J. Intell. Syst. **Galley Proof**, 24 pages
DOI 10.1515/JISYS.2011.021

Natural Deduction System in Paraconsistent Setting: Proof Search for PCont

Alexander Bolotov and Vasilyi Shangin

Abstract. This paper continues a systematic approach to build natural deduction calculi and corresponding proof procedures for non-classical logics. Our attention is now paid to the framework of paraconsistent logics. These logics are used, in particular, for reasoning about systems where paradoxes do not lead to the 'deductive explosion', i.e., where formulae of the type 'A follows from false', for any A, are not valid. We formulate the natural deduction system for the logic PCont, explain its main concepts, define a proof searching technique and illustrate it by examples. The presentation is accompanied by demonstrating the correctness of these developments.

Keywords. Paraconsistent Logic, Natural Deduction, Proof Search.

2010 Mathematics Subject Classification. 03B53.

1 Introduction

When we speak about the reasoning tools related to modern computer systems we must take into account that these systems are complex, dynamic and heterogenous. Consider, for example, the problem of formation of heterogeneous resources into networks or into clouds. Conflicts of various types are inevitable here and very often a system functions quite well despite their present. This leads us to the necessity of equipping the system with reasoning techniques capable of coping with such conflicts. It is natural to think of a conflict as of an anomaly, some kind of a paradox, or simply of a contradiction. Classical reasoning is not appropriate here as it validates *ex falso quodlibet* the famous principle of deriving anything from a contradiction. If we obtain a specification, S, of a system with conflicts, and reason classically then S becomes trivial. Therefore, there is a need to develop deductive methods which make it possible to reason about paradoxical statements correctly, but at the same time without turning S into trivial. We will enable then

The first author is supported by an internal School of Electronics and Computer Science grant, January 2011. The second author is supported by the Russian Foundation for Humanities, project 10-03-00570a.

the system to identify, localise conflicts and to 'live with them' not violating its essential functionalities.

Classical reasoning is based on the assumptions that possible worlds cannot contain contradictions and are complete. If Prop stands for the set of propositions and W for the set of possible worlds, then the former means that for every possible world $w \in W$, and any $\alpha \in$ Prop, it is not possible that $\alpha \in w$ and $\neg\alpha \in w$, while the latter principle suggests that for every $w \in W$, and any $\alpha \in$ Prop, we require $\alpha \in w$ or $\neg\alpha \in w$. When the first principle is not required we are led to the framework of *paraconsistency*.

This paper gives a revised and extended account of the developments of the natural deduction system for a system of paraconsistent logic originally published as [7]. We concentrate on paraconsistent logic PCont, known under different abbreviations [1, 13, 16]. In our presentation of a natural deduction (abbreviated in this paper by 'ND') formulation of PCont, which we call NPCont, we directly follow the notation of the latter.

The particular approach to build an ND calculus we are interested in is described in detail in [4]. It is a modification of Quine's representation of subordinate proof [15] developed for classical propositional and first-order logic. Recall that natural deduction calculi of this type were originally developed by Jaskowski [10]. Jaskowski-style natural deduction was improved by Fitch [8] and simplified by Quine [15].

The ND technique initially defined for classical propositional logic was extended to first-order logic (see complete descriptions on the method and relevant proof search techniques in [4, 5]) to the non-classical framework of propositional intuitionistic logic [12]. In [3] it was further extended to capture propositional linear-time temporal logic PLTL and in [6] the ND system was proposed for the computation tree logic CTL.

The paper is organised as follows. In §2 we describe PCont reviewing its axiomatics and semantics. In §3 we formulate the natural deduction calculus and give examples of the construction of the proof. Subsequently, in §4, we introduce the main proof-searching procedures, an algorithm $\text{NPCont}_{\text{ALG}}$. Examples of $\text{NPCont}_{\text{ALG}}$ proofs are in given in §5. Further, §6 is devoted to proving the correctness issues, i.e., that $\text{NPCont}_{\text{ALG}}$ terminates, is sound and complete. Finally, in §7, we provide concluding remarks and identify future work.

2 Paraconsistent Logic PCont

Fixing a set Prop of propositions, we export the following axiomatics of PCont from [16].

PCont Axiomatics.

1. $(A \supset B) \supset ((B \supset C) \supset (A \supset C))$
2. $A \supset (A \vee B)$
3. $A \supset (B \vee A)$
4. $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$
5. $(A \wedge B) \supset A$
6. $(A \wedge B) \supset B$
7. $(C \supset A) \supset ((C \supset B) \supset (C \supset (A \wedge B)))$
8. $A \supset (B \supset A)$
9. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$
10. $((A \supset B) \supset A) \supset A$
11. $\neg(A \vee B) \supset (\neg A \wedge \neg B)$
12. $(\neg A \wedge \neg B) \supset \neg(A \vee B)$
13. $\neg(A \wedge B) \supset (\neg A \vee \neg B)$
14. $(\neg A \vee \neg B) \supset (\neg A \wedge \neg B)$
15. $\neg(A \supset B) \supset (A \wedge \neg B)$
16. $(A \wedge \neg B) \supset \neg(A \supset B)$
17. $\neg\neg A \supset A$
18. $A \supset \neg\neg A$
19. $A \vee \neg A$.

Rule of inference: From A and $A \supset B$ infer B.

Semantics. The axioms of PCont are adequate to the following matrix semantics with three values 1, t, 0 and two designated values 1, t.

$\Rightarrow$	1	t	0
1	1	t	0
t	1	t	0
0	1	1	1

p	$\neg p$
1	0
t	t
0	1

$\vee$	1	t	0
1	1	1	1
t	1	t	t
0	1	t	0

$\wedge$	1	t	0
1	1	t	0
t	t	t	0
0	0	0	0

3 Natural Deduction System NPCont

Notation.

- By *literal* we understand a proposition or its negation. Let Lit abbreviate a set of literals.
- We will use the symbols '$\vdash$' and '$\models$' as follows. By writing $\Gamma \vdash B$ we mean a task to establish a natural deduction derivation of a formula B from a 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 $\vdash B$. 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 $\models B$.

Therefore, we might be given either of the following tasks: to find an ND derivation $\Gamma \vdash B$ or to find an ND proof $\vdash B$.

Specifically for an ND calculus, in constructing an ND derivation, we are allowed to introduce arbitrary formulae as new assumptions. 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.

Further, the set of rules is divided into the two classes: *elimination* and *introduction* rules. Rules of the first group allow us to simplify formulae to which they are applied. These are rules for the 'elimination' of logical constants.

Rules of the second group are aimed at 'building' formulae, introducing new logical constants. In the following we define sets of elimination and introduction rules, where the prefixes 'el' and 'in' abbreviate an elimination and an introduction rule, respectively.

