methods such as for example resolution or tableaux. The recent interest in ND systems [3, 26, 22] is, to large extend, due to its potential to give an explicit construction of the proof based on some goal-directed proof search. This enables applications of ND in various areas [27, 13].

In this paper we concentrate on paracomplete logic PComp, [4], [1], and [23]. In our presentation of an ND formulation of PComp we directly follow the notation of the latter.

We, in general, follow Quine's representation of subordinate deduction [24] which, in turn, originates from Jaskowski [16] and Fitch [14].

[1]The second author is supported by the Russian Foundation for Humanities, project 13-03-00088a. The authors thank two anonymous referees for useful suggestions.

In our previous work we extended the original formulation for classical propositional logic to first-order logic [10, 11] and then to the non-classical framework of propositional intuitionistic logic [18]. Then, in [8] it was further extended to capture propositional linear-time temporal logic PLTL and in [12] the computation tree logic CTL. Subsequent works defined the proof technique for natural deduction in linear-time case [9] and tackled expressive formalisms of quantified temporal logic [5] and in [7] we have presented a natural deduction system for the paraconsistent logic PCont and relevant proof searching algorithm.

The main contribution of this paper is the definition of the natural deduction proof search technique for the paracomplete logic PComp.

The rest of the paper is organized as follows. In § 2 we describe PComp reviewing its axiomatics and semantics. In § 3 we formulate the natural deduction calculus and give an example of the construction of the proof. Subsequently, in § 4, we introduce the main proof-searching procedures, formulate the searching algorithm and show its correctness. Finally, in § 5, we provide concluding remarks and identify future work.

2 Paracomplete Logic PComp

Fixing a set *Prop* of propositions, we export the following axiomatics of PComp from [1]. The axiomatics is a subset of classical propositional logic with the characteristic PComp Axiom 18.

PComp Axiomatics

1. $A \supset (A \vee B)$
2. $A \supset (B \vee A)$
3. $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$
4. $(A \wedge B) \supset A$
5. $(A \wedge B) \supset B$
6. $(C \supset A) \supset ((C \supset B) \supset (C \supset (A \wedge B)))$
7. $A \supset (B \supset A)$
8. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$
9. $((A \supset B) \supset A) \supset A$
10. $\neg(A \vee B) \supset (\neg A \wedge \neg B)$

11. $(\neg A \wedge \neg B) \supset \neg(A \vee B)$
12. $\neg(A \wedge B) \supset (\neg A \vee \neg B)$
13. $(\neg A \vee \neg B) \supset (\neg A \wedge \neg B)$
14. $\neg(A \supset B) \supset (A \wedge \neg B)$
15. $(A \wedge \neg B) \supset \neg(A \supset B)$
16. $\neg\neg A \supset A$
17. $A \supset \neg\neg A$
18. $\neg A \supset (A \supset B)$

The only rule of inference of PComp is modus ponens: From A and $A \supset B$ infer B.

Semantics
The semantics for the above axiom system is a matrix semantics which reflects the nature of this system — truth gaps. There are three values 1, f, 0 with the designated value 1 such that $0 < f < 1$ and $A \vee B = max(A, B)$ and $A \wedge B = min(A, B)$. The matrix semantics for PComp is given below:

$\vee$	1	f	0
1	1	1	1
f	1	f	f
0	1	f	0

$\wedge$	1	f	0
1	1	f	0
f	f	f	0
0	0	0	0

$\supset$	1	f	0
1	1	f	0
f	1	1	1
0	1	1	1

p	$\neg\ p$
1	0
f	f
0	1

In this context, for example, the law of excluded middle, $p \vee \neg p$ is not a valid formula. Indeed, as it is easy to check, that when p is assigned 1 or 0, formula $p \vee \neg p$ behaves as in the classical setting, getting the designated value 1. However, when p is assigned f, the same value, f, is assigned to its negation, $\neg p$, and thus $p \vee \neg p$ is assigned the value f.

3 Natural deduction system N*PComp*

In this section we present the natural deduction system for logic PComp slightly changing its formulation given in [25]. Due to the nature of the set up of this system, we need to take care of the behaviour of logical operations. In particular, to reflect the truth values gapping, we need to guarantee that some known formulae that are dependent on this semantical property, do not hold as theorems. For example, formulae of the type $A \vee \neg A$, are not, in general, valid, for example, $p \vee \neg p$ is not a theorem. Further, some known classical derivabilities fail. Thus, for example, from $A \supset B$ we should not be able, in general, to derive $\neg A \vee B$. Similarly, the following variant of the contraposition should also fail: from $\neg A \supset B$ derive $\neg B \supset A$.

In the subsequent presentation we will utilise the following notation. Firstly, by a *literal* we understand a proposition or its negation. Let *Lit* abbreviate a set of literals. Next, when writing $\Gamma \vdash B$ we refer to the task of establishing a natural deduction derivation (defined below) of a formula B from some set of assumptions Γ. If Γ, in $\Gamma \vdash B$, is empty then the task is to prove that B is a theorem, and in this case we will simply write $\vdash B$. Finally, the abbreviation $\Gamma \models B$ stands for establishing that B is a logical consequence of a set of assumptions Γ. If Γ, in $\Gamma \models B$, is empty then the task is to show that B is a valid formula and in this case we will simply write $\models B$. Throughout the paper, symbols '$\vdash$' and '$\models$' stand for PComp-derivability and PComp-validity, respectfully.

Natural deduction calculi can be used to solve different deductive tasks. For example, we might be given any of the following tasks:

1 to find an ND derivation $\Gamma \vdash B$,

2 to find an ND proof $\vdash B$ or

3 to check the consistency of some given set of formulae.

