Paracomplete logic KI: natural deduction, its automation, complexity and applications

Bolotov, A., Kozhemiachenko, D. and Shangin, V.

Paracomplete Logic **Kl** — Natural Deduction, its Automation,Complexity and Applications

Alexander Bolotov*
University of Westminster, London, UK.
a.bolotov@westminster.ac.uk

Daniil Kozhemiachenko
Lomonosov Moscow State University, Russian Federation.
kodaniil@yandex.ru

Vasilyi Shangin†
Lomonosov Moscow State University, Russian Federation.
shangin@philos.msu.ru

Abstract

In the development of many modern software solutions where the underlying systems are complex, dynamic and heterogeneous, the significance of specification-based verification is well accepted. However, often parts of the specification may not be known. Yet reasoning based on such incomplete specifications is very desirable. Here, paracomplete logics seem to be an appropriate formal setup: opposite to Tarski's theory of truth with its principle of bivalence, in these logics a statement and its negation may be both untrue. An immediate result is that the law of excluded middle becomes invalid. In this paper we show how to apply an automatic proof searching procedure for the natural deduction formulation of the paracomplete logic **Kl** to reason about incomplete information systems. We provide an original account of complexity of natural deduction systems, which leads us closer to the efficiency of the presented proof search algorithm. Moreover, we have turned the assumptions management into an advantage by showing the applicability of the proposed technique to assume-guarantee reasoning.

The authors are grateful to the referees for their fruitful advice which greatly improved the paper.

*The first author thanks the University of Westminster for supporting his Sabbaticals in January-June 2017.

†The third author is supported by Russian Foundation for Humanities, grant 16-03-00749 *Logical-epistemic problems of knowledge representation.*

1 Introduction

1.1 Problem Setup — Reasoning with Incomplete Information

The significance of formal specification with subsequent verification in Software Engineering is well accepted. It is quite standard to classify two types of verification — the explorative approach (with model checking as its typical representative) and the deductive one. In this paper, we are interested in specification-based deductive verification. Incorporating the notation of [22], we represent the task of deductive verification, DV, of a system Sys with its specification $Spec$ by the following signature:

$$DV :: Sys \times Spec \longrightarrow B \times [Proof]$$

where the Boolean result of deductive verification based on theorem proving is either a proof that a system satisfies a given property or a demonstration that no proof can be established — $B \times [Proof]$.

Traditionally, specifications follow two general classical principles: completeness and consistency. The former assumes that a statement — ϕ — about the specification $Spec$, or its negation — $\neg\phi$ — is true. Under the latter, a member of $Spec$ — ϕ — and its negation — $\neg\phi$ — cannot be both true. As a consequence, completeness and consistency govern reasoning applied to such formal specifications — regardless whether it is model checking, or deductive reasoning, classical or not (temporal, modal, etc). Inconsistent or incomplete specifications are results of rejecting one of (or both) the principles mentioned above. Here paraconsistent and paracomplete logics come into play [1]. We strongly believe such cases are of more interest when one considers the development of modern software solutions with their underlying complex, dynamic and heterogenous systems. This definitely applies to such areas as clouds or robotics, where software systems are defined to work in a complex, dynamic and heterogeneous environment. However, our thorough research of software engineering formal methods literature has not shown many works where authors tackle incomplete specifications. Perhaps one of the main reasons for this is the lack of deductive methods for such a non-standard setting. Among few of those that address this problem are [17, 18, 29, 28, 43]. However, none of the techniques proposed in these papers, gives any account of automation, and, to our believe, they are not open to an easy way of automation. Below we identify the following cases relevant to the account of incompleteness of specifications:

(a) the problem to simplify complex software requirements in incomplete specifications,

(b) a typical integration task of various resources, which could be the problem

of forming of heterogeneous resources into networks or clouds, or component-based system engineering where components are not fully specified, or

(c) the problem of finding assumptions in assume-guarantee reasoning in the context of incomplete specifications.

We argue that reasoning following the classical principles is unsuitable when one deals with incomplete information as such reasoning validates the excluded middle (bivalence) principle[1]. Informally, it says that a truth-value of any statement is either true or false. One may also say that under the given specification, any statement is fully defined. In case this principle does not hold (i.e. some statement is not fully defined, or, in other words, we have here a truth-value gap) we are required to propose both specification languages of high level and corresponding deductive methods.

In the paper, we deal with the paracomplete logic where the law of excluded middle and some other classical laws are invalid. For example, one can not deduce $A \supset B$ from $\neg A \vee B$. We confine ourselves to the sentential reasoning, and at the moment, abstract from temporal or dynamic dimensions. We assume the language of the paracomplete logic **Kl** [1] (which is called PComp in [38] and [8]) is the language to write incomplete specifications. One must find efficient deductive techniques to deal with the reasoning which corresponds to clauses (a)–(c) above. When we choose among available formalisms and methods of deduction which use assumptions, we believe it is reasonable to take into account the following considerations.

(i) Efficient management of assumptions: tracking assumptions, making sure the assumptions occur in the proof with some reasons, not randomly, and to managing the way how assumptions occur in the proof.

(ii) Availability of automated proof searching that enables implementation.

(iii) Potential to reuse and adapt deductive techniques and proof searching for to the various kinds of formal specifications; for instance, an option to deal with incomplete or inconsistent specifications as well as with specifications which are both incomplete and inconsistent, or an option to extend our results to such richer formalisms as dynamic systems.

We argue now that natural deduction seems to be an appropriate framework if one wants to satisfy (i)–(iii).

[1] Although the principles of bivalence and excluded middle are different, in the paper we will use them as synonyms.

In the framework of automated reasoning, provers are usually based upon either resolution method, analytic tableaux or Fitch-style natural deduction (see, for example [31, 34, 27, 35, 30, 40] for provers based on classical natural deduction). Automated theorem proving in many-valued logic is usually conducted via the method of analytic tableaux, which provides a useful way of constructing counter-models to non-provable formulas. Our target is different. We are interested in a proof technique that explicitly constructs proofs. Considering automated natural deduction for the three-valued paracomplete logic, we use a Fitch-style calculus. Furthermore, in contrast to analytic tableaux we aim at developing a proof search algorithm which constructs explicit proofs for tautologies, not only counter-models for non-provable formulae.

In the rest of this introductory section we first provide some argumentation in favour of our choice of the underlying logic, **Kl**, to reason about incomplete specifications and then we will analyse possible approaches to build a desired natural deduction proof technique.

1.2 Choice of Logic — Paracomplete Logic Kl as a Many-valued Logic

The logic **Kl** was originally introduced by Avron [1]. In Avron's paper, **Kl** plays an important role in the definition of a family of paracomplete natural logics (though, Avron himself doesn't use the term 'paracomplete'; in his terminology, such logics are logics with the 'undefined' interpretation). This family includes strong Kleene's logic, logic of partial functions LPF and Łukasiewicz's 3-valued logic. In the following we highlight the importance of **Kl** in the context of these logics and explore some arguments in favour of our natural deduction presentation in comparison to Carnielli's approach to systematization of finite many-valued logics [14].

Considering strong Kleene's logic we note its famous property of not having theorems. As in our paper we want to tackle both derivations and proofs we find this logic inappropriate for our purposes. The logic of partial functions, LPF, has an additional unary connective (so to speak, another kind of negation), and for this reason we consider LPF being not in the scope of our research. Finally, Łukasiewicz's 3-valued logic lacks the deduction theorem which is crucial for our proof searching procedure, where the deduction theorem is incorporated in the form of the implication introduction rule. However, these arguments only justify our choice of logic and do not mean that proof searching procedures for these logics won't be a task for a future research that may be carried out. We note that these systems can be tackled, for example, in the spirit of [16].

It is also worth to analyse here Sette and Carnielli's weakly-intuitionistic logic I^1

[39]. Note that I^1 is both a counterpart of Sette's maximal paraconsistent logic P^1 and an extension of strong three-valued Kleene's logic $K3$. First, we observe that I^1 is different from our target logic, **Kl**, with respect to the validity of the formulae representing the law of excluded middle. In particular, $A \vee \neg A$ is invalid in **Kl** for an arbitrary A while in I^1 it is invalid for an atomic A only. Another difference lies within the matrix definitions of both implication and negation. The valuation of $A \supset B$ when $A = 1$ and $B = f$ is 'f' in the semantics of **Kl** but it is '0' in the semantics of I^1. The valuation of $\neg A$ when $A = f$ is 'f' in the semantics of **Kl** but it is '0' in the semantics of I^1. Last, not least, logics I^1 and **Kl** don't coincide in respect to their notions of theoremhood. For example, only a restricted version of $\neg\neg A \supset A$ is valid in I^1 while in **Kl** this law holds without restriction.

1.3 Choice of Deductive Approach — Natural Deduction for Kl

Considering the nature of our approach to build a natural deduction system, it is worth to compare it to [14, 13] and [1]. Carnielli's approach essentially uses the idea of signed formulae. Following this approach, a prefix of a formula used in a tableaux or natural deduction, would have been the corresponding matrix evaluation for this formula. For instance, given three values 1, T, 0 we would formulate in a sequent calculus (and with a slight adaptation, a natural deduction system) exactly three rules for each signed formula, $1 \supset$, $T \supset$, and $0 \supset$. A different approach was adapted by Avron, (see in particular [1], p. 277, footnote 2) and we follow this approach. Also note that some natural deduction system can be routinely extracted from Avron's paper, however, it would be considerably different from our natural deduction construction.

Natural deduction allows not only to establish that the proof one wants to achieve exists, but it also makes it very explicit. Both a natural deduction system for **Kl** and its proof searching (as presented in [10]) satisfy (i)–(ii). To the best of our knowledge, no other (direct) natural deduction system for **Kl** has been proposed. We believe this can be explained by the following. Both paraconsistent and paracomplete logics are likely to be analysed with some philosophical motivation and, therefore, in computer science framework the preferential methods have been Hilbert-style systems [24], analytic tableaux [12] or sequent-style calculi [20]. The only exception here is [4], where a kind of natural deduction system for a paracomplete setting is introduced. However, one can't consider such an approach as a direct method of deduction as it is based on the translation techniques to Isabelle [26]. We remind the reader that the system PCont, the dual of **Kl** (named as three-valued paraconsistent logic [1, p.278]), deals with inconsistent systems. Both a natural deduction system for PCont and its proof searching can be found in [7] and [33]. Consequently, the latter paper

together with the results of this paper, imply that our choice of natural deduction satisfies (iii).