Elimination Rules.

$$\wedge \mathrm{el}_1 \ \frac{A \wedge B}{A} \qquad \wedge \mathrm{el}_2 \ \frac{A \wedge B}{B}$$

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

$$\mathrm{PCont} \quad \vee\mathrm{el} \ \frac{[A]\ C, [\neg A]\ C}{C} \qquad \neg \vee \mathrm{el}_1 \ \frac{\neg(A \vee B)}{\neg A}$$

$$\neg \vee \mathrm{el}_2 \ \frac{\neg(A \vee B)}{\neg B}$$

$$\supset_{\mathrm{el}} \ \frac{A \supset B, \quad A}{B}$$

$$\neg\supset \text{el}_1\ \frac{\neg(A\supset B)}{A} \qquad \neg\supset \text{el}_2\ \frac{\neg(A\supset B)}{\neg B}$$

$$\neg\,\text{el}\ \ \frac{\neg\neg A}{A}$$

Introduction Rules.

$$\wedge\,\text{in}\ \frac{A,\quad B}{A\wedge B}$$

$$\neg\wedge \text{in}_1\ \frac{\neg A}{\neg(A\wedge B)} \qquad \neg\wedge \text{in}_2\ \frac{\neg B}{\neg(A\wedge B)}$$

$$\vee\,\text{in}_1\ \frac{A}{A\vee B} \qquad \vee\,\text{in}_2\ \frac{B}{A\vee B}$$

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

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

Definition 1 (Inference). An *inference* in the system NPCont 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 a NPCont-rule.
- By applying $\supset_{\text{in}}$ each formula starting from the last alive assumption C until B, the result of the application of this rule, inclusively, is discarded from the inference.
- By applying $\vee_{\text{el}}$ each formula starting from assumption A until formula C, inclusively, as well as each formula starting from assumption B until formula C, inclusively, is discarded from the inference.
- By applying PCont $\vee_{\text{el}}$ each formula starting from assumption A until formula C, inclusively, as well as each formula starting from assumption $\neg A$ until formula C, inclusively, is discarded from the inference.

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

Let us give three examples of ND proofs.

PCont Example 1. Proof of $p \vee \neg p$.

list_proof	annotation
1. p	assumption
2. $p \vee \neg p$	$\vee_{\text{in}_1}$, 1
3. $\neg p$	assumption
4. $p \vee \neg p$	$\vee_{\text{in}_2}$, 3
5. $p \vee \neg p$	$\text{PCont}_{V_{\text{el}}}$, 2, 4 [1–2], [3–4]

PCont Example 2. Proof of Axiom 4.

list_proof	annotation
1. $p \supset r$	assumption
2. $q \supset r$	assumption
3. $p \vee q$	assumption
4. p	assumption
5. r	$\supset_{\text{el}}$, 1, 4
6. q	assumption
7. r	$\supset_{\text{el}}$, 2, 6
8. r	$\vee$el, 3, 4, 6, [4–5], [6–7]
9. $(p \vee q) \supset r$	$\supset_{\text{in}}$, 8, [3–8]
10. $(q \supset r) \supset ((p \vee q) \supset r)$	$\supset_{\text{in}}$, 9, [2–9]
11. $(p \supset r) \supset ((q \supset r) \supset ((p \vee q) \supset r))$	$\supset_{\text{in}}$, 10, [1–10]

PCont Example 3. Proof of $((p \supset q) \supset p) \supset p$.

list_proof	annotation
1. $(p \supset q) \supset p$	assumption
2. $p \supset q$	assumption
3. p	$\supset_{\text{el}}$, 1, 2
4. $\neg(p \supset q)$	assumption
5. p	$\neg \supset_{\text{el}_1}$, 4
6. p	PCont $\vee_{\text{el}}$, 3, 5, [2–3], [4–5]
7. $((p \supset q) \supset p) \supset p$	$\supset_{\text{in}}$, 6, [1–6]

Note that the rule PCont $\vee_{\text{el}}$ makes it possible to prove the Pierce law and therefore there is no need for a specific rule associated with this formula.

It also implies an interesting (and, seemingly, new) fact that Axiom 10 (see §2) is derivable via the other PCont axioms. The proof isn't difficult. The reader should use the following instance of Axiom 4:

$$((p \supset q) \supset p) \supset ((\neg(p \supset q) \supset p) \supset (((p \supset q) \vee \neg(p \supset q)) \supset p))$$

and the standard deduction theorem. From the philosophical point of view, it shows that the positive intuitionistic fragment is already sufficient to serve as a base for PCont and, therefore, PCont is a neighbour to Nelson's paraconsistent logic [14].

NPCont has been shown to be sound and complete, i.e., the following theorem holds:

Theorem 1 ([17]). $\Gamma \vdash_{\text{NPCont}} A \Leftrightarrow \Gamma \models A$.

4 Proof Searching Techniques for NPCont

We preserve the *goal-directed* nature of the proof searching strategy creating two sequences: *list_proof* and *list_goals*. The first sequence represents formulae which form a proof. In the second sequence we keep track of the list of goals. An *algo-derivation*, ND_{alg}, as previously, is a pair (*list_proof*, *list_goals*) whose construction is determined by the searching procedure outlined below. On each step of constructing an ND_{alg}, a specific goal is chosen, which we aim to reach at the *current stage*, we call this goal a *current_goal*. The first goal of *list_goals* is extracted from the given task, we will refer to this goal as to the *initial goal*.

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 there is a formula A in *list_proof* such that A is not discarded and $A = G_n$ or if there are formulae A, B in the *list_proof* such that A is the last alive assumption, then both A, B are not discarded and $G_n = [A]B$.

When we construct a derivation, we check whether the *current_goal* has been reached. If it has been reached then we apply the appropriate introduction rule, and this is *the only reason* for the application of introduction rules. Alternatively, (if the *current_goal* has not been reached), we continue searching how to update *list_proof* and *list_goals*.

4.1 Proof-Searching Procedures

Procedure 1. This procedure updates a sequence *list_proof* by searching for an applicable elimination ND-rule. If it finds in *list_proof* a formula, or two formulae, which can serve as premises of an elimination ND-rule, this rule is enforced and the sequence *list_proof* is updated by the conclusion of this rule.

Procedure 2. This procedure is invoked when Procedure 1 terminates and the current goal is not reached.

Procedure 2.1. Here we update sequences *list_proof* and *list_goals* analysing the structure of G_n. Let $list\_proof = P_1, \ldots, P_k$ and $list\_goals = G_1, \ldots, G_n$, where G_n is the current goal. Given that G_n is not reached, then looking at its structure, we derive a new goal G_{n+1} and set the latter as the current goal. Below we identify various cases of applying sub-Procedure 2.1, where

$$G_n = A \wedge B | A \vee B | A \supset B | \neg(A \wedge B) | \neg(A \vee B) | \neg(A \supset B) | F | \neg\neg A,$$

where A, B are any formulae and $F \in$ Lit.