In the first and third cases, the ND derivation would start with some given set of assumptions Γ. In the second case, i.e. when we need to establish if B is a theorem, we commence our reasoning by introducing some assumptions. As in other ND calculi, in constructing an ND derivation, we are allowed to introduce arbitrary formulae as new assumptions[2]. Consequently, any formula in a derivation is either an assumption or a formula which is obtained as a result of the application of one of the inference rules.

Now, as it is indicative for the natural deduction construction, we define two classes of rules of inference: *elimination* and *introduction* rules. Applying elimination rules we simplify formulae while applying introduction rules we aim at 'constructing' formulae, introducing new logical constants. In Figures 1-3 we define these sets of elimination and introduction rules, where prefixes '*el*' and '*in*' stand for elimination and an introduction rule, respectively.

Elimination Rules :

$$\wedge\, el_1 \quad \frac{A \wedge B}{A} \qquad\qquad \wedge\, el_2 \quad \frac{A \wedge B}{B}$$

$$\neg \wedge\ el \quad \frac{\neg(A \wedge B)}{\neg A \vee \neg B} \qquad\qquad \neg\ el \quad \frac{\neg\neg A}{A}$$

$$\neg \vee\ el_1 \quad \frac{\neg(A \vee B)}{\neg A} \qquad\qquad \neg \vee\ el_2 \quad \frac{\neg(A \vee B)}{\neg B}$$

$$\supset el \quad \frac{A \supset B, \quad A}{B} \qquad\qquad \neg \supset el_1 \quad \frac{\neg(A \supset B)}{A}$$

$$\neg \supset el_2 \quad \frac{\neg(A \supset B)}{\neg B} \qquad\qquad \vee el \quad \frac{A \vee B, [A]C, [B]C,}{C}$$

Figure 1. NPComp Elimination rules

[2]Note that for many researchers, this opportunity to introduce arbitrary formulae as assumptions has been a point of great scepticism regarding the very possibility of the automation of the proof search.

Introduction Rules :

$$\wedge\ in \quad \frac{A, \quad B}{A \wedge B} \qquad \neg\wedge\ in \quad \frac{\neg A \vee \neg B}{\neg(A \wedge B)}$$

$$\vee\ in_1 \quad \frac{A}{A \vee B} \qquad \vee\ in_2 \quad \frac{B}{A \vee B}$$

$$\neg\vee\ in \quad \frac{\neg A, \neg B}{\neg(A \vee B)} \qquad \supset\ in \quad \frac{[C] \quad B}{C \supset B}$$

$$\neg\supset in \quad \frac{A, \neg B}{\neg(A \supset B)} \qquad \neg\ in \quad \frac{B}{\neg\neg B}$$

$$\supset_p \quad \frac{[A \supset B] \quad A}{A}$$

Figure 2. NPComp Introduction rules

Characteristic PComp Rule

$$PComp_{\neg\ in} \quad \frac{A, \ \neg A}{B}$$

Figure 3. NPComp Characteristic rule

Definition 1 (Inference). An *inference* in the system NPComp is a finite non-empty sequence of formulae with the following conditions:

- each formula is an assumption or is derived from the previous ones via an NPComp-rule;
- by applying $\supset_{in}$ each formula starting from the last alive assumption up to the one which is the result of the application of this rule, is discarded from the inference;
- by applying $\vee_{el}$ each formula starting from assumption A up to formula C, inclusively, as well as each formula starting from assumption B up to formula C, inclusively, is discarded from the inference;

- by applying $\supset_p$ each formula starting from assumption $A \supset B$ up to formula A, inclusively, is discarded from the inference. This, as before, is abbreviated by enclosing relevant discharged and discarded formulae into square brackets.

Definition 2 (Proof). A proof in the system NPComp is an inference from the empty set of assumptions.

Two rules deserve attention. First, it is $PComp_{\neg\ in}$ rule which is specific for this logic and allows to derive arbitrary formulae from a contradiction. However, when applying this rule, we do not discharge any assumptions if they were part of the contradiction.

The other rule, $\supset_p$ rule, is an analogue to axiom 10 which represents Pierce law.

As an example of the ND proof let us consider the proof for axiom 10.

list_proof	annotation
1. $(p \supset q) \supset p$	assumption
2. $p \supset q$	assumption
3. p	1,2,$\supset$ el
4. p	3, $\supset_p$, [2-3]
5. $((p \supset q) \supset p) \supset p$	$\supset\ in$, 4, [1-4]

A slightly different formulation of NPComp has been shown to be sound and complete [25] where instead of $PComp_{\neg in}$ as in our current presentation, tollendo ponens rule was used called $PComp_{\vee el}$:

$$\frac{A \vee B, \quad \neg A}{B}$$

However, it is easy to show that $PComp_{\neg in}$ used in the formulation above and $PComp_{\vee el}$ are derivable in both systems, respectively. Let us show that $PComp_{\vee el}$ is derivable via $PComp_{\neg in}$.

lproof	*annotation*
1. $A \vee B$	*given*
2. $\neg A$	*given*
3. A	*assumption*
4. B	$2, 3, PComp_{\neg in}$
5. B	*assumption*
6. B	$1, 4, 5, [3-4], [5] \vee el,$

$PComp_{\neg in}$ is derivable via $PComp_{\vee el}$

lproof	*annotation*
1. A	*given*
2. $\neg A$	*given*
3. $A \vee B$	$\vee\ in, 1$
4. B	$PComp_{\vee in}, 2, 3$

Thus, since ND formulations of PComp in [25] and in the current paper are deductively equivalent the established correctness in [25] covers our current presentation:

THEOREM 1. $\Gamma \vdash_{\boldsymbol{N}PComp} A \iff \Gamma \models A$ *[25].*

4 Proof searching techniques for N*PComp*

The proof searching procedure presented in this section is based on our generic approach developed for various ND constructions — in classical and non-classical settings, see for example [9, 7], where the main components of this generic method are defined in full. The main features of this generic procedures can be summarised as follows:

- The proof search strategy is *goal-directed*, it runs over two sequences: list_proof which lists formulae in the proof and list_goals which lists goals to be reached.