The novelty of our paper is in the following. First, we show the way an automated natural deduction for **Kl** in [10] is applicable to reason about incomplete information systems. We also provide proofs of some statements previously announced and presented without proof in [8], thus significantly improving and expanding the latter. We present substantial conceptual and methodological considerations, introduce new technical concepts, refine and polish proofs and provide several examples. Finally, we provide an account of complexity and efficiency.

The paper is organised as follows. To make reading self-contained, §2 reviews the formulation of the natural deduction system for classical propositional logic. Next, §3 introduces the underlying logic **Kl**, its axiomatics, and natural deduction calculus, it also contains sketches of results in [10]. In §4 we discuss the complexity account. This follows by an overview of the proof searching procedure and the core algorithm in §5. We also provide a detailed example of the algorithmic proof search. The next section, §6, classifies problems to which natural deduction is applicable as a tool for deductive verification. We also present a methodology for solving some of the problems of the type (a)–(c) mentioned above and consider typical scenarios of component-based system synthesis and assume-guarantee technique. Finally, §7 contains the conclusion and the roadmap to future work.

2 Natural Deduction System for Classical Propositional Logic — $\mathbf{CPL_{ND}}$

We commence with the review of the natural deduction system for classical propositional logic, $\mathbf{CPL_{ND}}$. The natural deduction system presented below is a standard Fitch-style natural deduction system. One of the specifics of this type of natural deduction systems is that a derivation is defined in a linear format, opposite to Gentzen-style, or tree-like format. The rules of derivation are traditionally divided into elimination and introduction rules — the former allow to decompose compound formulae while the latter allow to construct compound formulae. Recall that in constructing proofs in natural deduction systems, we introduce assumptions. In some cases we need to discard alive assumptions. To indicate that a natural deduction rule with the conclusion C discards the last alive assumption, A, and all formulae $A, \dots, C^-$ (where C^- is the formula preceding C), we will use a standard abbreviation, $[A]C$.

The system $\mathbf{CPL_{ND}}$ has the following rules of derivation.

Elimination rules:

$$\wedge_{el_1}\ \frac{A \wedge B}{A},\ \wedge_{el_2}\ \frac{A \wedge B}{B},\ \neg_{el}\ \frac{\neg\neg A}{A},\ \vee_{el}\ \frac{\neg A, A \vee B}{B}, \supset_{el}\ \frac{A \supset B, A}{B}$$

Introduction rules:

$$\wedge_{in}\ \frac{A, B}{A \wedge B},\ \vee_{in_1}\ \frac{A}{A \vee B},\ \vee_{in_2}\ \frac{B}{A \vee B},\ \supset_{in}\ \frac{[A]B}{A \supset B},\ \neg_{in}\ \frac{[A]B, [A]\neg B}{\neg A}$$

Definition 1 (**CPL$_{\mathbf{ND}}$**-derivation)**.** An **CPL$_{\mathbf{ND}}$**-derivation of a formula A from a set of formulae Γ is a finite sequence of formulae, each of which is either a member of Γ (an assumption) or is derived from the previous formulae by one of the elimination or introduction rules. In case $\supset in$ or $\neg\ in$ are used, all formulae from the last alive assumption to the resulting formula should be discarded from the derivation.

Definition 2 (Proof)**.** A *proof* in the system **CPL$_{\mathbf{ND}}$** is a derivation with the empty set of alive assumptions.

Note that this and the other definitions of a derivation in natural deduction systems in the paper are 'standard' textbook ones and are sufficient for the purposes of the paper. For a more accurate definition of proof see [42].

It has been shown that **CPL$_{\mathbf{ND}}$** is sound and complete [5]. The natural deduction system for paracomplete logic **Kl** given in §3 is a modification of the **CPL$_{\mathbf{ND}}$** which reflects its characteristic features.

3 Paracomplete Logic Kl and its natural deduction calculus Kl$_{\mathbf{ND}}$

Here, to make the presentation self-contained, we define fully the logic **Kl**, its syntax and semantics, give a full set of rules of the natural deduction calculus and provide an account of its metatheoretical properties — the main results of [10].

3.1 Kl and Its Axiomatics

Kl is a propositional logic with the infinite number of propositional symbols $Prop = p, q, r, \ldots$ and the semantics assigning to each propositional symbol from $Prop$ one of the three truth-values 1 — 'true' (the designated one), 0 — 'false', and 1/2 — 'none'

such that $A \vee B = \max(A, B)$ and $A \wedge B = \min(A, B)$ The matrices for connectives are defined as follows.

$\vee$	1	1/2	0
1	1	1	1
1/2	1	1/2	1/2
0	1	1/2	0

$\wedge$	1	1/2	0
1	1	1/2	0
1/2	1/2	1/2	0
0	0	0	0

$\supset$	1	1/2	0
1	1	1/2	0
1/2	1	1	1
0	1	1	1

p	$\neg p$
1	0
1/2	1/2
0	1

It is the presence of the third truth assignment, 1/2, that makes the calculus paracomplete allowing to identify the cases of incompleteness (uncertainty, etc.) and thus allowing to consider systems with incomplete information, (see §6 for details). Often the properties and the flavour of the logic become more transparent in the axiomatic construction. For these reasons we export the axiomatic of **Kl** from [1] which is a subset of the set of axioms of classical propositional logic.

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. $\neg A \supset (A \supset B)$

The only rule of inference of **Kl** is modus ponens: from A and $A \supset B$ infer B.

Note that this axiomatics reflects the failure of the law of excluded middle so, for example, $p \vee \neg p$ is not provable in this system. We also observe that Axiom 19 is equivalent to $(B \supset \neg A) \supset ((B \supset A) \supset \neg B)$ [1, p.288].

3.2 $\mathbf{Kl_{ND}}$ — Natural Deduction Calculus for Kl.

Definition 3 ($\mathbf{Kl_{ND}}$-derivation)**.** A *derivation* in the system $\mathbf{Kl_{ND}}$ is a finite non-empty sequence of formulae where each formula is an alive assumption or is derived from the previous ones by one of the following $\mathbf{Kl_{ND}}$-rules.

Elimination rules:

$$\wedge_{el_1}\ \frac{A\wedge B}{A},\ \wedge_{el_2}\ \frac{A\wedge B}{B},\ \neg\wedge_{el}\ \frac{\neg(A\wedge B)}{\neg A\vee\neg B},\neg_{el}\ \frac{\neg\neg A}{A},$$

$$\neg\vee_{el_1}\ \frac{\neg(A\vee B)}{\neg A},\ \neg\vee_{el_2}\ \frac{\neg(A\vee B)}{\neg B},\ \supset_{el}\ \frac{A, A\supset B}{B},$$

$$\neg\supset_{el_1}\ \frac{\neg(A\supset B)}{A},\ \neg\supset_{el_2}\ \frac{\neg(A\supset B)}{\neg B},\ \vee_{el}\ \frac{A\vee B,[A]C,[B]C}{C},$$

$$\vee\supset_{el_1}\ \frac{(A\vee B)\supset C}{A\supset C},\ \vee\supset_{el_2}\ \frac{(A\vee B)\supset C}{B\supset C}.$$

Introduction rules:

$$\wedge_{in}\ \frac{A,B}{A\wedge B},\ \neg\wedge_{in}\ \frac{\neg A\vee\neg B}{\neg(A\wedge B)},\ \vee_{in_1}\ \frac{A}{A\vee B},\ \vee_{in_2}\ \frac{B}{A\vee B},$$

$$\neg\vee_{in}\ \frac{\neg A,\neg B}{\neg(A\vee B)},\ \supset_{in}\ \frac{[A]B}{A\supset B},\ \neg\supset_{in}\ \frac{A,\neg B}{\neg(A\supset B)},$$

$$\neg_{in}\ \frac{B}{\neg\neg B},\ \supset_{p}\ \frac{[A\supset B]A}{A},\ Kl_{\neg_{in}}\ \frac{A,\neg A}{B}$$

Definition 4 (Proof)**.** A *proof* in the system $\mathbf{Kl_{ND}}$ is a derivation with the empty set of alive assumptions.

Let us give now a short, but indicative, example of proof for $((p\wedge q)\vee(p\wedge r))\supset(p\wedge(q\vee r))$ in the described natural deduction calculus. Below we use the square brackets to indicate which formulae are discarded from the proof. Thus, the application of $\vee_{el}$ rule to $p\wedge(q\vee r)$ on step 12 requires to discard all formulae from the assumption $p\wedge q$ on step 2 up to $p\wedge(q\vee r)$ on step 6 and all formulae from the assumption $p\wedge r$ on step 7 up to formula $p\wedge(q\vee r)$ on step 11. Finally, applying $\supset_{in}$ rule to $p\wedge(q\vee r)$ on step 12, we obtain the desired derivation for $((p\wedge q)\vee(p\wedge r))\supset(p\wedge(q\vee r))$ discarding all formulae from the last alive assumption $(p\wedge q)\vee(p\wedge r)$ on step 1, up to the conclusion of this rule.

1. $(p \wedge q) \vee (p \wedge r)$ — assumption
   2. $p \wedge q$ — assumption
      3. p — $\wedge_{el_1}$: 2
      4. q — $\wedge_{el_2}$: 2
      5. $p \vee r$ — $\vee_{in_1}$: 4
      6. $p \wedge (q \vee r)$ — $\wedge_{in}$: 3, 5
   7. $p \wedge r$ — assumption
      8. p — $\wedge_{el_1}$: 7
      9. r — $\wedge_{el_2}$: 7
      10. $q \vee r$ — $\vee_{in_2}$: 9
      11. $p \wedge (q \vee r)$ — $\wedge_{in}$: 8, 10
   12. $p \wedge (q \vee r)$ — $\vee_{el}$: 1, 6, 11
13. $((p \wedge q) \vee (p \wedge r)) \supset (p \wedge (q \vee r))$ — $\supset_{in}$: 12

As the derivation does not have any alive assumptions it is also a proof for $((p \wedge q) \vee (p \wedge r)) \supset (p \wedge (q \vee r))$.