2.1.1	Γ	$\vdash$	$\Delta, A \wedge B$	$\longrightarrow$	Γ	$\vdash$	$\Delta, A \wedge B, B, A$
2.1.2.1	Γ	$\vdash$	$\Delta, A \vee B$	$\longrightarrow$	Γ	$\vdash$	$\Delta, A \vee B, A$ $\star$
2.1.2.2	Γ	$\vdash$	$\Delta, A \vee B$	$\longrightarrow$	Γ	$\vdash$	$\Delta, A \vee B, B$ $\star$
2.1.3	Γ	$\vdash$	$\Delta, A \supset B$	$\longrightarrow$	Γ, A	$\vdash$	$\Delta, A \supset B, B$
2.1.4	Γ	$\vdash$	$\Delta, \neg(A \supset B)$	$\longrightarrow$	Γ	$\vdash$	$\Delta, \neg(A \supset B), A, \neg B$
2.1.5	Γ	$\vdash$	$\Delta, \neg(A \vee B)$	$\longrightarrow$	Γ	$\vdash$	$\Delta, \neg(A \vee B), \neg A, \neg B$
2.1.6.1	Γ	$\vdash$	$\Delta, \neg(A \wedge B)$	$\longrightarrow$	Γ	$\vdash$	$\Delta, \neg(A \wedge B), \neg A$ $\star$
2.1.6.2	Γ	$\vdash$	$\Delta, \neg(A \wedge B)$	$\longrightarrow$	Γ	$\vdash$	$\Delta, \neg(A \wedge B), \neg B$ $\star$
2.1.7	Γ	$\vdash$	Δ, F	$\longrightarrow$	$\Gamma, \neg F$	$\vdash$	$\Delta, [F]F, [\neg F], F$ $\star\star$
2.1.8	Γ	$\vdash$	$\Delta, \neg\neg A$	$\longrightarrow$	Γ	$\vdash$	$\Delta, \neg\neg A, A$
2.1.9	Γ	$\vdash$	$\Delta, [A]B$	$\longrightarrow$	Γ, A	$\vdash$	Δ, B

$\star$ searching rule 2.1.2.2 applies when rule 2.1.2.1 fails, i.e., when applying rule 2.1.2.1, we have not managed to reach A, the left disjunct of the goal $A \vee B$, in which case the subroutine invoked into this attempt is deleted and rule 2.1.2.2 is fired. In both cases we require to terminate the subroutine if it fails to derive a goal A or B straightforwardly using the elimination rules.

Similarly, searching rule 2.1.6.2 applies when rule 2.1.6.1 fails, i.e., when applying rule 2.1.6.1, we have not managed to reach $\neg A$, in which case the

subroutine invoked into this attempt is deleted and rule 2.1.6.2 is fired. In both cases we require to terminate the subroutine if it fails to derive a goal $\neg A$ or $\neg B$ straightforwardly using the elimination rules.

$\star\star$ where $F \in \text{Lit}$ or $F = A \vee B$ or $F = \neg(A \wedge B)$.

Applying Procedure 2.1 we mark literals and formulae of the type $A \vee B$ and $\neg(A \wedge B)$ 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 $A \vee B$ or $\neg(A \wedge B)$. When we cannot 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 setting of paraconsistent logic, we deal with this situation differently. Namely, once we assumed $\neg F$ 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. These two inferences would give us the required basis to apply PCont $\vee_{\text{el}}$ rule, namely, $[\neg F]$, F and $[F]$, F which would enable us to derive the desired F.

Procedure 2.2. This procedure is invoked when the current goal is not reached and Procedure 2.1 has been terminated. It searches for those compound formulae in *list_proof* which can serve as 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. If one of these formulae is found then its structure will determine the new goal to be generated.

$$\begin{array}{lllllllll}
2.2.1 & \Gamma, A \vee B & \vdash & \Delta, C & \longrightarrow & \Gamma & \vdash & \Delta, [A]C & \Gamma \vdash \Delta, [B]C \\
2.2.2 & \Gamma, A \supset B & \vdash & \Delta, C & \longrightarrow & \Gamma & \vdash & \Delta, [A]C & \Gamma \vdash \Delta, [\neg A]C
\end{array}$$

Let us clarify how Procedure 2.2.1 applies.

$$\begin{array}{lllllllll}
2.2.1 & \Gamma, A \vee B & \vdash & \Delta, C & \longrightarrow & \Gamma & \vdash & \Delta[B]C, [A]C & \\
2.2.1.1 & \Gamma & \vdash & \Delta, [B]C, [A]C & \longrightarrow & \Gamma & \vdash & \Delta, [B]C, [A]C, [\neg B]C & \star \\
2.2.1.2 & \Gamma & \vdash & \Delta, [B]C, [A]C & \longrightarrow & \Gamma & \vdash & \Delta, [B]C, [A]C, [B]C & \star\star \\
2.2.1.3 & \Gamma & \vdash & [B]C & \longrightarrow & \Gamma & \vdash & \Delta, [B]C, [A]C & \star\star\star \\
2.2.1.4 & \Gamma & \vdash & [B]C & \longrightarrow & \Gamma & \vdash & \Delta, [B]C, [A]C & \star\star\star\star
\end{array}$$

$\star$ when 2.2.1 fails.

$\star\star$ when 2.2.2.1 is successful, i.e., we manage to derive C from $\neg B$.

$\star\star\star$ when both 2.2.1.1 and 2.2.1.2 are successful, i.e., we have managed to derive C from A.

$\star\star\star\star$ when 2.2.1.3 is successful, i.e., we have managed to derive C from $\neg A$.

$$2.2.2. \qquad \Gamma, A \supset B, \ \vdash \ \Delta, C \ \longrightarrow \ \Gamma \ \vdash \ \Delta, [A]C, [\neg A]C$$

Let us clarify how Procedure 2.2.2 applies.

$$2.2.2.1. \qquad \Gamma \ \vdash \ \Delta, [A]C, [\neg A]C \ \longrightarrow \ \Gamma \ \vdash \ \Delta, [A]C \ \star$$

$\star$ when 2.2.2. is successful, i.e., we have managed to derive C from $\neg A$.

Procedure 3. This procedure checks reachability of the current goal in the sequence *list_goals*. If, according to Definition 3, the current goal G_n is reached then the sequence *list_goals* is updated by deleting G_n and setting G_{n-1} as the current goal.

Procedure 4. Procedure 4 indicates that one of the introduction ND-rules, i.e., a rule which introduces a logical connective, must 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 $\wedge_{\text{in}}$ rule, i.e., given 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 $\longrightarrow \wedge_{\text{in}}$	Procedure 2.1.5 $\longrightarrow \neg\vee_{\text{in}}$
Procedure 2.1.2.1 $\longrightarrow \vee_{\text{in}_1}$	Procedure 2.1.6.1 $\longrightarrow \neg\wedge_{\text{in}_1}$
Procedure 2.1.2.2 $\longrightarrow \vee_{\text{in}_2}$	Procedure 2.1.6.2 $\longrightarrow \neg\wedge_{\text{in}_2}$
Procedure 2.1.3 $\longrightarrow \supset_{\text{in}}$	Procedure 2.1.7 $\longrightarrow$ PCont $\vee_{\text{el}}$
Procedure 2.1.4 $\longrightarrow \neg\supset_{\text{in}}$	Procedure 2.1.8 $\longrightarrow \neg_{\text{in}}$

Note that although Procedure 2.1.7 invokes an elimination rule, PCont $\vee_{\text{el}}$, it has a special role in our heuristic and we consider this rule here in line with other introduction rules involved into the searching technique.