- Each step of the algorithmic proof is associated with a specific goal, called *current_goal.*

- In forming a derivation, we check the reachability of the *current_goal.*

- Reaching the current goal fires an appropriate introduction rule. Note that this makes the application of introduction rules strictly determined.
- Otherwise, we continue searching for updating both sequences, list_proof and list_goals. In particular, a special place in our searching technique is devoted to the rules that explore compound formulae in the proof in searching for new goals.
- We define special marking techniques to avoid infinite loops in list_proof and list_goals during the proof search: for example, to prevent the same application of an elimination rule or updating list_goals with the same goal.

For the sake of integrity of this presentation and for the readability of the text, we introduce the proof technique for PComp below in full, defining all main concepts and presenting both the searching procedures and the formulation of the algorithm in detail.

We proceed first defining the notion of a *goal reachability* noting that there are three possible situations here. Let G_n be the current goal in list_goals $= \langle G_0, G_1, \ldots, G_n \rangle$. Let G_n^+ stand for 'G_n is reached'. Let list_proof(G_n^+) stand for a sequence of formulae satisfying the definition of an NPComp proof of G_n. Firstly, a goal G_n can be some formula B and to reach such a goal we must have a non-discarded formula in list_proof simply graphically identical to B. Secondly, a goal G_n can have a form $[A]B$ and to reach such a goal we must have a proof list_proof(B^+) such that there exists a non-discarded assumption $A \in$ list_proof and B is the last formula of list_proof. Finally, a goal can be a contradiction, i.e. we need to have in list_proof two contradictory statements, A and $\neg A$ and abbreviated as $\perp$.

Definition 3 (Current goal reachability). Current goal, G_n, $0 \leq n$, occurring in list_goals$= \langle G_0, G_1, \ldots, G_n \rangle$, is reached if

- G_n is some formula B and there is a formula $A \in$ list_proof such that A is not discarded and $A = B$ or

- G_n is of the form $[A]B$ and there is a list_proof(B^+) such that a non-discarded assumption $A \in$ list_proof and B is the last formula of list_proof.

- G_n is a contradiction and there are two contradictory statements, $A \in$ list_proof and $\neg A \in$ list_proof.

When we generate list_goals we commence it with the *initial goal*, which is either a formula to be proved from some set of assumptions or as a theorem (i.e. in the resulting proof all assumptions should be discarded), or a contradiction.

4.1 Proof-searching procedures

First of all, to simplify search, we introduce two new rules, that are derivable in NPComp:

$$\vee\supset_1 \ \frac{(A \vee B) \supset C}{A \supset C} \qquad\qquad \vee\supset_2 \ \frac{(A \vee B) \supset C}{B \supset C}$$

Now we will give the definition of an algo-derivation and overview the proof search procedures for PComp.

Definition 4 (Algo-derivation NPComp$_{\mathsf{ALG}}$). A PComp *algo-derivation*, abbreviated as NPComp$_{\mathsf{ALG}}$, is a pair (list_proof, list_goals) whose construction is determined by the searching procedure outlined below.

Searching Procedures. The formulation of the searching procedures below utilises various techniques that have been originally defined for the classical setting [10] and also formed part of the proof-search for temporal logic [9]. Additionally, we use procedures that were introduced for paraconsistent logic PCont [7] but are also useful in the PComp setting. Finally, there are novel techniques that are introduced specifically for the setting of PComp to tackle its paracomplete nature.

Procedure (1). Here we search for an applicable elimination ND-rule in order to update list_proof. If we have a formula, or several formulae, which enable an application of an elimination ND-rule, we apply this rule and update list_proof by the conclusion of this rule. Procedure 1 terminates when either the current goal is reached or there are no applicable elimination rules.

Procedure (2). We apply procedure 2 when Procedure 1 terminates but the current goal is not reached. Here we distinguish two subroutines.

Procedure (2.1). This subroutine looks at the structure of the current goal and updates list_proof and list_goals, respectively, by new goals or new assumptions. Let list_proof $= P_1, \dots P_k$ and list_goals $= G_1, \dots, G_n$, where G_n is the current goal. A new goal, G_{n+1}, is generated by applying the subroutines (2.1.1) – (2.1.9) below depending on the various possible structures of G_n: $G_n = A \wedge B | A \vee B | A \supset B | \neg(A \wedge B) | \neg(A \vee B) | \neg(A \supset B) | L | \neg\neg A$, where A, B are any formulae and $L \in \mathit{Lit}$.