The presented natural deduction calculus is sound and complete, below $\models$ stands for $\mathbf{Kl_{ND}}$ logical consequence:

Theorem 1. $\Gamma \vdash_{\mathbf{Kl_{ND}}} A \iff \Gamma \models A$ [10]

Theorem 1 semantically justifies applications of derivations based on natural deduction. We argue that the natural deduction style of a proof is a powerful technique to tackle formal specification/verication software engineering problems. It is particularly important when there is an obvious need to not only establish if a desired proof exists but to also explicitly show how the proof (for some desired property) is constructed. Let us give here an informal insight into the way how the proof in natural deduction is formed. Assume we have a specification S, and would like to investigate if some statement $B \in S$ holds under some set of assumptions Γ. In this introductory case, we have a task to derive B from the specification S, given the assumptions Γ. Following the specifics of natural deduction, now, we either simplify compound formulae in the proof by elimination rules, or synthesise formulae by introduction rules. In the subsequent sections we present a proof search algorithm which guides such applications of elemination/introduction rules in an efficient manner, and give an annotated example.

4 Complexity of Natural Deduction

Convention 1. We will, according to Reckhow [41] and Pelletier [32], say that a given calculus is *natural* if it allows to use arbitrary assumptions in the proofs of theorems

and incorporates the deduction theorem as one of its rules.

It is evident then that systems $\mathbf{CPL_{ND}}$ and $\mathbf{Kl_{ND}}$ are "natural" systems.

Now we will consider three sound and complete classical propositional natural calculi, namely, $\mathbf{CPL_{ND}}$ described in §2, nested deduction Frege system and general deduction Frege system described in [11].

Definition 5 (Nested deduction Frege system — $nd\mathscr{F}$)**.** The system $nd\mathscr{F}$ is characterised by the following constraints:

- it has two rules of derivation:
  1. mp_n — $\dfrac{A \quad A \supset B}{B}$
  2. dr_n — $\dfrac{[A]B}{A \supset B}$ (where A is the last alive assumption)
- it uses a finite number of axiom schemas.

An $nd\mathscr{F}$-derivation of a formula A from a set of formulae Γ is a finite sequence of formulae, each of which is

- either a member of Γ (an assumption), or
- an instance of an axiom schema or
- is derived from previous formulae by mp_n or dr_n. In case dr_n rule is used, all formulae from the last alive assumption up to (but not including) formula A should be discarded from derivation.

We write $\Gamma \overset{nd\mathscr{F}}{\underset{n}{\vdash}} A$ if there is an $nd\mathscr{F}$-derivation of A from Γ with the length of no more than n formulae. We use here and below, in the formulation of the rules, a lower index n to indicate that these are derivations and rules in Nested deduction Frege system.

Definition 6 (General deduction Frege system — $d\mathscr{F}$)**.** Derivations in $d\mathscr{F}$ have steps presented as sequents of the form $\Gamma \mapsto A$ with Γ being a set of formulae and A being a formula. We use below, in the formulation of the rules, a lower index g to indicate that these are derivations and rules in General deduction Frege system. There are four rules of derivation in $d\mathscr{F}$:

1. $\mapsto A$, where A is an instance of an axiom schema of a consistent and complete set of axioms taken, for example, from [21].
2. $\{A\} \mapsto A$, where A is either a member of Γ or an assumption

3. mp_g — $\dfrac{\Gamma_1 \mapsto A \quad \Gamma_2 \mapsto A \supset B}{\Gamma_1 \cup \Gamma_2 \mapsto B}$

4. dr_g — $\dfrac{\Gamma \mapsto B}{\Gamma \setminus \{A\} \mapsto A \supset B}$

We define a $d\mathscr{F}$-derivation of a formula A from a set of formulae Γ as a finite sequence of sequents, each of which is obtained by one of the rules above, and the last sequent is $\Gamma \mapsto A$. We write $\Gamma|\frac{d\mathscr{F}}{n}A$ to indicate that there is a $d\mathscr{F}$-derivation of $\Gamma \mapsto A$ containing no more than n sequents.

We will now prove some theorems related to speedups (better performance) of these calculi.

Theorem 2. $\Gamma|\frac{\mathbf{CPL_{ND}}}{n}C \Rightarrow \Gamma|\frac{nd\mathscr{F}}{O(n)}C$

Proof. We prove the theorem by induction on the number of steps n of $\mathbf{CPL_{ND}}$-derivation.

The proof splits into two cases depending on how the last formula C in $\mathbf{CPL_{ND}}$-derivation was inferred.

Case 1 C is an assumption or a member of Γ. Then an $nd\mathscr{F}$-derivation consists of only one formula — C itself.

Case 2 C was derived by a rule of a derivation. We will now show that the conclusion of every $\mathbf{CPL_{ND}}$-rule can be derived from its premises in $nd\mathscr{F}$ in a constant number of steps. This is obvious in case of rules $\wedge_{in}$, $\wedge_{el_1}$, $\wedge_{el_2}$, $\vee_{in_1}$, $\vee_{in_2}$, $\supset_{el}$, and $\neg_{el}$. Next, we substitute each application of $\supset_{in}$ with dr_n and each application of $\supset_{el}$ with mp_n.

In case C was derived by $\neg_{in}$, let $C = \neg A$. We have a $\mathbf{CPL_{ND}}$-derivation of length n. We proceed as follows. We will also provide necessary comments explaining how steps of the proofs are derived.

$$
\begin{array}{l}
\vdots \\
\left[\begin{array}{l}
A \text{ — the last alive assumption} \\
\vdots \\
B \\
\vdots \\
\neg B
\end{array}\right. \\
\neg A \text{ — } \neg_{in} \text{ applied to } B \text{ and } \neg B
\end{array}
$$

The $nd\mathscr{F}$-derivation will be as follows:

$\vdots$
$\lceil$ A — the last alive assumption
$\vdots$
B
$\vdots$
$\neg B$
$\vdots$
$\lfloor$ $B \wedge \neg B$ — in a constant number of steps using $A \supset (B \supset (A \wedge B))$
$A \supset (B \wedge \neg B)$ — dr_n
$\lceil$ A — assumption
$\vdots$
$\lfloor$ B — in a constant number of steps using $(B \wedge \neg B) \supset B$
$A \supset B$ — dr_n
$\lceil$ A — assumption
$\vdots$
$\lfloor$ $\neg B$ — in a constant number of steps using $(B \wedge \neg B) \supset B$
$A \supset \neg B$ — dr_n
$\vdots$
$\neg A$ — in a constant number of steps using $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$

□

Theorem 3. $\Gamma \vert\!\overset{nd\mathscr{F}}{\underset{n}{}} C \Rightarrow \Gamma \vert\!\overset{\mathbf{CPL_{ND}}}{\underset{O(n)}{}} C$

Proof. To prove this we simply note that assumptions and formulae of Γ in an $nd\mathscr{F}$-derivation become, respectively, assumptions and formulae ofΓ in a $\mathbf{CPL_{ND}}$-derivation. Similarly, each application of mp_n becomes an application of $\supset_{el}$, and each application of dr_n becomes an application of $\supset_{in}$. We substitute all instances of axiom schemata with their proofs which are constructed in a constant number of steps. □

Theorem 4. *Assume, there is a* $\mathbf{CPL_{ND}}$*-derivation of* C *from* Γ *in* n *steps. Then, there is a* $d\mathscr{F}$*-derivation of* C *from* Γ *in* $O(n)$ *steps.*

The proof is similar to the proof of Theorem 2.

Theorem 5. *Assume, there is a $d\mathscr{F}$-derivation of C from Γ in n steps. Then, there is a* $\mathbf{CPL_{ND}}$*-derivation of C from Γ in $O(n^2)$ steps.*

Proof. For this theorem, let $\bigwedge\limits_{i=1}^{m} A_i$ be a conjunction of m formulae A_i which are ordered arbitrarily. Also, if Γ is a finite set of formulae, then $\bigwedge(\Gamma_1 \cup \Gamma_2)$ is a conjunction of its members ordered and associated arbitrarily.

It suffices to prove that if $\{A_1, \ldots, A_m\} \mapsto C$ has a $d\mathscr{F}$-proof of the length n, then $\bigwedge\limits_{i=1}^{m} A_i \supset C$ has a $\mathbf{CPL_{ND}}$-proof of the length $O(n^2)$. The proof of this theorem is similar to the proof of Theorem 4 in [11]. We substitute each sequent in a $d\mathscr{F}$-derivation with its relevant formula and then fill in the gaps. Now we show that all gaps can be filled in $O(n)$ steps. The proof splits into four cases depending on how the sequent in a $d\mathscr{F}$-derivation was inferred.

Case 1 The sequent has the form $\mapsto A$, where A is an instance of an axiom schema. Then we substitute it with the formula A which can be proved in a constant number of steps (since A is a tautology).

Case 2 The sequent has the form $A \mapsto A$, where A is an assumption. We substitute it with the formula $A \supset A$ which has a $\mathbf{CPL_{ND}}$-derivation of a constant number of steps.