We will see below that any application of the introduction rule is completely determined by the current goal of the sequence of goals. This property of our proof searching technique protects us from inferring an infinite number of formulae in *list_proof*.

Now we are ready to describe a searching algorithm, specifying the application of the procedures above.

4.2 Proof-Searching Algorithm NPCont$_{\text{ALG}}$

Let us explain, schematically, the performance of the proof-searching algorithm by describing its major components. These components correspond to the searching procedures presented in §4.1. Given a task G, we commence the algorithm

by setting the initial goal, $G_0 = G$. Then for any goal G_i $(0 \leq i)$, we apply Procedure 3, to check if G_i is reached. If G_i is not reached we apply Procedure 1, obtaining all possible conclusions of the elimination rules to obtain G_i. If we fail, then Procedure 2 is invoked, and, dependent on the structure of the goal G_i, the sequence *list_proof* is updated by adding new assumptions and the sequence *list_goals* by adding new goals. If the current goal is reached, then we determine which introduction rules are to be applied (Procedure 4). Otherwise, we update *list_goals* looking for possible sources of new goals in *list_proof*. We continue searching until either we reach the initial goal, G_0, in which case we terminate having found the desired proof, or until *list_proof* and *list_goals* cannot be updated any further. In the latter case we terminate, and no proof has been found and a counterexample can be extracted.

Before formulating the main stages of the proof-searching algorithm we have to describe our marking technique which introduces and eliminates special marks for formulae in *list_proof* and *list_goals*. Most of these marks are devoted to prevent looping either in the application of elimination rules or in searching. Thus, we mark:

- formulae that have been used as premises of the rules invoked in Procedure 1;
- those formulae in *list_proof* which were considered as sources of new goals in Procedure 2.2 and these new goals themselves to prevent looping in Procedure 2.1.2 – see $\star$ comments to this procedure. Note that if a formula A has generated a goal B in this way, but later B has been reached, hence discarded, from the proof, we get rid of the mark for A allowing to consider this formula again as a source of new goals.

Now we are ready to formulate a searching algorithm.

Algorithm NPCont$_{\mathbf{ALG}}$.

(1) Given a task $\Gamma \vdash G$, we consider G as the initial goal of the proof and write G into *list_goals*. If the set of given assumptions in Γ is not empty then these assumptions are written in a *list_proof*. Let G_{cur} abbreviate a current goal. $G_{\text{cur}} = G$, go to (2).

(2) Apply Procedure 3 (analysis of the reachability of the current goal, G_{cur}).

(2a) If G_{cur} is reached then go to (3) else

(2b) if elimination rules are applicable go to (4) else (5).

(3) Based on the structure of the goal reached.

(3a) If G_{cur} (reached) is the initial goal then go to (6a) else

(3b) (G_{cur} is reached and it is not the initial goal) apply Procedure 4 (which invokes introduction rules), go to (2).

(4) Apply Procedure 1, go to (2).

(5) Apply Procedure 2.

(5a) Apply Procedure 2.1 (analysis of the structure of G_{cur}), go to (2) else

(5b) apply Procedure 2.2 (searching for the sources of new goals in *list_proof*), go to (2) else

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

(6) Apply Procedure 2.

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

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

5 Algo-Proof Examples

We will give three examples of ND proofs following the algorithm $\text{NPCont}_{\text{ALG}}$. The first example is an $\text{NPCont}_{\text{ALG}}$ proof of $p \vee \neg p$, previously proved without the application of the searching technique. The second example deals with Axiom 4 while the third example shows how the procedure terminates without finding a proof. An interested reader may want to compare algo-proofs Examples 1 and 2 with the 'manual' proofs of these theorems given above, PCont Example 1 and PCont Example 2.

$\text{NPCont}_{\text{ALG}}$ Example 1. $p \vee \neg p$

list_proof	annotation	*list_goals*
		$G_0 = p \vee \neg p$
		$G_1 = p \mid \neg p$

Neither p nor $\neg p$ are in the *list_proof*. We apply a proof search rule 2.5 deleting G_1 and increasing *list_goals* by new goals

$$G_{1.1} = [\neg(p \vee \neg p)]p \vee \neg p, \qquad G_{1.2} = [p \vee \neg p]p \vee \neg p.$$

When reaching goals $G_{1.1}$ and $G_{1.2}$ we do not have any priority, so in the example below we first reach $G_{1.1}$ and then $G_{1.2}$.

list_proof	annotation	*list_goals*
		$G_0 = p \vee \neg p$
1. $\neg(p \vee \neg p)$	assumption	$G_0, G_{1.1} = [p \vee \neg p]p \vee \neg p$
		$G_0, G_{1.1}, G_{1.2} = [\neg(p \vee \neg p)]p \vee \neg p$
		$G_0, G_{1.1}, G_{1.2}, G_2 = p \vert \neg p$
		$G_0, G_{1.1}, G_{1.2}, G_2 = p$
2. $\neg p$	$\neg\vee_{\text{el}_1}$, 1	
3. $\neg\neg p$	$\neg\vee_{\text{el}_2}$, 1	
4. p	$\neg$el, 3	G_2 is reached
5. $p \vee \neg p$	$\vee_{\text{in}_1}$,4	$G_{1.2}$ is reached
6. $p \vee \neg p$	assumption	$G_{1.1}$ is reached
7. $p \vee \neg p$	PCont_{el},	G_0 is reached
	5, 6, [1–5], [6]	

$\text{NPCont}_{\text{ALG}}$ Example 2. $(p \supset r) \supset ((q \supset r) \supset ((p \vee q) \supset r))$

We will see where an automated proof differs from the manual one given in the previous section. We set the initial goal as Axiom 4. As its main symbol is $\supset$ and the goal is not reached we apply Procedure 2.4 which gives us a new assumption $A \supset C$ (step 1) and a new goal $(B \supset C) \supset ((A \vee B) \supset C)$. Again, the current goal is not reached, and analysing its structure, we apply Procedure 2.4 to give us a new assumption $B \supset C$ (step 2) and a new goal $(A \vee B) \supset C$. Similar reasoning gives us step 3. Now, the current goal is C, it is not reached, hence we apply Procedure 2.5 to give us new assumption $\neg C$ at step 4. This latter technique is the one which distinguishes automated proof from the manual one given in the previous section. In the analysis, in *list_goals* we explicitly show that C must be obtained from the assumption $[\neg C]$ which should later be discarded.