$$\begin{array}{lllllll}
(2.1.1) & \Gamma & \vdash & \Delta, A \wedge B & \longrightarrow & \Gamma & \vdash & \Delta, A \wedge B, B, A \\
(2.1.2.1) & \Gamma & \vdash & \Delta, A \vee B & \longrightarrow & \Gamma & \vdash & \Delta, A \vee B, A^{\star} \\
(2.1.2.2) & \Gamma & \vdash & \Delta, A \vee B & \longrightarrow & \Gamma & \vdash & \Delta, A \vee B, B^{\star} \\
(2.1.3) & \Gamma & \vdash & \Delta, A \supset B & \longrightarrow & \Gamma, A & \vdash & \Delta, A \supset B, B \\
(2.1.4) & \Gamma & \vdash & \Delta, \neg(A \supset B) & \longrightarrow & \Gamma & \vdash & \Delta, \neg(A \supset B), A, \neg B \\
(2.1.5) & \Gamma & \vdash & \Delta, \neg(A \vee B) & \longrightarrow & \Gamma & \vdash & \Delta, \neg(A \vee B), \neg A, \neg B \\
(2.1.6) & \Gamma & \vdash & \Delta, \neg(A \wedge B) & \longrightarrow & \Gamma & \vdash & \Delta, \neg(A \wedge B), \neg A \vee \neg B \\
(2.1.7.1) & \Gamma & \vdash & \Delta, F & \longrightarrow & \Gamma & \vdash & \Delta, F, \bot \\
(2.1.7.2) & \Gamma & \vdash & \Delta, F & \longrightarrow & \Gamma, F \supset p \wedge \neg p & \vdash & \Delta, [F \supset p \wedge \neg p] F^{\star\star} \\
(2.1.8) & \Gamma & \vdash & \Delta, \neg\neg A & \longrightarrow & \Gamma & \vdash & \Delta, \neg\neg A, A \\
(2.1.9) & \Gamma & \vdash & \Delta, [A]B & \longrightarrow & \Gamma, A & \vdash & \Delta, B
\end{array}$$

$\star$ when the current goal is disjunction, we apply Procedure (2.1.2.1), if it fails, i.e. we have not reached A, the left disjunct of the desired goal $A \vee B$, this subroutine is deleted and we apply Procedure (2.1.2.2). We terminate these subroutines if they are not successful in deriving goals A or B straightforwardly, using the elimination rules.

$\star\star$ where $F \in \mathit{Lit}$ or $F = A \vee B$. Procedure (2.1.7.2) applies when Procedure (2.1.7.1) fails. Also, in Procedure (2.1.7.2) variable p should be fresh.

Marking. Applying Procedure (2.1) we mark literals and formulae of the type $A \vee B$ if we start proof by refutation. The mark means that in reaching these goals we cannot any longer apply reasoning by refutation.

Let us explain Procedure (2.1.7) which deals with an unreached goal, F, which is either a literal or $A \vee B$. When we cannot reach the current goal, F, and Procedures (2.1.1) – (2.1.4) are not applicable, we follow similar to the classical refutation. However, now, in the new setting, we deal with this situation differently. Namely, first, we look for the contradictions in the proof — Procedure (2.1.7.1). If no contradictions are found then we turn into the refutation style proof noting that 'refutation' is understood in a very specific for this logic sense. Namely, once we have assumed $F \supset (p \wedge \neg p)$, where p is fresh, we aim at achieving the goal F. If this can be done then we can always add to list_proof a proof of F from $F \supset (p \wedge \neg p)$ and F by $\supset_p$.

Procedure (2.2). If Procedure (2.1) terminates and the current goal is not reached we apply Procedure (2.2). Here we analyse compound formulae in list_proof in order to find sources for new goals. Unlike in classical case, here only two types of compound formulae in list_proof can serve as sources for new goals, namely disjunctive and implicative formulae but not of the type $A \supset \bot$. If one of these formulae is found then its structure will determine the new goal to be generated.

(2.2.1) $\Gamma, A \vee B \vdash \Delta, C \longrightarrow \Gamma \vdash \Delta, [A]C \quad \Gamma \vdash \Delta, [B]C$

(2.2.2) $\Gamma, A \supset B \vdash \Delta, C \longrightarrow \Gamma \vdash \Delta, C,\ A$

We do not go further into the details of applying both procedures which can be found in [7].

Procedure (3). This is the standard, for our technique, routine, which checks the application of Definition 3 to establish if the current goal in the sequence list_goals has been reached: when we reach the current goal, G_n, we delete G_n from the sequence list_goals and set G_{n-1} as the current goal.

Procedure (4). Procedure (4) results in finding a relevant introduction rule to be applied. Procedures (2.1.1) – (2.1.8) are associated with correspondent introduction rules. Recall that Procedure (2.1) splits a conjunctive goal and is associated with the $\wedge_{in}$ rule, i.e. given that both goals A and B by applying this rule we would obtain the desired goal $A \wedge B$. Similarly, Procedure (2.2) looks for goals A or B, etc; so the following table represents the association of the procedures with the introduction rules as follows:

Procedure (2.1.1)	$\longrightarrow \wedge_{in}$	Procedure (2.1.5)	$\longrightarrow \neg\vee_{in}$
Procedure (2.1.2.1)	$\longrightarrow \vee_{in_1}$	Procedure (2.1.6)	$\longrightarrow \neg\wedge_{in}$
Procedure (2.1.2.2)	$\longrightarrow \vee_{in_2}$	Procedure (2.1.7)	$\longrightarrow \supset\ p$
Procedure (2.1.3)	$\longrightarrow \supset_{in}$	Procedure (2.1.8)	$\longrightarrow \neg_{in}$
Procedure (2.1.4)	$\longrightarrow \neg\supset_{in}$		

As we have already noted, the specifics of our searching technique is complete determination of the application of the introduction rules. Any application of such a rule is strictly determined by the current goal in list_goals.

4.2 Proof-searching algorithm NPComp$_{\mathsf{ALG}}$

In this section we explain the proof-searching algorithm and give its pseudo-code. The components of the algorithms described below correspond to the searching procedures above.

1. **Initialisation.** The algorithm commences by setting some initial task G_0 as its initial goal, $G_0 = G$.