Case 3 The sequent was inferred by mp_g. Then it has the form $\Gamma_1 \cup \Gamma_2 \mapsto B$ and there are also two sequents prior to it, namely, $\Gamma_1 \mapsto A \supset B$ and $\Gamma_2 \mapsto A$. It suffices to show that $\bigwedge(\Gamma_1 \cup \Gamma_2) \supset B$ can be inferred from $\bigwedge \Gamma_1 \supset (A \supset B)$ and $\bigwedge \Gamma_2 \supset A$. The derivation proceeds as follows.

$$
\begin{array}{l}
\vdots \\
\bigwedge \Gamma_1 \supset (A \supset B) \\
\vdots \\
\bigwedge \Gamma_2 \supset A
\end{array}
$$

$$
\begin{array}{l}
\left[\begin{array}{l}
\bigwedge(\Gamma_1 \cup \Gamma_2) \text{ — assumption} \\
\vdots \\
\bigwedge \Gamma_1 \text{ applying } \wedge_{el} \text{ to } \bigwedge(\Gamma_1 \cup \Gamma_2) \text{ and then } \wedge_{in} \\
\vdots \\
\bigwedge \Gamma_2 \text{ applying } \wedge_{el} \text{ to } \bigwedge(\Gamma_1 \cup \Gamma_2) \text{ and then } \wedge_{in} \\
A \supset B \text{ — applying } \supset_{el} \text{ to } \bigwedge \Gamma_1 \supset (A \supset B) \text{ and } \bigwedge \Gamma_1 \\
A \text{ — } \supset_{el} \text{ applying } \supset_{el} \text{ to } \bigwedge \Gamma_2 \supset A \text{ and } \bigwedge \Gamma_2 \\
B \text{ — } \supset_{el} \text{ applying } \supset_{el} \text{ to } A \supset B \text{ and } A
\end{array}\right. \\
\bigwedge(\Gamma_1 \cup \Gamma_2) \supset B \text{ — applying } \supset_{in} \text{ to } B
\end{array}
$$

If there are m formulae in $\bigwedge(\Gamma_1 \cup \Gamma_2)$, it can be shown by induction on m that $\bigwedge \Gamma_1$ and $\bigwedge \Gamma_2$ can be inferred from $\bigwedge(\Gamma_1 \cup \Gamma_2)$ in $O(m)$ steps via $\wedge_{el}$ and $\wedge_{in}$ rules. Since $m \leqslant n$, we can infer both $\bigwedge \Gamma_1$ and $\bigwedge \Gamma_2$ in $O(n)$ steps which proves the case.

Case 4 The sequent was inferred by dr_g. Then it has the form $\Gamma \mapsto A \supset B$ and there is also the sequent $\Gamma \setminus \{A\} \mapsto B$ prior to it. It suffices to show that $\bigwedge(\Gamma \setminus \{A\}) \supset (A \supset B)$ can be inferred from $\bigwedge \Gamma \supset B$ in $O(n)$ steps. We proceed as follows.

$$
\begin{array}{l}
\vdots \\
\bigwedge \Gamma \supset B \\
\left[\begin{array}{l}
\bigwedge(\Gamma \setminus \{A\}) \text{ — assumption (if } A \notin \Gamma) \\
\quad \left[\begin{array}{l}
A \text{ — assumption} \\
\vdots \\
\bigwedge \Gamma \text{ — from } \bigwedge(\Gamma \setminus \{A\}) \text{ and } A \text{ using } \wedge_{el} \text{ and } \wedge_{in} \\
B \text{ — } \supset_{el}
\end{array}\right. \\
A \supset B \text{ — } \supset_{in}
\end{array}\right. \\
\bigwedge(\Gamma \setminus \{A\}) \supset (A \supset B) \text{ — } \supset_{in}
\end{array}
$$

If there are m formulae in Γ, then it can be shown by induction on m that $\bigwedge \Gamma$ can be derived in $O(m)$ steps from A and $\bigwedge(\Gamma \setminus \{A\})$. Since $m \leqslant n$, we infer $\bigwedge(\Gamma \setminus \{A\}) \supset (A \supset B)$ from $\bigwedge \Gamma \supset B$ in $O(n)$ steps which proves the case. □

We will prove theorems showing the speedup of $\mathbf{Kl_{ND}}$ over the axiomatic calculus for **Kl** presented above which we will further designate as $\mathbf{Kl_{Ax}}$.

Definition 7 (proof simulation, speedup)**.** A proof system S_1 simulates S_2 with an $f(n)$ increase in number of steps if for any S_2-proof of formula A in n steps there is

a proof of A in S_1 in $O(f(n))$ steps. We say that S_2 provides at most $f(x)$ speedup w.r.t. S_1 if S_1 simulates S_2 with an increase of number of steps in $f(x)$.

Theorem 6. $\mathbf{Kl_{ND}}$ *linearly simulates* $\mathbf{Kl_{Ax}}$.

The proof of this theorem is straightforward since $\mathbf{Kl_{ND}}$ has modus ponens rule ($\supset_{el}$) and all axioms have $\mathbf{Kl_{ND}}$-proofs of a constant length. The details are left to the reader.

As it had been shown in [1], $\mathbf{Kl_{Ax}}$ is sound and complete (and so is $\mathbf{Kl_{ND}}$). This means that we can add $\supset_{in}$ rule to $\mathbf{Kl_{Ax}}$ thus transforming it into the natural calculus which we will further denote as $\mathbf{Kl_{Ax}}n$. One can see that $\mathbf{Kl_{Ax}}n$ is actually a nested deduction Frege system for $\mathbf{Kl}$ — hence our use of the index n for this system.

Theorem 7. $\mathbf{Kl_{ND}}$ *and* $\mathbf{Kl_{Ax}}n$ *linearly simulate one another.*

Proof. It is obvious that $\mathbf{Kl_{ND}}$ linearly simulates $\mathbf{Kl_{Ax}}n$ since all axioms can be proven in a constant number of steps while instances of modus ponens and $\supset_{in}$ as well as assumptions in a $\mathbf{Kl_{Ax}}n$-derivation become, without loss of generality, instances of $\supset_{el}$, $\supset_{in}$ and assumptions in a $\mathbf{Kl_{ND}}$-derivation.

Next we show that $\mathbf{Kl_{Ax}}n$ linearly simulates $\mathbf{Kl_{ND}}$. It suffices to show that we can obtain conclusions of all rules of derivation from their premises in a constant number of steps. We will prove the cases of $\vee_{el}$ and $\supset_{p}$ rules only.

$\vee_{el}$ $\mathbf{Kl_{ND}}$-proof has the following form:

$$
\begin{array}{l}
A \vee B \\
\left[\begin{array}{l} A \text{ — assumption} \\ \vdots \\ C \end{array}\right. \\
\left[\begin{array}{l} B \text{ — assumption} \\ \vdots \\ C \end{array}\right. \\
C \text{ — } \vee_{el}
\end{array}
$$

We proceed here as follows.

$A \vee B$
$\lceil$ A — assumption
$\vdots$
$\lfloor$ C
$A \supset C$ — $\supset_{in}$ — to C
$\lceil$ B — assumption
$\vdots$
$\lfloor$ C
$B \supset C$ — $\supset_{in}$ — to C
$\vdots$
C — in a constant number of steps using $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$

$\supset_p$ $\mathbf{Kl_{ND}}$-proof has the following form:

$\lceil$ $A \supset B$ — assumption
$\vdots$
$\lfloor$ A
A — $\supset_p$

We proceed here as follows.

$\lceil$ $A \supset B$ — assumption
$\vdots$
$\lfloor$ A
$(A \supset B) \supset A$ — $\supset_{in}$ to A
$\vdots$
A — in a constant number of steps using $((A \supset B) \supset A) \supset A$

□

Theorem 8. *If there is an* $\mathbf{Kl_{ND}}$*-proof of* A *of length* n*, then there is a* $\mathbf{Kl_{Ax}}$*-proof of* A *of length* $O(n \cdot \alpha(n))$ *with* α *being the inverse Ackermann function.*

Theorem 7 shows that $\mathbf{Kl_{Ax}}n$ linearly simulates $\mathbf{Kl_{ND}}$. The former is, by virtue of definition, a nested deduction Frege system. This means that we can apply the result of Buss and Bonet (Main Theorem 6 proved in [11]) which states that nested

deduction Frege systems provide a near-linear speedup over Frege systems (and $\mathbf{Kl_{Ax}}$ is a Frege system).

The above observations at least give us an idea how some fragments of proof search technique perform from the point of view of complexity. It also gives us grounds to expect that similar developments can be applied to the case of non-classical logics.

Concluding this section, we note that Theorems 6-8 provide us with an important tool of checking whether or not our proof-search algorithm presented in the next section is optimal. We know that natural deduction for **Kl** gives at most a near-linear speedup over Frege system for **Kl**. This means that we can test our algorithm on known examples that are hard for proof systems like analytical tableaux but have Frege proofs in a polynomial number of steps. If an algorithmic proof happens to be near-linearly faster than Frege proof, the algorithm works optimally at least on these examples. On the other hand, if the algorithm proves these examples polynomially slower than Frege system does, we will learn that it is not optimal. Finally, if the algorithm proves these formulae in an exponential number of steps, we will find out that it is considerably less effective than Frege systems.

Concluding this section, we observe that these general theoretical discussions should be supported by the study of the implementation of the proof searching algorithm, which forms part of our future work.

5 Algorithmic Proof Searching for $\mathbf{Kl_{ND}}$

The potential of the application of a logical deductive method to some practical specification/verification problem depends on the existence of the proof search and its efficiency. Here, we review the proof search technique for the logic **Kl** originally defined in [10]. To keep the presentation self-contained, we describe the procedures behind this search and then present the searching algorithm referring an interested reader to [10] for full details.

The proof search strategy is *goal-directed*, which means that it runs over two sequences: list_proof and list_goals. The former is a list of formulae in the proof, while the latter is a list of goals to be reached. A specific goal, the last goal in list_goals, is called *current_goal*. We identify three types of goals in list_goals.

Definition 8 (Types of goals)**.** A goal, $G_i, 0 \leq i \leq n$, occurring in list_goals $= \langle G_0, G_1, \ldots, G_n \rangle$, is one of the following

- G_i is a formula B, or

- G_i is of the form $[A]B$, i.e, it is a derivation of a formula B from an assumption A, or
- G_i is a contradiction, i.e. two contradictory **Kl** formulae, A and $\neg A$. In this case we will write $G_i = \bot$.

In our introductory case, we have a task to derive B from the specification S, given the assumptions Γ, or $S, \Gamma \Vdash B$. Note that here and below we distinguish the task of establishing that B is derivable from S, Γ (abbreviated by $S, \Gamma \Vdash B$) from the statement that such a derivation exists ($S, \Gamma \vdash B$). We will see that our searching procedures transform derivation tasks. Thus, $\mathsf{list\_proof} = \{A | A \in S \cup \Gamma\}$ and $\mathsf{list\_goals} = B$. Now, if our goal is not reachable, we either simplify compound formulae in list_proof invoking applicable elimination rules, or manage list_goals to generate new goals, applying introduction rules only when and if necessary. Each step of the algorithmic proof is associated with the *current_goal*. In our introductory case *current_goal* $= B$. Checking the reachability of the current goal, one of the core procedures, is introduced below and is based on Definition 8.