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

G_3 cannot be reached, hence we try to obtain it via the PCont_{el} rule by assuming $\neg r$ and then r.

list_proof	annotation	*list_goals*
	assumption	$G_0 - G_3, G_{4.1} = [r]r, G_{4.2} = [\neg r]r$
4. $\neg r$	assumption	$G_0 - G_{4.2}, G_5 = r$

The current goal, r, isn't reached. We start analysing complex formulae from the *list_proof* and, as in the classical case, analysing complex formulae from the *list_proof* and consider the first, $p \supset r$. The specific of PCont is that we want to obtain the current goal, r, from both $\neg p$ and then p.

list_proof	annotation	*list_goals*
		$G_0 - G_5, G_{6.1} = [p]r, G_{6.2} = [\neg p]r$
5. $\neg p$	assumption	$G_0 - G_{6.2}, G_7 = r$

The current goal, r, isn't reached. So, we try to obtain it, on contrary, via the PCont_{el} rule by assuming $\neg r$ and then r.

list_proof	annotation	*list_goals*
		$G_0 - G_7, G_{8.1} = [r]r, G_{8.2} = [\neg r]r$
6. $\neg r$	assumption	$G_0 - G_{8.2}, G_9 = r$

The current goal is not reached, we pick up the second complex formula from the *list_proof* (Procedure 1 in the old draft), i.e., $q \supset r$. The specific of PCont is that we want to obtain the current goal, r, from both $\neg q$ and then q.

list_proof	annotation	*list_goals*
		$G_0 - G_9, G_{10.1} = [q]r, G_{10.2} = [\neg q]r$
7. $\neg q$	assumption	$G_0 - G_{10_2}, G_{11} = r$

The current goal, r, cannot be reached. So, we try to obtain it, via the PCont_{el} rule by assuming $\neg r$ and then r.

list_proof	annotation	*list_goals*
		$G_0 - G_{11}, G_{12.1} = [r]r, G_{12.2} = [\neg r]r$
8. $\neg r$	assumption	$G_0 - G_{12.2}, G_{13} = r$

Having, still, no luck, we pick up the third complex formula from the *list_proof* (Procedure 1 in the old draft), i.e., we take $p \vee q$. The specific of PCont is that we want to obtain the current goal, r, from both p and then q.

list_proof	annotation	*list_goals*
		$G_0 - G_{13}, G_{14.1} = [p]r, G_{14.2} = [q]r$
9. p	assumption	$G_0 - G_{14.2}, G_{15} = r$
10. r	$\Rightarrow_{\text{el}}$, 9, 1	$G_{14.2}, G_{15} = r$ are reached

By saying that $G_{14.2} = r$ and $G_{15} = p$ are reached we mean that after proving a goal r from an assumption p we immediately reach a goal to obtain a proof of r from p. At this stage the list of goals is $G_1 - G_{14.1}$. Since our goals $G_{14.1}$ and $G_{14.2}$ were coupled, and $G_{14.2}$ has been reached, leaving $G_{14.1}$ as the current goal, we update *list_goals* with $G_{15} = r$ and set it as a new current goal.

list_proof	annotation	*list_goals*
		$G_0 - G_{14.1} = [p]r, G_{15} = r$
11. q	assumption	$G_0 - G_{14.1} = [p]r, G_{15} = r$
12. r	$\Rightarrow_{\text{el}}$, 11, 2	$G_{14.1} = [p]r, G_{15} = r$ are reached
13. r	$\vee_{\text{el}}$, 3, 10, 12	$G_{12.2} = [\neg r]r$ and $G_{13} = r$ are reached
	[9–10], [11–12]	
		$G_0 - G_{11}, G_{12.1} = [r]r$
14. r	assumption	$G_1 - G_{12.1} = [r]r, G_{13} = r$

G_{13} and $G_{12.1}$ are reached (a goal A is trivially reachable if A is an assumption in the *list_proof*).

list_proof	annotation	*list_goals*
15. r	PCont_{el}, 14, 13	G_{11} and $G_{10.2}$ are reached
	[14], [8–13]	
16. q	assumption	$G_1 - G_{10.1} = [q]r, G_{11} = r$
17. r	$\Rightarrow_{\text{el}}$,16,2	$G_{11} = r$ and $G_{10.1} = [q]r$ are reached
18. r	PCont_{el}, 17, 15	$G_{8.2} = [\neg r]r$ and $G_9 = r$ are reached
	[16–17], [7–15]	
		$G_0 - G_7, G_{8.1} = [r]r$
19. r	assumption	$G_1 - G_{8.1}, G_9 = r$
		$G_9 = r$ and $G_{8.1} = [r]r$ are reached
		$G_0 - G_{6.2} = [\neg p]r, G_7 = r$

list_proof	annotation	*list_goals*
20. r	$\mathrm{PCont}_{\mathrm{el}}$, 19, 18	G_7 and $G_6.2$ are reached
	[6–18], [19]	
		$G_0 - G_5, G_{6.1} = [p]r$
21. p	assumption	$G_1 - G_{6.1}, G_7 = r$
22. r	$\Rightarrow_{\mathrm{el}}$,21,1	$G_7 = r$ and $G_{6.1} = [p]r$ are reached
		$G_0 - G_5 = r$
23. r	$\mathrm{PCont}_{\mathrm{el}}$, 20, 22	G_5 and $G_{4.2}$ are reached
	[21–22], [5–21]	
		$G_0 - G_{4.1} = [r]r$
24. r	assumption	$G_1 - G_{4.1}, G_5 = r$
25. r	$\mathrm{PCont}_{\mathrm{el}}$, 24, 23	G_5 and $G_{4.1}$ are reached
	[24], [4–23]	
		$G_0 - G_3 = r$

Now the following final steps complete the proof.

list_proof	annotation	*list_goals*
26. $(p \vee q) \supset r$	25, $\Rightarrow_{\mathrm{in}}$, [3–25]	G_0–G_2
27. $(q \supset r) \supset ((p \vee q) \supset r))$	26, $\Rightarrow_{\mathrm{in}}$, [2–26]	G_0,G_1
28. $(p \supset r) \supset ((q \supset r) \Rightarrow$		
$((p \vee q) \supset r))$	27, $\Rightarrow_{\mathrm{in}}$, [1–27]	G_0 reached

Finally we will give an example of a formula for which the proof is not found.

$\mathrm{NPCont}_{\mathrm{ALG}}$ Example 3. $(p \supset \neg q) \supset (q \supset \neg p)$.

list_proof	annotation	*list_goals*
1. $p \supset \neg q$	assumption	$q \supset \neg p$
2. q	assumption	$\neg p$
3. p	assumption	$[p] \neg p, [\neg p] \neg p$
4. $\neg q$	$\supset_{\mathrm{el}}$, 1, 3	

Goal $[p]\neg p$ cannot be reached. The algorithm terminates failing finding the proof. Let us show how from the above *list_proof* we can construct a counter-model $\langle M, \psi \rangle$ for $(p \supset \neg q) \supset (q \supset \neg p)$. Namely, considering a set of literals in

list_proof, we define ψ as follows:

- Since $p \in$ *list_proof*, we define $\psi(p) = 1$.
- Since both $q \in$ *list_proof* and $\neg q \in$ *list_proof*, we define $\psi(q) = t$.