2. **Checking the reachability.** We apply a recursive call to check if the current goal is reached. Letting G_{cur} to abbreviate the current goal we formalise this recursive check as follows

$$\forall G_i (0 \leq i) \in \mathsf{list\_goals}((G_i = G_{cur}) \longrightarrow \mathit{Procedure}\ (3)(G_i) = \mathbf{true}).$$

Thus, Procedure (3) checks the reachability of the current goal, for example, after initialisation, we would let $G_0 = G_{cur}$ and apply Procedure $(3)(G_{cur}) = \mathbf{true}$.

3. **Searching for applicable elimination rules.** If the current goal is not reached and it is not the initial goal, we apply Procedure (1) determining an applicable elimination rule.

$(\mathit{Reached}(G_{cur}) = \mathbf{true}) AND\ (G_{cur} \neq G_0) \longrightarrow$
$\mathit{Procedure}\ (1)(\langle \mathsf{list\_proof}, \mathsf{list\_goals}\rangle) = \mathbf{true}$.

If there are no applicable elimination rules we fire Procedure (2).

4. **Analysis of the current goal and compound formulae in $\mathsf{list\_proof}$.** Update $\mathsf{list\_proof}$ and $\mathsf{list\_goals}$ depending on the structure of the current goal G_{cur} or on the search for potential sources of new goals in $\mathsf{list\_proof}$.

$(Procedure\ (1)(\mathsf{list\_proof}) = \mathbf{false}) \longrightarrow$
$Procedure\ (2)(\langle \mathsf{list\_proof}, \mathsf{list\_goals}\rangle) = \mathbf{true}$.

5. **Introduction rules.** If the current goal is reached and it is not the initial goal, we apply Procedure (4) determining an applicable introduction rule.

$(Reached(G_{cur}) = \mathbf{true})\ AND\ (G_{cur} \neq G_0)$ 5
$\longrightarrow Procedure\ (4)(\langle \mathsf{list\_proof}, \mathsf{list\_goals}\rangle) = \mathbf{true}$.

6 **Termination.**

6a. $Reached(G_{cur}) = \mathbf{true}\ AND\ G_{cur} = G_0 \longrightarrow EXIT$. If the current goal is reached and it is the initial goal, terminate, proof found.

6b. If the current goal is reached and no more elimination rules are applicable and no more compound formulae in $\mathsf{list\_proof}$ can serve as sources for new goals then terminate, no proof found.

$(Reached(G_{cur}) = \mathbf{false})\ AND$
$(Procedure\ (2)(\langle \mathsf{list\_proof}, \mathsf{list\_goals}\rangle) = \mathbf{false})\ AND$
$(Procedure\ (4)(\langle \mathsf{list\_proof}, \mathsf{list\_goals}\rangle) = \mathbf{false}) \longrightarrow EXIT$.

This search terminates if we managed to reach the initial goal, G_0, or if the current goal is not reachable (i.e. no more updates for $\mathsf{list\_proof}$ and $\mathsf{list\_goals}$ are possible). In the former case we have found the desired proof, in the latter case there is no proof and we can construct a counterexample. Let us now describe the marking technique for formulae in $\mathsf{list\_proof}$ and $\mathsf{list\_goals}$ that prevent infinite loops when we apply procedures above. Thus, we mark:

- any formula in $\mathsf{list\_proof}$ once it serves as a premise of the rules invoked in Procedure (1);
- any formula in $\mathsf{list\_proof}$ once it serves as a source of new goals in Procedure (2.2) as well as new goals themselves. This prevents looping in Procedure (2.1.2) — see $\star$ comments above;

- in some cases marks are deleted, for example, when $A \in$ list_proof served as a source of a new goal B, but then B has been reached, hence discarded. In this case we delete the mark for A enabling this formula again to serve as a source of new goals.

Now we are ready to formulate a searching algorithm.

Algorithm NPComp$_{\mathsf{ALG}}$. list_proof = list_goals = $\emptyset$. Given a task $\Gamma \vdash G$,

```
(1) $G_{cur} = G$.
    $(\Gamma \neq \emptyset) \longrightarrow$ (list_proof $= \Gamma$, list_goals $= G$, *go to* (2)) else
    list_goals $= G$, *go to* (2).
(2) $Procedure(3)(G_{cur}) = \mathbf{true}$.
    $\forall G_i (0 \leq i) \in$ list_goals $((G_i = G_{cur}) \longrightarrow (Procedure\ (3)(G_i) = \mathbf{true}))$.
        (2a) $Reached(G_{cur}) \longrightarrow$ *go to* (3) else
        (2b) *go to* (4).
(3) $Procedure(4)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$.
    (3a) $(Reached(G_{cur}) = \mathbf{true}) AND\ (G_{cur} = G) \longrightarrow$
         *go to* $(6a)$ else
    (3b) $Procedure\ (4)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$, *go to* 2.
(4) $Procedure\ (1)(\langle$list_proof$\rangle) = \mathbf{true}$.
    (4a) Elimination rule is applicable, *go to* (2) else
    (4b) (if there are no compound formulae in list_proof to which
         an elimination rule can be applied), *go to* (5).
(5) $Procedure\ (2)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$.
    (5a) $Procedure\ (2.1)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$ (analysis
         of the structure of $G_{cur}$), *go to* (2) else
```

(5b) *Procedure* (2.2)($\langle$list_proof, list_goals$\rangle$) = **true** (searching for the sources of new goals in list_proof), *go to* (2) else

(5c) (if all compound formulae in list_proof are marked, i.e. have been considered as sources for new goals), *go to* (6b).

(6) Terminate(NPComp$_{\mathsf{ALG}}$).

(6a) The desired ND proof has been found. EXIT,

(6b) No ND proof has been found. EXIT.