Definition 9 (Current goal reachability)**.** Current goal, G_n, $0 \leq n$, occurring in $\mathsf{list\_goals} = \langle G_0, G_1, \ldots, G_n \rangle$, 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 derivation of B from a non-discarded assumption A, or
- G_n is a contradiction and there are two contradictory formulae, $A \in$ list_proof and $\neg A \in$ list_proof.

5.1 Proof-Searching Algorithm $\mathbf{Kl}_{\mathbf{ND}\mathsf{ALG}}$

Now we are ready to introduce the notion of an *algo-derivation* and searching procedures involved.

Definition 10 (Algo-derivation $\mathbf{Kl}_{\mathbf{ND}\mathsf{ALG}}$)**.** A **Kl** *algo-derivation*, abbreviated as $\mathbf{Kl}_{\mathbf{ND}\mathsf{ALG}}$, is a pair (list_proof, list_goals) whose construction is determined by the searching Procedures (1)–(4) outlined below.

5.1.1 Searching Procedures

Searching Procedures below update list_proof, list_goals or both of them.

Procedure (1) Here we follow one of the main ideas of natural deduction proof to simplify structures of obtained formulae: list_proof is updated due to an applicable elimination rule. If we find a formula, or two formulae, which can serve as premises of one of these rules, the rule is enforced and the sequence list_proof is updated by the relevant conclusion.

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 procedure applies when the current goal is not reached. Analysing the structure of the current goal we update 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 which depends on the possible structures of G_n:

$$G_n = A \wedge B | A \vee B | A \supset B | \neg(A \wedge B) | \neg(A \vee B) | \neg(A \supset B) | L | \neg\neg A | \bot | [C]A$$

where A, B are any formulae, $L \in Lit$ and $[C]A$ states for the derivation of A from assumption C. The rules below have structure $\Gamma \Vdash \alpha \longrightarrow \Gamma' \Vdash \alpha'$ indicating that the rule modifies some given derivation task $\Gamma \Vdash \alpha$ to a new derivation task $\Gamma' \Vdash \alpha'$. The procedures depend on the structure of the current goal: they tackle the cases when the current goal is a compound **Kl** formula. The last type of the goal — $\bot$ — is managed as follows.

$$(2.1.1) \quad \Gamma \Vdash \Delta, A \wedge B \quad \longrightarrow \quad \Gamma \Vdash \Delta, A \wedge B, B, A$$

In the above, Procedure (2.1.1) splits the current conjunctive goal into two conjuncts.

$$(2.1.2.1) \quad \Gamma \Vdash \Delta, A \vee B \quad \longrightarrow \quad \Gamma \Vdash \Delta, A \vee B, A$$
$$(2.1.2.2) \quad \Gamma \Vdash \Delta, A \vee B \quad \longrightarrow \quad \Gamma \Vdash \Delta, A \vee B, B$$

Procedure (2.1.2) tackles a disjunctive goal $A \vee B$ setting each disjunct as a separate goal. We need some clarifications for Procedures (2.1.2.1) and (2.1.2.2) to explain the way how we avoid infinite loops invoking a dedicated marking technique. For the former, when the current goal is disjunction, we try to reach the left disjunct (Procedure 2.1.2.1), and if we fail this subroutine is deleted and we apply Procedure (2.1.2.2). Similarly, if the latter fails we delete this subroutine and terminate the whole Procedure (2.1.2).

$$(2.1.3) \quad \Gamma \Vdash \Delta, A \supset B \quad \longrightarrow \quad \Gamma, A \Vdash \Delta, A \supset B, B$$

Procedure (2.1.3) tackles $A \supset B$ as a goal, requiring to update list_proof with A and list_goals with B.

$$\begin{array}{lrcl} (2.1.4) & \Gamma \Vdash \Delta, \neg(A \supset B) & \longrightarrow & \Gamma \Vdash \Delta, \neg(A \supset B), A, \neg B \\ (2.1.5) & \Gamma \Vdash \Delta, \neg(A \vee B) & \longrightarrow & \Gamma \Vdash \Delta, \neg(A \vee B), \neg A, \neg B \\ (2.1.6) & \Gamma \Vdash \Delta, \neg(A \wedge B) & \longrightarrow & \Gamma \Vdash \Delta, \neg(A \wedge B), \neg A \vee \neg B \end{array}$$

Procedures (2.1.4)–(2.1.6) transform negative compound goals $\neg(A \supset B)$, $\neg(A \vee B)$, $\neg(A \wedge B)$ into $A \ \wedge \neg B$, $\neg A \ \wedge \neg B$ and $\neg A \ \vee \neg B$, respectively.

$$\begin{array}{lrcl} (2.1.7.1) & \Gamma \Vdash \Delta, F & \longrightarrow & \Gamma \Vdash \Delta, F, \bot \\ (2.1.7.2) & \Gamma \Vdash \Delta, F & \longrightarrow & \Gamma, F \supset p \wedge \neg p \Vdash \Delta, [F \supset p \wedge \neg p]F \end{array}$$

Here F is a literal (a proposition or its negation) or $F = A \vee B$ and variable p should be fresh.

In the paracomplete setting, we also reason by refutation. When the current goal is not reached, and it is either a literal or disjunction (not reached by Procedure (2.1.2)) we first look for the contradictions in the proof — Procedure (2.1.7.1) which sets up a new goal, $\bot$.

If no contradictions are found, then we turn into the refutation style proof applying Procedure (2.1.7.2). The application of this procedure is linked to the rule $\supset_p$ which allows us to introduce to list_proof the derivation of F from $F \supset p \wedge \neg p$, the goal of Procedure (2.1.7.2), once this goal is achieved.

$$\begin{array}{lrcl} (2.1.8) & \Gamma \Vdash \Delta, \neg\neg A & \longrightarrow & \Gamma \Vdash \Delta, \neg\neg A, A \\ (2.1.9) & \Gamma \Vdash \Delta, [A]B & \longrightarrow & \Gamma, A \Vdash \Delta, B \end{array}$$

Procedure (2.1.9) corresponds to our interpretation of assumptions — the given goal $[A]B$ means to infer B from the assumption A, hence we update list_goals by B and list_proof by the assumption A.

Marking Various marking routines are applied to prevent infinite looping during the search. For example, applying Procedure (2.1) we mark literals and formulae of the type $A \vee B$. This mark serves proof by refutation — in reaching relevant goals we cannot any longer apply reasoning by refutation. Also, applying Procedure (2.1.7.2), we mark the assumption that this procedure defines, and these marks indicate that this assumption, and any formula which is derivable from it, cannot serve as source of a new goal, i.e. Procedure (2.2) described below, is not applicable (otherwise, the proof search will enter an infinite loop). Our example in §5.2 will further clarify how marking technique affects proof search.

Procedure (2.2). Here we analyse compound disjunctive and implicative formulae (but not of the type $A \supset \bot$, where $\bot$ is any contradiction, as explained above) contained in list_proof in order to find sources for new goals. If one of these formulae is found then its structure determines the generation of a new goal.

$$\begin{array}{lllll} (2.2.1) & \Gamma, A \vee B \Vdash \Delta, C & \longrightarrow & \Gamma \Vdash \Delta, [A]C & \Gamma \Vdash \Delta, [B]C \\ (2.2.2) & \Gamma, A \supset B \Vdash \Delta, C & \longrightarrow & \Gamma \Vdash \Delta, C, A & \end{array}$$

Procedure (3) Here we check the application of Definition 9. If the current goal G_n, $(n > 0)$ is reached, we delete G_n from the sequence list_goals and set G_{n-1} as the current goal. If the current goal G_0 is reached, we delete G_0 from the sequence list_goals.

Procedure (4) This is a search for an applicable introduction rule. It is based on the association of Procedures (2.1.1)–(2.1.8) with correspondent introduction rules presented below.

$$\begin{array}{rcl@{\qquad}rcl} \text{Procedure (2.1.1)} & \longrightarrow & \wedge_{in} & \text{Procedure (2.1.5)} & \longrightarrow & \neg\vee_{in} \\ \text{Procedure (2.1.2.1)} & \longrightarrow & \vee_{in_1} & \text{Procedure (2.1.6)} & \longrightarrow & \neg\wedge_{in} \\ \text{Procedure (2.1.2.2)} & \longrightarrow & \vee_{in_2} & \text{Procedure (2.1.7)} & \longrightarrow & \supset_{p} \\ \text{Procedure (2.1.3)} & \longrightarrow & \supset_{in} & \text{Procedure (2.1.8)} & \longrightarrow & \neg_{in} \\ \text{Procedure (2.1.4)} & \longrightarrow & \neg\supset_{in} & & & \end{array}$$

Note that Procedure (4) represents the unique specifics of our searching technique — it makes the application of the introduction rules completely determined by the analysis of the structure of the current goal (reached) and its preceding goals.

5.1.2 Algorithm $\mathbf{Kl_{ND_{ALG}}}$

Let us introduce the following abbreviations

- 'G_{cur}' abbreviates the current goal in list_goals
- 'last(list_goals)' returns the last element of list_goals, and
- list_goals — G_n deletes the last formula, G_n, from list_goals.

Now, based on the procedures (1)-(4) we introduce the proof search algorithm $\mathbf{Kl_{ND_{ALG}}}$ making comments to the steps of the algorithm within the '//'.