It is easy to see that under these conditions $\psi(p \supset \neg q) = 1$ while we have $\psi(q \supset \neg p) = 0$, and hence $\psi((p \supset \neg q) \supset (q \supset \neg p)) = 0$.

6 Correctness

There are three necessary conditions that a proof search procedure for a decidable logic should have: termination, soundness and completeness. Being decidable, PCont encourages researchers to look for algorithms that effectively tell us if any given input formula is a theorem building up a desired proof or there is an assignment falsifying it, providing a counter-model. Below we will sketch proofs of all these properties of $\text{NPCont}_{\text{ALG}}$.

Theorem 2. $\text{NPCont}_{\text{ALG}}$ *terminates for any input formula.*

Proof. For the termination we need to establish that both main sequences, *list_proof* and *list_goals*, that constitute $\text{NPCont}_{\text{ALG}}$, are finite, and also that there are no loops in the searching procedure. Two observations are important here. Firstly, the marking technique guarantees the finite number of application of rules in Procedure 1, and the finite number of formulae that are introduced into *list_proof* and *list_goals* by Procedure 2. Note that our special procedures to deal with the most difficult for the natural deduction cases related to disjunctive goals, namely, with the goals of the type $A \vee B$, $\neg(A \wedge B)$ reflected in the Procedure 2.1.7, prevent us of being involved into loops.

Secondly, any application of an introduction rule is completely determined by the algorithm. Namely, if the current goal is reached we consider the previous goal and the corresponding introduction rule is fired. Thus, for example, if the current goal (reached) is A or B and the previous goal is $A \vee B$, see Procedure 2.1.2, then we apply the $\vee_{\text{in}_1}$ or $\vee_{\text{in}_2}$ rule to either A or B reaching the previous goal, $A \vee B$ by simply adding the missing component of disjunction. □

Theorem 3. $\text{NPCont}_{\text{ALG}}$ *is sound.*

Proof. Soundness of $\text{NPCont}_{\text{ALG}}$ follows immediately from the fact that *list_proof* obtained following the steps of the algorithm is a proof in the calculus NPCont. Hence, if there is an algo-proof of F then a *list_proof* of this algo-proof is a proof of F in the system NPCont. By Theorem 1, NPCont is sound. Therefore, $\text{NPCont}_{\text{ALG}}$ is sound, too. □

Lemma 1. *A* PCont*-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	$\psi(A) = 1$.
1.2.	If $\psi(\neg\neg A) = t$	then	$\psi(A) = t$.
2.	$\psi(A \wedge B)$:		
2.1.	If $\psi(A \wedge B) = 1$	then	$\psi(A) = 1$ and $\psi(B) = 1$.
2.2.	If $\psi(A \wedge B) = t$	then	
		2.2.1.	$\psi(A) = t, \psi(B) = 1$; or
		2.2.2.	$\psi(A) = t, \psi(B) = t$; or
		2.2.3.	$\psi(A) = 1, \psi(B) = t$.
3.	$\psi(A \vee B)$:		
3.1.	If $\psi(A \vee B) = 1$	then	
		3.1.1	$\psi(A) = 1$; or
		3.1.2.	$\psi(B) = 1$.
3.2.	If $\psi(A \vee B) = t$	then	
		3.2.1.	$\psi(A) = t, \psi(B) = 0$; or
		3.2.2.	$\psi(A) = t, \psi(B) = t$; or
		3.2.3.	$\psi(A) = 0, \psi(B) = t$.
4.	$\psi(A \supset B)$:		
4.1.	If $\psi(A \supset B) = 1$	then	
		4.1.1	$\psi(A) = 0$; or
		4.1.2.	$\psi(B) = 1$.
4.2.	If $\psi(A \supset B) = t$	then	
		4.2.1.	$\psi(A) = 1, (B) = t$; or
		4.2.2.	$\psi(A) = t, (B) = t$.
5.	$\psi(\neg(A \wedge B))$:		
5.1.	If $\psi(\neg(A \wedge B)) = 1$	then	$\psi(\neg A \vee \neg B) = 1$.
5.2.	If $\psi(\neg(A \wedge B)) = t$	then	$\psi(\neg A \vee \neg B) = t$.
6.	$\psi(\neg(A \vee B))$:		
6.1.	If $\psi(\neg(A \vee B)) = 1$	then	$\psi(\neg A \wedge \neg B) = 1$.
6.2.	If $\psi(\neg(A \vee B)) = t$	then	$\psi(\neg A \wedge \neg B) = t$.

7. $\psi(\neg(A \supset B))$:
7.1. If $\psi(\neg(A \supset B)) = 1$ then $\psi(A \wedge \neg B) = 1$.
7.2. If $\psi(\neg(A \supset B)) = t$ then $\psi(A \wedge \neg B) = t$.

Proof. The proof immediately follows from the matrix definitions of PCont connectives. □

Lemma 2. *From an exhausted non-successful algo-proof G for a* PCont *formula F we can extract a model which falsifies F.*

Proof. It is sufficient to show that a set G for a formula F satisfies the following properties:

1. If $\neg\neg A \in G$ then
   1.1. $A \in G$; or
   1.2. $A \in G, \neg A \in G$.
2. If $A \wedge B \in G$ then
   2.1. $A \in G, B \in G$; or
   2.2. $A \in G, \neg A \in G, B \in G$; or
   2.3. $A \in G, \neg A \in G, B \in G, \neg B \in G$; or
   2.4. $A \in G, B \in G, \neg B \in G$.
3. If $A \vee B \in G$ then
   3.1. $A \in G$; or
   3.2. $B \in G$; or
   3.3. $A \in G, \neg A \in G, \neg B \in G$; or
   3.4. $A \in G, \neg A \in G, B \in G, \neg B \in G$; or
   3.5. $\neg A \in G, B \in G, \neg B \in G$.
4. If $A \supset B \in G$ then
   4.1. $\neg A \in G$; or
   4.2. $B \in G$; or
   4.3. $A \in G, B \in G, \neg B \in G$; or
   4.4. $A \in G, \neg A \in G, B \in G, \neg B \in G$.
5. If $\neg(A \wedge B) \in G$ then $\neg A \vee \neg B \in G$.
6. If $\neg(A \vee B) \in G$ then $\neg A \wedge \neg B \in G$.
7. If $\neg(A \supset B) \in G$ then $A \in G$ and $\neg B \in G$.