4.3 Algo-proof examples

As an example of an algorithmic ND proof we apply NPComp$_{\mathsf{ALG}}$ to search for the proof of Pierce Law (Axiom 10).

NPComp$_{\mathsf{ALG}}$ **Example**: $((p \supset q) \supset p) \supset p$

We commence the proof with the main goal, $G_0 = ((p \supset q) \supset p) \supset p$. According to the classical search Procedure (2.1.3) its antecedent $(p \supset q) \supset p$ becomes the new assumption, and its consequent, p — the new goal, $G_1 = p$.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$

The current goal $G_1 = p$ cannot be reached so we apply Procedure (2.1.7). Firstly, by Procedure (2.1.7.1) we search for $\perp$, i.e. the presence of contradictions in the proof. This fails hence we apply Procedure (2.1.7.2) and set up the new task — to reach the current literal goal (p in our case) from goal $\perp \supset p$, where goal $\perp$ stands for some contradictory statements not occurring in the proof earlier. We chose $\perp = r \wedge \neg r$. So our new assumption at step 2 is $p \supset (r \wedge \neg r)$ and we aim to reach $G_2 = p$.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$
2. $p \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 = p$

From this moment we are in the refutation style proof understood specifically to this logic. Therefore, we are looking for new assumptions considering compound formulae in the proof. Thus, applying Procedure (2.1) we analyse $(p \supset q) \supset p$ at step 1 and set up the new goal $G_3 = p \supset q$. This is an implicative goal therefore by Procedure (2.1.3), the new assumption is p (at step 3) and the new goal $G_4 = q$.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$
2. $p \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 = p$
		$G_0, G_1, G_2, G_3 = p \supset q$
3. p	*assumption*	$G_0, G_1, G_2, G_3, G_4 = q$

The next few steps are applications of elimination rules. First, we eliminate implication from 2 and 3 deriving $r \wedge \neg r$ at step 4 and then eliminate conjunction from the latter deriving formulae at steps 5 and 6.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$
2. $p \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 = p$
		$G_0, G_1, G_2, G_3 = p \supset q$
3. p	*assumption*	$G_0, G_1, G_2, G_3, G_4 = q$
4. $r \wedge \neg r$	$2, 3, \supset$ *el*	G_0, G_1, G_2, G_3, G_4
5. r	$4, \wedge$ *el*	G_0, G_1, G_2, G_3, G_4
6. $\neg r$	$4, \wedge$ *el*	G_0, G_1, G_2, G_3, G_4

Since we are at the refutation style proof, applying elimination rules we are looking at the possibility to use $\neg\ in_2$ rule which needs two contradictory statements. Now we have them at steps 5 and 6. Note that the application of $\neg\ in_2$ allows, in general, to introduce any formula as a consequence of the contradiction. However, in our setup the application of this rule is strictly determined so we aim at deriving the current goal only. Thus, at step 7 we get q. This gives us the goal G_4. The previous goal — $G_3 = p \supset q$ and according to Procedure (2.1.3) we derive $p \supset q$ at step 8 discharging assumption

3 and formulae [3-7]. Next, from 1 and 8 we derive p by $\supset el$ which gives us the reachability of goal G_2.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$
2. $p \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 = p$
		$G_0, G_1, G_2, G_3 = p \supset q$
3. p	*assumption*	$G_0, G_1, G_2, G_3, G_4 = q$
4. $r \wedge \neg r$	$2, 3, \supset el$	G_0, G_1, G_2, G_3, G_4
5. r	$4, \wedge el$	G_0, G_1, G_2, G_3, G_4
6. $\neg r$	$4, \wedge el$	G_0, G_1, G_2, G_3, G_4
7. q	$5, 6 \neg in_2$	$G_0, G_1, G_2, G_3 \quad \vert G_4$ *reached*
8. $p \supset q$	$7 \supset in$, [3-7]	$G_0, G_1, G_2 \quad \vert G_3$ *reached*
9. p	$1, 8 \supset el$	$G_0, G_1 \quad \vert G_2$ *reached*

The current goal is $G_1 = p$. At this stage we apply $\supset_p$ which is associated with Pierce Law. The application of this rule looks as follows:

$$\frac{[p \supset (r \wedge \neg r)], \quad p}{p}\ .$$

Thus, we derive p at step 9 discharging assumption 2 and discarding formulae [2-9]. Finally, introducing implication to step 9 we derive the desired formula reaching the main goal G_0.

list_proof	*annotation*	list_goals
		$G_0 = ((p \supset q) \supset p) \supset p$
1. $(p \supset q) \supset p$	*assumption*	$G_0, G_1 = p$
2. $p \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 = p$
		$G_0, G_1, G_2, G_3 = p \supset q$
3. p	*assumption*	$G_0, G_1, G_2, G_3, G_4 = q$
4. $r \wedge \neg r$	$2, 3, \supset el$	G_0, G_1, G_2, G_3, G_4
5. r	$4, \wedge el$	G_0, G_1, G_2, G_3, G_4
6. $\neg r$	$4, \wedge el$	G_0, G_1, G_2, G_3, G_4
7. q	$5, 6 \neg in_2$	$G_0, G_1, G_2, G_3 \quad \vert G_4$ *reached*
8. $p \supset q$	$7 \supset in$, [3-7]	$G_0, G_1, G_2 \quad \vert G_3$ *reached*
9. p	$1, 8 \supset el$	$G_0, G_1 \quad \vert G_2$ *reached*
10. p	$9 \supset_p$, [2-9]	$G_0 \quad \vert G_1$ *reached*
11. $((p \supset q) \supset p) \supset p$	$10 \supset in$, [1-9]	$\vert G_0$ *reached*

4.4 Correctness

The correctness of the proof searching algorithm NPComp$_{\mathsf{ALG}}$ consists of showing that it satisfies the following three properties: termination, soundness and completeness.