(0) list_proof(), list_goals(), *go to* (1) // initialisation of sequences list_proof and list_goals//

(1) Given a task $\Gamma \Vdash G_0$, $G_{cur} = G_0$ // initialisation of G_{cur} as G_0

$(\Gamma \neq \emptyset) \longrightarrow (\mathsf{list\_proof} = \Gamma, \mathsf{list\_goals} = G_0, \textit{go to } (2))$// when Γ is not empty update list_proof with formulae of Γ and list_goals with G_0//

ELSE

$\mathsf{list\_goals} = G_0, \textit{go to } (2)$ // when there are no given assumptions in Γ only update list_goals with G_0//

(2) Procedure $(3)(G_{cur}) = \textit{true}$ //checks the reachability of the current goal//

(2a) IF Reached (G_{cur}) = **true**, then $\mathsf{list\_goals} = \mathsf{list\_goals} - G_{cur}$ // when the current goal is reached it is deleted from list_goals, the new current goal is the previous goal in list_goals//

THEN

IF $(G_{cur} = G_0) \longrightarrow \textit{go to } (6a)$ // If the initial goal is reached, go to the terminating step//

ELSE

$G_{cur} \neq G_0$, then $G_{cur} = \textit{last}(\mathsf{list\_goals})$ *go to* (3)//If the reached goal is not the initial goal determine a new G_{cur} as the last goal in $G_{cur} = \textit{last}(\mathsf{list\_goals})$ and proceed with the relevant introduction rule//

(2b) IF Reached (G_{cur}) = **false**, THEN *go to* (4)//If (G_{cur}) is not reached proceed further with elimination rules//

(3) Procedure $(4)(\langle \mathsf{list\_proof}, \mathsf{list\_goals} \rangle)$ = **true** //apply a relevant introduction rule// *go to* (2).

(4) Procedure $(1)(\langle \mathsf{list\_proof} \rangle)$ = **true** //apply elimination rules//

(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 $(2)((\langle \mathsf{list\_proof}, \mathsf{list\_goals} \rangle)$ = **true**) // update list_proof and list_goals based on the structure of G_{cur}//

(5a) Procedure $(2.1)(\langle \mathsf{list\_proof}, \mathsf{list\_goals} \rangle)$ = **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 $\mathbf{Kl_{NDALG}}$.

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

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

5.2 Algo-Proof Example

As an example of an algorithmic ND proof we apply $\mathbf{Kl_{NDALG}}$ as an attempt to prove the following formula

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

Note that this formula is valid in the classical setting and is not in the setting of paracomplete logic. Its validity would have led to the validity of $\neg p \vee p$ as shown in the following: if ($\natural$) is valid then so would be

$$(\sharp) \quad (p \supset p) \supset (\neg p \vee p),$$

now since $p \supset p$ is valid, by modus ponens, we would derive $\neg p \vee p$.

This is an indicative formula which contains a disjunctive constraint and as the reader will see in the proof attempt, all core procedures related to disjunctive formulae are invoked.

Let us introduce a useful concept of *algo-step* which will make the understanding of the application of proof search easier. Recall that an algo-proof is a pair (list_proof, list_goals). At each step of the application of the procedures described above we have the sequences list_proof and list_goals of specific lengths, say i and j. Let's abbreviate them by (list_proof$_i$, list_goals$_j$), respectively, and let list_proof = $B_1, \ldots, B_i$ and list_goals = $G_0, \ldots, G_j$, where G_j is the last goal, that is it is the current goal. So an algo-step is the task to find a derivation $B_1, \ldots, B_i \Vdash G_0, \ldots, G_j$. Thus, the algo-proof for some formula C (with no given assumptions) commences with the first algo-step $\Vdash G_0$, where $G_0 = C$.

Now, for the input $(p \supset q) \supset (\neg p \vee q)$, we commence the proof with the main goal, $(p \supset q) \supset (\neg p \vee q)$. According to the classical search Procedure (2.1.3), the antecedent of the main goal, $p \supset q$, becomes the new assumption, and its consequent, $\neg p \vee q$ — the new goal, $G_1 = \neg p \vee q$. So the next algo-step would be $p \supset q \Vdash \neg p \vee q$. In the representation of the algo-proof below we will have the following columns indicating, in order, a step of the algo proof (step), so the abbreviation $as0$ stands for the first algo-step, formulae in the proof (list_proof), an annotation explaining how a formula appears in list_proof, and finally, a list of the goals (list_goals).

step	list_proof	annotation	list_goals
$as0$			$G_0 = (p \supset q) \supset (\neg p \vee q)$
$as1$	1. $p \supset q$	assumption	$G_0, G_1 = \neg p \vee q$

The current goal $G_1 = \neg p \vee q$ cannot be reached so we apply Procedure (2.1.2.1) and set a new goal $G_2 = \neg p$, hence list_goals $= \neg p \vee q, \neg p$. Since $\neg p$ is not reachable, we delete it from list_goals, and applying Procedure (2.1.2.2) we set a new goal $G_2 = q$, hence list_goals $= \neg p \vee q, q$. Since q is not reachable, we delete it from list_goals. At this stage we have failed to reach both disjuncts of G_1. Hence we start the refutation, applying first Procedure (2.1.7.1). Thus, we set up a new goal $G_2 = \bot$. This new goal, in turn, is not derivable, so we delete G_2 from list_goals, and apply Procedure (2.1.7.2) adding (a) a new assumption, $(\neg p \vee q) \supset (r \wedge \neg r)$ and (b) a new goal, $[(\neg p \vee q) \supset (r \wedge \neg r)]\neg p \vee q$. Here we mark the assumption on step 'as3' indicating that it should not be subject to Procedure (2.2).

Note that, according to the definition of Procedure (2.1.7.1), in the $r \wedge \neg r$ constraint, the variable r should be fresh.

step	list_proof	annotation	list_goals
$as0$			$G_0 = (p \supset q) \supset (\neg p \vee q)$
$as1$	1. $p \supset q$	assumption	$G_0, G_1 = \neg p \vee q$
$as2$			$G_0, G_1, G_2 = \bot$
$as3$	2. $(\neg p \vee q) \supset (r \wedge \neg r)$	*assumption*	$G_0, G_1, G_2 =$ $[(\neg p \vee q) \supset (r \wedge \neg r)]\neg p \vee q$

Now, looking for the applicable elimination rule, we notice that $\vee\supset_{el_1}$ and $\vee\supset_{el_2}$ are applicable to formula 2, thus we derive steps 3 and 4.

step	list_proof	annotation	list_goals
$as0$			$G_0 = (p \supset q) \supset (\neg p \vee q)$
$as1$	1. $p \supset q$	assumption	$G_0, G_1 = \neg p \vee q$
$as2$		$G_0, G_1, G_2 = \bot$	
$as3$	2. $(\neg p \vee q) \supset (r \wedge \neg r)$	assumption	$G_0, G_1, G_2 =$ $[(\neg p \vee q) \supset (r \wedge \neg r)]\neg p \vee q$
$as4$	3. $\neg p \supset (r \wedge \neg r)$	$\vee \supset_{el_1}$	G_0, G_1, G_2
$as5$	4. $q \supset (r \wedge \neg r)$	$\vee \supset_{el_2}$	G_0, G_1, G_2

At this stage, the current goal, G_2 is not reachable, so we look for the sources of new goals analysing compound formulae in the proof applying Procedure (2.2.2). Hence by analysing step 1, we set up a new goal $G_3 = p$. This is not reachable, so we again apply Procedure (2.1.7.1), setting a new goal, $G_4 = \bot$.

step	list_proof	annotation	list_goals
$as0$			$G_0 = (p \supset q) \supset (\neg p \vee q)$
$as1$	1. $p \supset q$	assumption	$G_0, G_1 = \neg p \vee q$
$as2$			$G_0, G_1, G_2 = \bot$
$as3$	2. $(\neg p \vee q) \supset (r \wedge \neg r)$	assumption	$G_0, G_1, G_2 =$ $[(\neg p \vee q) \supset (r \wedge \neg r)]\neg p \vee q$
$as4$	3. $\neg p \supset (r \wedge \neg r)$	$\vee \supset_{el_1}$	G_0, G_1, G_2
$as5$	4. $q \supset (r \wedge \neg r)$	$\vee \supset_{el_2}$	G_0, G_1, G_2
$as6$			$G_0, G_1, G_2, G_3 = p, G_4 = \bot$

The current goal, G_4 is not reachable, so we delete it and applying Procedure (2.1.7.2) we set up a new assumption, $p \supset (s \wedge \neg s)$ and a new goal, $G_4 = [p \supset (s \wedge \neg s)]p$. Note that s is a fresh variable.

step	list_proof	annotation	list_goals
$as0$			$G_0 = (p \supset q) \supset (\neg p \vee q)$
$as1$	1. $p \supset q$	assumption	$G_0, G_1 = \neg p \vee q$
$as2$			$G_0, G_1, G_2 = \bot$
$as3$	2. $(\neg p \vee q) \supset (r \wedge \neg r)$	assumption	$G_0, G_1, G_2 =$ $[(\neg p \vee q) \supset (r \wedge \neg r)]\neg p \vee q$
$as4$	3. $\neg p \supset (r \wedge \neg r)$	$\vee \supset_{el_1}$	G_0, G_1, G_2
$as5$	4. $q \supset (r \wedge \neg r)$	$\vee \supset_{el_2}$	G_0, G_1, G_2
$as6$			$G_0, G_1, G_2, G_3 = p, G_4 = \bot$
$as7$	5. $p \supset (s \wedge \neg s)$	assumption	$G_0, G_1, G_2, G_3, G_4 =$ $[p \supset (s \wedge \neg s)]p$

At this stage the searching algorithm terminates as there are no procedures to apply and all formulae in list_proof are marked: as a result, we still have goals to reach, however, no more elimination rules can be applied, we do not have any more formulae in list_proof that could give us new goals and, once again, introduction rules are only applied as a result of Procedure (4), which is now void. Note that although formula 2 in list_proof is compound, it was set up as an assumption due to Procedure (2.1.7.2), hence it is marked and is not considered as a source for new goals. These marks are carried on for the derivable formulae on steps 3 and 4.

Now, looking at the list_proof we can extract the counterexample as follows. Formula $p \supset (s \wedge \neg s)$ means that p has the value f while $q \supset (r \wedge \neg r)$ means q has the value f. Under these values for p and q, formula $(p \supset q) \supset (\neg p \vee q)$ also takes the value f.

5.3 Correctness

The following theorems reflect the metatheoretical properties of the above algorithm [10].

Theorem 9. $\mathbf{Kl_{ND}}_{ALG}$ *terminates for any input formula.*

Theorem 9 guarantees that for any input formula for the $\mathbf{Kl_{NDALG}}$ the sequences list_proof and list_goals are finite.