We generalise Hintikka set technique [9]. In an exhausted non-successful algo-proof for F the algorithm terminates without finding a proof, having applied all its procedures and with the final goal which is not reached. We can show that in this case *list_proof* contains a set of literals sufficient to construct a model $M = \langle M, \psi \rangle$ and $\psi(F) = 0$. Let us recall that $\psi(p) = 1$ if $p \in G$ and $\neg p \notin G$; $\psi(p) = 0$ if $p \notin G$ and $\neg p \in G$; and, at last, $\psi(p) = t$ if $p \in G$ and $\neg p \in G$. Under this convention, cases 1–7 of Lemma 1 correspond to cases 1–7 below. Since all of these cases of Lemma 1 are conditional statements, it is sufficient to show that both parts of conditions hold. We will show complete reasoning for the very first situation and similar arguments will apply to all other situations considered.

Case 1. If $\neg\neg A \in G$ then $\neg_{\text{el}}$ is applied and A is derived, so $A \in G$ holds. If also $\neg A \in G$ we have both $\neg A \in G$ and $A \in G$. Therefore, $\psi(A) = t$ and $\psi(\neg\neg A) = t$, hence $\psi(\neg\neg A) = t$ implies $\psi(A) = t$ holds (case 1.2 of Lemma 1). If $\neg A \notin G$ we have $\neg A \notin G$ and $A \in G$. Therefore, $\psi(A) = 1$ and $\psi(\neg\neg A) = 1$, hence $\psi(\neg\neg A) = 1$ implies $\psi(A) = 1$ holds (case 1.1 of Lemma 1).

Case 2. If $A \wedge B \in G$ then both $\wedge_{\text{el}_1}, \wedge_{\text{el}_2}$ are applied (Procedure 2.2.1) and both A, B are derived. There are four possible situations concerning whether $\neg A \in G$ or $\neg B \in G$:

- If $\neg A \in G, \neg B \in G$ we have $A \in G, B \in G, \neg A\neg \in G, \neg B \in G$. Therefore, $\psi(A) = \psi(B) = t$ and $\psi(A \wedge B) = t$ (case 2.2.2 of Lemma 1).
- If $\neg A \in G, \neg B \notin G$ we have $A \in G, B \in G, \neg A \in G, \neg B \notin G$. Therefore, $\psi(A) = t$, $\psi(B) = 1$ and $\psi(A \wedge B) = t$ (case 2.2.1 of Lemma 1).
- If $\neg A \notin G, \neg B \in G$ we have $A \in G, \neg B \in G, \neg A \notin G, B \in G$. Therefore, $\psi(B) = t$ and $\psi(A) = 1$ (case 2.2.3 of Lemma 1).
- If $\neg A \notin G, \neg B \notin G$ we have $A \in G, B \in G, \neg A \notin G, \neg B \notin G$. Therefore, $\psi(A) = \psi(B) = \psi(A \wedge B) = 1$ (case 2.1 of Lemma 1).

Case 3. If $A \vee B \in G$ then the proof searching rule for $\vee$ in the *list_proof* is applied. According to this rule, four situations are possible. The first situation happens if we assume both A and $\neg B$. The second situation happens if we assume both A and B. It happens when we are lucky to derive C from $\neg B$. The third situation happens if we assume both B and $\neg A$. It happens when we are lucky to derive C from A. (In turn, it happens when we are lucky to derive C from $\neg B$ as well as to derive C from B.) At last, the forth situation happens if we assume B and A. It happens when we are lucky to derive C from A.

First, we assume A and $\neg B$ via the proof searching rule for $\vee$ in the *list_proof*.

There are four possible situations concerning whether $\neg A \in G$ or $B \in G$:

- If $\neg A \in G, B \in G$ we have $A \in G, B \in G, \neg A \in G, \neg B \in G$. Therefore, $\psi(A) = \psi(B) = t$ and $\psi(A \vee B) = t$ (case 3.2.2 of Lemma 1).
- If $\neg A \in G, B \notin G$ we have $A \in G, \neg A \in G, B \notin G, \neg B \in G$. Therefore, $\psi(A) = t$ and $\psi(B) = 0$ and $\psi(A \vee B) = t$ (case 3.2.1 of Lemma 1).
- If $\neg A \in G, B \in G$ we have $A \in G, \neg A \notin G, B \in G, \neg B \in G$. Therefore, $\psi(A) = 1$ and $\psi(B) = t$ and $\psi(A \vee B) = 1$ (case 3.1.1 of Lemma 1).
- If $\neg A \notin G, B \notin G$ we have $A \in G, \neg A \notin G, B \notin G, \neg B \in G$. Therefore, $\psi(A) = 1$ and $\psi(B) = 0$ and $\psi(A \vee B) = 1$ (case 3.1.1 of Lemma 1).

Second, we assume A and B via the proof searching rule for $\vee$ in the *list_proof*. There are four possible situations concerning whether $\neg A \in G$ or $\neg B \in G$:

- If $\neg A \in G, \neg B \in G$ we have $A \in G, B \in G, \neg A \in G, \neg B \in G$. Therefore, $\psi(A) = \psi(B) = t$ and $\psi(A \vee B) = t$ (case 3.2.2 of Lemma 1).
- If $\neg A \in G, \neg B \notin G$ we have $A \in G, \neg A \in G, B \in G, \neg B \notin G$. Therefore, $\psi(A) = t$ and $\psi(B) = 1$ and $\psi(A \vee B) = 1$ (case 3.1.2 of Lemma 1).
- If $\neg A \notin G, \neg B \in G$ we have $A \in G, \neg A \notin G, B \in G, \neg B \in G$. Therefore, $\psi(A) = 1$ and $\psi(B) = t$ and $\psi(A \vee B) = 1$ (case 3.1 of Lemma 1).
- If $\neg A \notin G, \neg B \notin G$ we have $A \in G, \neg A \notin G, B \in G, \neg B \notin G$. Therefore, $\psi(A) = \psi(B) = 1$ and $\psi(A \vee B) = 1$ (cases 3.1 of Lemma 1).

Third, we assume B and $\neg A$ via the proof searching rule for $\vee$ in the *list_proof*. There are four possible situations concerning whether $A \in G$ or $\neg B \in G$:

- If $A \in G, \neg B \in G$ we have $A \in G, B \in G, \neg A \in G, \neg B \in G$. Therefore, $\psi(A) = \psi(B) = t$ and $\psi(A \vee B) = t$ (case 3.2.2 of Lemma 1).
- If $A \in G, \neg B \notin G$ we have $A \in G, \neg A \in G, B \in G, \neg B \in G$. Therefore, $\psi(B) = 1$ and $\psi(A) = t$ and $\psi(A \vee B) = 1$ (case 3.2 of Lemma 1).
- If $A \notin G, \neg B \in G$ we have $A \notin G, \neg A \in G, B \in G, \neg B \in G$. Therefore, $\psi(A) = 0$ and $\psi(B) = t$ and $\psi(A \vee B) = t$ (case 3.2.3 of Lemma 1).
- If $A \notin G, \neg B \notin G$ we have $A \notin G, \neg A \in G, B \in G, \neg B \notin G$. Therefore, $\psi(A) = 0$ and $\psi(B) = 1$ and $\psi(A \vee B) = 1$ (case 3.2 of Lemma 1).