THEOREM 2. NPComp$_{\mathsf{ALG}}$ *terminates for any input formula.*

PROOF. Proof of the termination of NPComp$_{\mathsf{ALG}}$ consists of showing that list_proof and list_goals are finite. This will follow from establishing that NPComp$_{\mathsf{ALG}}$ does not contain infinite loops. The marking technique ensures that there is finite number of application of elimination rules (Procedure (1)), as well as it guarantees that number of formulae introduced into list_proof and list_goals by Procedure (2) is also finite. The way how we manage disjunctive goals $A \vee B$ and $\neg(A \wedge B)$ and goals-literals also ensures the absence of infinite loops.

As mentioned above, introduction rules are completely determined by the algorithm. The reachability of the current and the type of the previous goal determines the relevant introduction rule. Also, though the specific for PComp, $Pcomp_{\neg\ in}$ rule, in general, allows to derive any formula from the contradiction, the application of this rule is strictly determined by the searching procedures, namely, by Procedure (2.1.7), hence the formula that we derived from a contradiction is always the one mentioned in list_goals. □

THEOREM 3. NPComp$_{\mathsf{ALG}}$ *is sound.*

PROOF. Note that according to its construction, list_proof in NPComp$_{\mathsf{ALG}}$, is in fact an NPComp proof. Hence, if there is an NPComp$_{\mathsf{ALG}}$ of F then its list_proof is a proof of F in the system NPComp. Therefore, as NPComp is sound then so is NPComp$_{\mathsf{ALG}}$. □

LEMMA 1. A PComp-model truth-value assignment ψ for a formula $F, \psi(F)$, is as follows:

1. $\psi(\neg\neg A)$:
1.1. If $\psi(\neg\neg A) = 1$ then $\psi(A) = 1$;
2. $\psi(A \wedge B)$:
2.1. If $\psi(A \wedge B) = 1$ then $\psi(A) = 1$ and $\psi(B) = 1$;

3. $\psi(A \vee B)$:
3.1. If $\psi(A \vee B) = 1$ then
 3.1.1 $\psi(A) = 1$; or
 3.1.2. $\psi(B) = 1$;
4. $\psi(A \supset B)$:
4.1. If $\psi(A \supset B) = 1$ then
 4.1.1 $\psi(A) = 0$; or
 4.1.2. $\psi(B) = 1$; or
 4.1.3. $\psi(A) = f$;
5. $\psi(\neg(A \wedge B))$:
5.1. If $\psi(\neg(A \wedge B)) = 1$ then $\psi(\neg A \vee \neg B) = 1$;
6. $\psi(\neg(A \vee B))$:
6.1. If $\psi(\neg(A \vee B)) = 1$ then $\psi(\neg A \wedge \neg B) = 1$;
7. $\psi(\neg(A \supset B))$:
7.1. If $\psi(\neg(A \supset B)) = 1$ then $\psi(A \wedge \neg B) = 1$.

PROOF. Proof immediately follows from the matrix definitions of PComp connectives. □

LEMMA 2. From list_proof(F^-), an exhausted and unsuccessful algo-proof for a PComp formula F, it is possible to extract a counter-model for F.

PROOF. Similar to [25] we generalise Hintikka set technique [15]. According to the construction of an algo-proof for F, if it is exhausted and non successful then the algorithm terminates with all its procedures applied and the final goal, F not reached. We show that in this case list_proof contains a set of literals from which we can build a model $M = \langle M, \psi \rangle$ such that $\psi(F) = 0$ or $\psi(F) = f$. We omit the details of the proof but will provide an example of an unsuccessful algo-proof and the corresponding counter-model in the next section.

□

THEOREM 4. *$PComp_{ALG_{ND}}$ is complete.*

PROOF. We must show that for every valid formula, A, $PComp_{ALG_{ND}}$ finds a $PComp_{ND}$ proof. This is a simple consequence (by contraposition) of Lemma 2. □

Theorems 2, 3 and 4 imply the following fundamental property of our algorithm:

THEOREM 5. *For any input formula A, the $PComp_{ALG_{ND}}$ terminates either building up a $PComp_{ND}$-proof for A or providing a counter-model.*

5 Conclusion and future work

We have presented a proof search technique in natural deduction system for paracomplete logic PComp. To the best of our knowledge, there is no other similar work on the automation of paracomplete natural deduction systems. Our developments presented in this paper form basis for the development of automated goal directed techniques for more expressive formalisms mentioned above. The feasibility of these extensions is based on the systematic nature of the approach which we follow to build natural deduction systems. When we started in our earlier works with the classical setting, we have managed to construct the rules of the natural deduction system for classical propositional [11] and first order logics [10] which enable an efficient extensions to more expressive frameworks of linear-time temporal logic [8] and branching-time logic [12] and to the setting of dynamic normative systems [6]. All these systems are still based on the classical setting which makes them only applicable to the complete and consistent specifications. Further, the developments of the ND calculus for the paraconsistent logic PCont [7] and the current paper are dealing with inconsistent and incomplete specifications, respectively. Due to the coherence of the approaches to build the natural systems above, it looks feasible to combine paraconsistent and paracomplete ND systems together which will form basis for temporal and normative extensions. These will be part of our future explorations.

Similarly, our initial proof search technique for ND in classical setting, was incorporated and further extended for the linear-time case and to large extend updated to the setting of paraconsistent and paracomplete logics. This gives us confidence in the success of further extensions of the proof search for the above combinations of logic so we will aim at constructing more expressive formalisms:

ND1 Combining paracomplete and paraconsistent frameworks.