Theorem 10. $\mathbf{Kl_{NDALG}}$ *is sound.*

Theorem 10 ensures that every formula for which an ND proof is constructed according with $\mathbf{Kl_{NDALG}}$ is valid.

Theorem 11. $\mathbf{Kl_{NDALG}}$ *is complete.*

Theorem 11 establishes that for every valid formula, A, $\mathbf{Kl_{NDALG}}$ finds a $\mathbf{Kl_{ND}}$ proof.

Altogether, theorems 9, 10 and 11 imply the following fundamental property of our algorithm:

Theorem 12. *For any input formula A, the* $\mathbf{Kl_{NDALG}}$ *terminates either building up a* $\mathbf{Kl_{ND}}$*-proof for A or providing a counter-model.*

Let us now present some important observations on the proof search and on some of its core and important features.

As in the other ND calculi, in constructing an ND derivation, we are allowed to introduce arbitrary formulae as new assumptions. 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. It is true that without the proof search technique assumptions can be introduced arbitrarily. However, due to the goal-directed feature of the presented algorithm, any assumption that appears in the proof is well justified serving a specific target. Let us emphasise that we also turned the assumptions management into an advantage showing the applicability of the proposed technique to assume-guarantee reasoning as shown in §6.

We also note that, according to the algorithm, the order in which assumptions are discharged, is the reverse order to their introduction into the proof.

Finally, introduction rules that have been another point of scepticism concerning the automation of natural deduction, in our algorithm are completely determined. Namely, the reachability of the current goal and the type of the previous goal determine the relevant introduction rule. Also, though $\neg_{in}$ rule of our system $\mathbf{Kl_{ND}}$, in general, allows to derive any formula from the contradiction, the application of this rule is strictly determined by the searching procedures. Therefore, the formula that we derived from a contradiction is always the one mentioned in list_goals.

6 Applications in Specification-Based Verification

Our development of the automated reasoning technique tackles at this stage only the propositional basis. However, even at this more or less simple level, we argue that it can significantly contribute in specification-based verification.

6.1 Methodology of applying $\mathbf{Kl_{NDALG}}$ as Deductive Verification

Here we draw several routes of applying natural deduction enhanced with the proof search.

Below we list relevant problems and indicate the relevant methodology of their solution based on natural deduction.

1. To find if a system satisfies some desired property
   - 1.1. obtain the specification of the system, *Spec*, with some core properties, Γ and the specification of the desired property, say, B;
   - 1.2. find an ND derivation $\Gamma \Vdash B$.
2. To reason about requirements
   - 2.1. specify the requirements;

2.2. for a given requirement B, find if there is an ND proof of B;

2.3. drop such requirements since they are valid regardless of a system.

3. To check the consistency of a given system

   3.1. obtain the specification, *Spec*, of a system and run the searching technique to obtain the contradiction, i.e. setting up the goal $\bot$;

   3.2. if $\bot$ has been reached, the given system is inconsistent.
   We will show in the present section how this works in the framework of component-based system.

4. To look for non-explicit assumptions, apply the presented **Kl** proof search algorithm, and the procedures will automatically upgrade list_proof with new assumptions.
   We will show in this section how this works in finding assumptions in the framework of assume-guarantee reasoning.

In the following subsections we tackle problem setting 3–4 leaving the discussion of problems 1–2 for the conclusion.

6.2 Component-Based Systems

Here we justify the application of the natural deduction to component-based system assembly. Thus, we aim to apply the searching algorithm $\mathbf{Kl}_{\mathbf{ND}\mathsf{ALG}}$ as the deductive verification technique for a component system.

As an example, let us consider a simple component system interpreted in The Grid Component Model (GCM) based on Fractal [3].

Let our component system, *Sys* have the following specification *Spec*. Components interact together by being bound through interfaces. The system has four core components P, Q, R and S. Let p, q, r and s represent properties that core components, P, Q, R and S are bound to the system (one that should be always available and should not be "touched").

Consider as an example the following set of global requirements and their formalisation:

- whenever P is bound R should be bound: $p \supset r$
- whenever P is not bound S should be bound: $\neg p \supset s$.
- whenever Q is bound both R and S should not be bound: $q \supset (\neg r \wedge \neg s)$.
- Q should be bound to the system: q.

Consider now the verification task to establish if the above configuration of components is consistent. We commence the proof (see below) by the given conditions of the *Spec* and set up the goal of the procedure to derive the contradiction, abbreviated in the proof annotation below as $\bot$. If the contradiction is derivable, then we would have been able to see its sources tracing the proof backwards. Otherwise, the *Spec* would have been shown consistent.

We commence the proof by listing all four given formulae on steps 1-4. From 3 and 4 by eliminating implication we derive $\neg r \wedge \neg s$ and then eliminating conjunction from the latter, derive steps 6 and 7. We have not reached the goal $\bot$. By Procedure (2.2) we analyse compound formulae in the proof. Thus, analysing formula on step 1 we apply Procedure (2.2.2) and set up p, the antecedent of 1, as the new goal.

step	list_proof	annotation	goals
$as0$			$\bot$
$as1$	1. $p \supset r$	given	$\bot$
$as2$	2. $\neg p \supset s$	given	$\bot$
$as3$	3. $q \supset (\neg r \wedge \neg s)$	given	$\bot$
$as4$	4. q	given	$\bot$
$as5$	5. $\neg r \wedge \neg s$	$3, 4 \supset_{el}$	$\bot$
$as6$	6. $\neg r$	$5, \wedge_{el}$	$\bot$
$as7$	7. $\neg s$	$5, \wedge_{el}$	$\bot$
$as8$			$\bot, p$

The current goal, p has not been reached — we apply Procedure (2.1.7.1) setting up the new goal, $\bot$. If we derive $\bot$, then by $\mathbf{Kl}_{\neg_{in}}$ we would be able to derive the desired p. However, $\bot$ is not reachable so we delete it and apply Procedure (2.1.7.2) so the new assumption is $p \supset (t \wedge \neg t)$ (where $t \wedge \neg t$ is the formula $\bot$ in the formulation of Procedure (2.1.7.2)) and our task is now to derive p. Since we cannot do it we apply Procedure (2.2.2) and analyse formula 2 putting its antecedent, $\neg p$, as the new goal.

Again, as it is reachable we apply Procedures (2.1.7.1) and (2.1.7.2) consequently. The latter procedure sets up the new assumption $\neg p \supset (u \wedge \neg u)$ on step 9 and the

new goal $\neg p$, where $u \wedge \neg u$ is $\bot$ in Procedure (2.1.7.2).

step	list_proof	annotation	list_goals
$as0$			$\bot$
$as1$	1. $p \supset r$	given	$\bot$
$as2$	2. $\neg p \supset s$	given	$\bot$
$as3$	3. $q \supset (\neg r \wedge \neg s)$	given	$\bot$
$as4$	4. q	given	$\bot$
$as5$	5. $\neg r \wedge \neg s$	$3, 4 \supset_{el}$	$\bot$
$as6$	6. $\neg r$	$5, \wedge_{el}$	$\bot$
$as7$	7. $\neg s$	$5, \wedge_{el}$	$\bot$
$as8$			$\bot, p$
$as9$	8. $p \supset (t \wedge \neg t)$	assumption	$\bot, p, p$
$as10$			$\bot, p, p, \neg p$
$as11$	9. $\neg p \supset (u \wedge \neg u)$		$\bot, \neg p, \neg p$

At this moment, the proof search stops. A model is extractable as follows: p is assigned f because $p \supset (t \wedge \neg t)$ is in the list_proof or because $\neg p \supset (t \wedge \neg t)$ is in the list_proof. Note that p is assigned f if, and only if, $\neg p$ is assigned f. Next, r gets the value 0 because $\neg r$ is in the list_proof and s is assigned 0 because $\neg s$ is in the list_proof. Under this valuation, each formula $p \supset r, \neg p \supset s, q \supset (\neg r \wedge \neg s)$ and q is assigned 1. So, this set of formulae in *Spec* is consistent.

This explicitly shows the nature of the applicability of paracomplete logic — the given *Spec* does not have a precise information about p — if this component should be bound or not. So the reasoning stops.

Had we reasoned about this specification in the classical set up, we would have been able to use classically valid formula $p \vee \neg p$ (which is not valid in **Kl**) to derive the contradiction. We will give the corresponding proof a little later, after presenting a derivable rule which we will use in the proof:

$$\frac{A \supset B,\ C \supset D}{(A \vee C) \supset (B \vee D)}$$

Note that this rule is also derivable in logic $\mathbf{Kl_{ND}}$, so we will construct the proof applying our algorithm $\mathbf{Kl_{NDALG}}$. It will return the conclusion of this rule, $(A \vee C) \supset (B \vee D)$, given that the premisses are constituted.

The $\mathbf{Kl_{NDALG}}$ (hence the classical algorithm [9] as well) would set up $(A \vee C) \supset (B \vee D)$ as the main goal G_0 to be derived from the given set $A \supset B,\ C \supset D$. Because the goal is implicative, by Procedure (2.1.3), its antecedent $A \vee C$ becomes

the new assumption, and its consequent, $B \vee D$ — the new goal, G_1.

step	list_proof	annotation	list_goals
$as0$			$G_0 = ((A \vee C) \supset (B \vee D))$
$as1$	1. $A \supset B$	given	G_0
$as2$	2. $C \supset D$	given	G_0
$as3$	3. $A \vee C$	assumption	$G_0, G_1 = B \vee D$

The current goal, G_1, is disjunctive, therefore, by Procedure (2.1.2.1), the left disjunct of G_1 is set up as the new goal $G_2 = B$.

This goal cannot be reached so it is deleted from list_goals, and, by Procedure (2.1.2.2), the right disjunct is set up as the new goal $G_2 = D$.

This goal cannot be reached so it is deleted from list_goals. Therefore, we have a disjunctive goal G_1 which so far has not been reached.

Next, the Procedure (2.2.1) is fired. The algorithm finds a disjunctive formula $A \vee C$ in list_proof and it should take in turn two branches.

First, to derive G_1 adding A as the new assumption and then to derive G_1 adding C as the new assumption.

Solving the first derivation, A is the new assumption on step 4 as below. Now G_1 is a disjunctive goal and its antecedent becomes the new goal $G_2 = B$.