At last, forth, we assume B and A via the proof searching rule for $\vee$ in the *list_proof*. We treat it analogously to the way we assumed A and B (the second situation).

Case 4. If $A \supset B \in G$ then the proof searching rule for $\supset$ in the *list_proof* is applied as follows (Procedure 2.2.2). According to this rule, three situations are possible. The first situation happens if we have A in the *list_proof*. By an application of el, we immediately derive B. The second situation happens if we assume $\neg A$. It happens when we don't have A in the *list_proof*. The third situation happens if we assume A. It happens if we are lucky to derive C from $\neg A$.

First, we have A in the *list_proof*. By an application of $\supset_{\text{el}}$, we immediately derive B. There are four possible situations concerning whether $\neg A \in G$ or $\neg B \in G$:

- If $\neg A \in G, \neg B \in G$ we have $A \in G, B \in G, \neg A \in G, \neg B \in G$. Therefore, $\psi(A) \in \psi(B) = \psi(A \supset B) = t$ (case 4.4 of Lemma 1).
- If $\neg A \in G, \neg B \notin G$ we have $A \in G, \neg A \in G, B \in G, \neg B \notin G$. Therefore, $\psi(B) = \psi(A \supset B) = 1$ and $\psi(A) = t$ (case 4.2 of Lemma 1).
- If $\neg A \notin G, \neg B \in G$ we have $A \in G, \neg A \notin G, B \in G, \neg B \in G$. Therefore, $\psi(A \supset B) = \psi(B) = t$ and $\psi(A) = 1$ (case 4.3 of Lemma 1).
- If $\neg A \notin G, \neg B \notin G$ we have $A \in G, \neg A \notin G, B \in G, \neg B \notin G$. Therefore, $\psi(A) = \psi(B) = \psi(A \supset B) = 1$ (case 4.2 of Lemma 1).

Second, we assume $\neg A$ via the proof searching rule for $\supset$ in the *list_proof*. Without loss of generality, we can stick ourselves to the case $A \notin G$. (Otherwise, see the first situation.) There are four possible situations concerning whether $B \in G$ or $\neg B \in G$:

- If $B \in G, \neg B \in G$ we have $A \notin G, \neg A \in G, B \in G, \neg B \in G$. Therefore, $\psi(A \supset B) = 1, \psi(A) = 0$ and $\psi(B) = t$ (case 4.1 of Lemma 1).
- If $B \in G, \neg B \notin G$ we have $A \notin G, \neg A \in G, B \in G, \neg B \notin G$. Therefore, $\psi(B) = \psi(A \supset B) = 1$ and $\psi(A) = 0$ (cases 4.1–4.2 of Lemma 1).
- If $B \notin G, \neg B \in G$ we have $A \notin G, \neg A \in G, B \notin G, \neg B \in G$. Therefore, $\psi(A \supset B) = 1$ and $\psi(A) = \psi(B) = 0$ (case 4.1 of Lemma 1).
- If $B \notin G, \neg B \notin G$ we have $A \notin G, \neg A \in G, B \notin G, \neg B \notin G$. Therefore, $\psi(A \Rightarrow B) = 1$ and $\psi(A) = 0$ (case 4.1 of Lemma 1).

Third, we assume A via the proof searching rule for $\supset$ in the *list_proof*. We treat it analogously to the way we have A in the *list_proof* (the first situation). □

Theorem 4. $\text{NPCont}_{\text{ALG}}$ *is complete.*

Proof. We must show that for every valid formula A, $\text{NPCont}_{\text{ALG}}$ finds a NPCont proof. This is a simple consequence (by contraposition) of Lemma 2. □

Termination, soundness and completeness results imply the fundamental property of our algorithm reflected in the following theorem.

Theorem 5. *For any input formula A, the* $\text{NPCont}_{\text{ALG}}$ *terminates either building up a* NPCont*-proof for A or providing a counter-model.*

In §5 we have illustrated how the algorithm works with non-provable formulae.

7 Conclusion and Future Work

We have presented a proof search technique in natural deduction system for paraconsistent logic PCont and proved its correctness. To the best of our knowledge, there is no other similar work.

While our proof-searching technique preserves many of the strategies developed earlier it also introduces new methods that are specific for PCont. Our next task is to adapt the proposed technique to Avron's paracomplete logic as well as to Nelson's paraconsistent logic [11, 14] and the other neighbours of the former.

Bibliography

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

[2] D. Batens, Paraconsistent extensional propositional logics, *Logique et Analyse* **23** (1980), 127–139.

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

[4] A. Bolotov, V. Bocharov, A. Gorchakov, V. Makarov and V. Shangin, *Let Computer Prove It* (in Russian), Logic and Computer, Nauka, Moscow, 2004.

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

[6] A. Bolotov, O. Grigoriev and V. Shangin, Natural deduction calculus for computation tree logic, in: *IEEE John Vincent Atanasoff Symposium on Modern Computing*, Sophia (2006), 175–183.

[7] A. Bolotov and V. Shangin, natural deduction system in paraconsistent setting: proof search for PCont, in: *Proceedings of the 5th Indian International Conference on Artificial Intelligence* (IICAI-2011), Tumkur (2011), 630–638.

[8] F. Fitch, *Symbolic Logic*, Roland Press, New York, 1952.

[9] J. Hintikka, Notes on the quantification theory, *Commentationes Physico-Mathematicae* **12** (1955), 1–13.

[10] S. Jaskowski, On the rules of suppositions in formal logic, in: *Polish Logic 1920–1939*, Oxford University Press (1967), 232–258.

[11] N. Kamide, Natural deduction systems for Nelson's paraconsistent logic and its neighbors, *Journal of Applied Non-Classical Logics* **15** (2005), 405–435.

[12] V. Makarov, Automatic theorem-proving in intuitionistic propositional logic (in Russian), in: *Modern Logic: Theory, History and Applications*, Proceedings of the 5th Russian Conference, St. Petersburg (1998), 45–49.

[13] C. Middelburg, A survey of paraconsistent logics, *The Computing Research Repository* (CoRR), vol. 1103.4324 (2011).

[14] D. Nelson, Constructible falsity, *Journal of Symbolic Logic* **14** (1949), 16–26.

[15] W. Quine, On natural deduction, *Journal of Symbolic Logic* **15** (1950), 93–102.

[16] L. Rozonoer, On finding contradictions in formal theories (in Russian), *Automatica and Telemekhanica* **6** (1983), 113–124.

[17] V. Shangin, Natural deduction systems of some logics with truth-value gluts and truth-value gaps (in Russian), *Logical Investigation* **17** (2011), 293–308.

Received October 14, 2011.

Author information

Alexander Bolotov, University of Westminster, 115 New Cavendish Street,
London W1W 6UW, UK.
E-mail: a.bolotov@wmin.ac.uk

Vasilyi Shangin, Lomonosov Moscow State University, Moscow, 119991, Russia.
E-mail: shangin@philos.msu.ru