ND2 Extending this combination to the frameworks of linear and branching-time logics, and

ND3 to the framework of deontic reasoning.

Finally, aiming at practical implementation of the ideas presented in this paper, we will target the following related developments:

- Implementation of the proof-search technique for the setting of paraconsistent and paracomplete logics;
- Application of these developments to a suitable use case;
- Study of complexity and refinements of the proof search procedure.

References

[1] *Avron A.* Natural 3-valued logics — characterization and proof theory // The Journal of Symbolic Logic. 1991. Vol. 56(1). P. 276 – 294.

[2] *Avron A.*, *Lev I.* A formula-preferential base for paraconsistent and plausible non-monotonic reasoning. // Proceedings of the Workshop on Inconsistency in Data and Knowledge (KRR-4), Int. Joint Conf. on AI (Ijcai 2001). 2001. P. 60–70.

[3] *Basin D.*, *Matthews S.*, *Viganò L.* Natural deduction for non-classical logics // Studia Logica. 1998. N. 60(1). P. 119–160.

[4] *Batens D.* Paraconsistent extensional propositional logics // Logique et Analyse. 1980. Vol. 23. P. 127–139.

[5] *Bolotov A.* Handling Periodic Properties: Deductive Verification for Quantified Temporal Logic Specifications // Fifth International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27–29 June, 2011, Jeju Island, Korea. 2011. P. 179–186.

[6] *Bolotov A.*, *Basso A.*, *Grigoriev O.* Deontic Extension of Deductive Verification of Component Model: Combining Computation Tree Logic and Deontic Logic in Natural Deduction Style Calculus // Proceedings of IICAI-2009. 2009. P. 166–185.

[7] *Bolotov A.*, *Shangin. V.* Natural Deduction System in Paraconsistent Setting: proof search for PCont // Journal of Intelligent Systems. 2012. Vol. 21. N. 1. P 1–24.

[8] *Bolotov A., Basukoski A., Grigoriev O., Shangin V.* Natural deduction calculus for linear-time temporal logic. // Joint European Conference on Artificial Intelligence (JELIA-2006). 2006. P. 56–68.

[9] *Bolotov A., Grigoriev O., Shangin V.* Automated natural deduction for propositional linear-time temporal logic // the Proceedings of the Time-2007, International Symposium on Temporal Representation and Reasoning, June, 2007.

[10] *Bolotov A., Bocharov V., Gorchakov A., Makarov V., Shangin V.* Let Computer Prove It // Logic and Computer. M.: Nauka, 2004, (In Russian).

[11] *Bolotov A., Bocharov V., Gorchakov A., Shangin V.* Automated first order natural deduction // Proceedings of the 4th Indian International Conference on Artificial Intelligence (IICAI-2009): Tumkur, 16–18 December 2009. P. 1292–1311.

[12] *Bolotov A., Grigoriev O., Shangin V.* Natural deduction calculus for computation tree logic // IEEE John Vincent Atanasoff Symposium on Modern Computing. 2006. P. 175–183.

[13] *Clarke E., Jha S., Marrero W. R.* Using state space exploration and a natural deduction style message derivation engine to verify security protocols // Proceedings of the IFIP TC2/WG 2.2, 2.3 International Conference on Programming Concepts and Methods, June 1998. P. 87–96.

[14] *Fitch F.* Symbolic Logic. NY: Roland Press, 1952.

[15] *Hintikka J.* Notes on the quantification theory // Commentationes physico-mathematicae. Societas scientiarum Fennica, 1955. Vol. 12, N. 17.

[16] *Jaskowski S.* On the rules of suppositions in formal logic // Polish Logic 1920–1939. Oxford Univ. Press, 1967. P. 232–258.

[17] *Kamide N.* Natural deduction systems for Nelson's paraconsistent logic and its neighbors // Journal of Applied Non-Classical Logics. 2005. Vol. 15 (4).

[18] *Makarov V.* Automatic theorem-proving in intuitionistic propositional logic // Modern Logic: Theory, History and Applications. Proceedings of the 5th Russian Conference. StPetersburg, 1998. (In Russian).

[19] *Middelburg C.* A Survey of Paraconsistent Logics // The Computing Research Repository (CoRR). Vol.1103.4324, 2011.

[20] *Naddeo A.* Axiomatic Framework applied to Industrial Design Problem formulated by Para-complete logics approach: the power of decoupling

on Optimization-Problem solving // Proceedings of Fourth International Conference on Axiomatic Design. 2006. P 1–8.

[21] *Nelson D.* Constructible falsity // Journal of Symbolic Logic. 1949. Vol. 14. P. 16–26.

[22] *Pfenning F.* Logical frameworks // Handbook of Automated Reasoning, J. A. Robinson and A. Voronkov eds. Elsevier, 2001. Chapter XXI, P. 1063–1147.

[23] *Popov V.* Sequence axiomatisation of simple paralogics // Logical Investigations. 2010. Issue 16. P. 205–220. (In Russian).

[24] *Quine W.* On natural deduction // Journal of Symbolic Logic. 1950. Vol. 15. P. 93–102.

[25] *Shangin V.* Natural deduction systems of some logics with truth-value gluts and truth-value gaps // Logical investigations. M.-SPb: C.G.I., 2011. P. 293–308. (in Russian).

[26] *Sieg W.*, *Byrnes J.* Normal natural deduction proofs (in classical logic) // Studia Logica. 1998. Vol. 60. P. 67–106.

[27] *Wooldridge M.* Reasoning about Rational Agents. MIT Press, 2000.