This can be reached by eliminating implication from 1 and 4 obtaining B on step 5 and then introducing disjunction to the latter obtaining $B \vee D$ on step 6.

step	list_proof	annotation	list_goals
$as0$			$G_0 = ((A \vee C) \supset (B \vee D))$
$as1$	1. $A \supset B$	given	G_0
$as2$	2. $C \supset D$	given	G_0
$as3$	3. $A \vee C$	assumption	$G_0, G_1 = B \vee D$
$as4$			$G_0, G_1, G_2 = B$
$as5$	4. A	assumption	G_0, G_1, G_2
$as6$	5. B	$1, 4 \supset_{el}$	G_0, G_1
$as7$	6. $B \vee D$	$5, \vee_{in}$	G_0, G_1

Although we have obtained $B \vee D$ on step 6, we have not reached the goal G_1 — to reach the latter we also need to achieve the second subderivation — from the set of formulae $1, 2, 3, C$ where C is the new assumption, to derive $B \vee D$. The

application of the algorithm is similar to the above, so the proof continues as follows:

step	list_proof	annotation	list_goals
$as0$			$G_0 = ((A \vee C) \supset (B \vee D))$
$as1$	1. $A \supset B$	given	G_0
$as2$	2. $C \supset D$	given	G_0
$as3$	3. $A \vee C$	assumption	$G_0, G_1 = B \vee D$
$as4$			$G_0, G_1, G_2 = B$
$as5$	4. A	assumption	G_0, G_1, G_2
$as6$	5. B	$1, 4 \supset_{el}$	G_0, G_1
$as7$	6. $B \vee D$	$5, \vee_{in}$	G_0, G_1
$as8$			$G_0, G_1, G_2 = D$
$as9$	7. C	assumption	G_0, G_1, G_2
$as10$	8. D	$2, 7 \supset_{el}$	G_0, G_1
$as11$	9. $B \vee D$	$8, \vee_{in}$	G_0, G_1

Both subderivations tasks have been completed so the algorithm applies $\vee_{el}$ rule as we have the disjunctive formula $A \vee C$ in the proof and from either of its disjuncts we have derived $B \vee D$. The result of this rule is $B \vee D$ on step 10 with annotations as below.

Finally, introducing implication to the formula on step 10 we derive the desired goal G_0 from the formulae on steps 1 and 2.

step	list_proof	annotation	list_goals
$as0$			$G_0 = (A \vee C) \supset (B \vee D)$
$as1$	1. $A \supset B$	given	G_0
$as2$	2. $C \supset D$	given	G_0
$as3$	3. $A \vee C$	assumption	$G_0, G_1 = B \vee D$
$as4$			$G_0, G_1, G_2 = B$
$as5$	4. A	assumption	G_0, G_1, G_2
$as6$	5. B	$1, 4 \supset_{el}$	G_0, G_1
$as7$	6. $B \vee D$	$5, \vee_{in}$	G_0, G_1
$as8$			$G_0, G_1, G_2 = D$
$as9$	7. C	assumption	G_0, G_1, G_2
$as10$	8. D	$2, 7 \supset_{el}$	G_0, G_1
$as11$	9. $B \vee D$	$8, \vee_{in}$	G_0, G_1
$as12$	10. $B \vee D$	$3, 4, 7, [4-6], [7-9]$	G_0
$as13$	11. $(A \vee C) \supset (B \vee D)$	$10, \supset_{in}, [3-10]$	

Now we use this derivable rule returning to the task of showing that in the classical setting the given $SPEC$ is inconsistent. We will not show below the algo-steps as they would correspond to the steps of the proof.

list_proof	annotation
1. $p \supset r$	given
2. $\neg p \supset s$	given
3. $q \supset (\neg r \wedge \neg s)$	given
4. q	given
5. $(p \vee \neg p) \supset (r \vee s)$	$1, 2,$ derived rule
6. $\neg r \wedge \neg s$	$3, 4, \supset_{el}$
7. $\neg r$	$6, \wedge_{el}$
8. $\neg s$	$6, \wedge_{el}$
9. $p \vee \neg p$	classical validity
10. $r \vee s$	$5, 9, \supset_{el}$
11. s	$7, 10, \vee_{el}$

Now steps 8 and 11 constitute the contradiction hence the classical reasoning would have detected the contradiction while, in fact, in our initial setup with incomplete knowledge on p we do not have any inconsistency due to this lack of the exact information about the truth conditions of p.

6.3 Assume-Guarantee Reasoning

We consider here how the reasoning based upon natural deduction can be applied to the automation of the assume-guarantee reasoning [19, 36] technique, the most used technique in the framework of compositional analysis.

In assume-guarantee reasoning, a verification problem is represented as a triple, $\langle A \rangle S \langle P \rangle$, where S is the subsystem being analyzed, P is the property to be verified, and A is an assumption about the environment in which S is used.

The standard interpretation of $\langle A \rangle S \langle P \rangle$ suggests that A is a constraint on S and if S as constrained by A satisfies P, then the formula $\langle A \rangle S \langle P \rangle$ is true.

Let us formulate the semantics of $\langle A \rangle S \langle P \rangle$ in the following way: $S/A \models P$ where S/A means the system S with the additional information A. Now, the typical example of the application of assume-guarantee reasoning is in the context of decomposing a given system S into two subsystems S_1 and S_2 that run in parallel. Suppose we need to verify that the property P is satisfied in S. Then we can apply the assume-guarantee rule † as follows.

$$(\dagger)\quad \frac{\langle A\rangle S_1\langle P\rangle \quad \langle \mathbf{true}\rangle S_2\langle A\ \rangle}{\langle \mathbf{true}\rangle S_1||S_2\langle P\rangle}$$

Here $\langle \mathbf{true}\rangle S_2\langle A\rangle$ and $\langle \mathbf{true}\rangle S_1||S_2\langle P\rangle$ mean, respectively, that A is verified in S_2 (without any constraints) and P is verified in $S_1||S_2$ (without any constraints).

In terms of natural deduction we can rewrite this rule as ‡ below.

$$(\ddagger)\quad \frac{S_1, A \vdash P \quad S_2 \vdash A}{S_1||S_2 \vdash P}$$

Now new tasks are to find the natural deduction derivations $S_1, A \Vdash P$ and $S_2 \Vdash A$ in order to conclude that $S_1||S_2 \vdash P$ and the application of the proof search technique is the next logical step here.

One of the major obstacles in the efficient application of assume-guarantee approach [15] is that once decomposition is selected, to manually find an assumption A to complete an assume-guarantee proof is difficult. Indeed, the assumption must be strong enough to sufficiently constrain the behavior of S_1 so that $S_1, A \vdash P$ holds, and must be weak enough so that $S_2 \vdash A$ holds. The problem of finding such as assumption A would become even more difficult if the systems in question are constrained with an incomplete information. The application of the proof search algorithm of paracomplete logic **Kl** described above would represent an efficient solution. (Of course we would need to introduce the rigorous reasoning here defining what are 'strong' and 'weak' conditions.)

Let us draw here some directions of the application of the presented proof search towards the automation of assume-guarantee technique.

In the reasoning below we rigorously follow the proof search algorithm for $\mathbf{Kl_{ND}}$.

When solving the problem $S_1||S_2 \Vdash P$ we look for the assumption A such that $S_1, A \Vdash P$ and $S_2 \Vdash A$. Assume that S_1 and S_2 are systems with the specifications containing statements $B_1, \ldots, B_m$ and $C_1, \ldots, C_n$, respectively. Our task is to find an assumption A, following rule (†) above, such that $B_1, \ldots, B_m, A \Vdash P$ and $C_1, \ldots, C_n \Vdash A$. In the description of our reasoning, we will use the concept of the algo-step, introduced above. Now we commence $\mathbf{Kl_{ND}}$ proof setting

list_proof $= B_1, \ldots, B_n$ and list_goals $= P$:

step	list_proof	annotation	list_goals
$as0$	1. B_1	given	P
$as1$	.	given	P
$as2$	.	given	P
$as\ m$	m. B_m	given	P
$as\ m+1$	$m+1$.		$P, \bot$

On algo-step m, since the goal P is not reachable, we update list_goals by $\bot$. If on algo-step $m+1$ list_proof contains contradictory elements, then the new goal $\bot$ would be reachable and we would have two contradictory statements within $B_1, \ldots, B_m$, say, C and $\neg C$ at the stages $1 \leq i < j \leq m$. Thus, our new goal would have been P again which we would reach by applying $\mathbf{Kl}_{\neg_{in}}$ rule:

step	list_proof	annotation	list_goals
$as0$	1.	given	P
$as1$	.	given	P
$as3$	i. C	given	P
.	given	P	
$as\ j$. $\neg C$	given	P	
$as\ m$	m.	given	P
$as\ m+1$.		$P, \bot$	
$as\ m+2$.		P	
$as\ m+3$. P	$i,\ j,\ \mathbf{Kl}_{\neg_{in}}$		

Now we found our first candidate for A — contradiction. Hence we set up the new task — $C_1, \ldots, C_n \Vdash \bot$ and thus check if we can establish the latter.

Alternatively, we consider the second case on step m above, when the goal $\bot$ on algo-step m is not reachable. In this case we would have the following continuation of the proof:

step	list_proof	annotation	list_goals
$as0$	1. B_1	given	P
.	.	given	P
.	.	given	P
$as\ m$	m. B_m	given	P
$as\ m+1$	$m+1$. $P \supset r \wedge \neg r$	*assumption*	$P,\ [P \supset r \wedge \neg r]\ P$

At this stage, since P was not reachable, it is not contained in list_proof hence no elimination rules are applicable and we search for new assumptions. Namely, we

would be looking for disjunctive and implicative formulae in list_proof (but ignoring the formula $P \supset r \wedge \neg r$ on step $m+1$).

If successful, we would introduce into the proof the corresponding assumption and proceed further applying the searching algorithm until it terminates with either finding the desired proof for P or failing to do so.

In the former case, P would be the last formula of list_proof and we will be able to consider assumptions appearing in list_proof between algo-step $m+1$ to test them in the second task $C_1, \ldots, C_n \Vdash A$.

7 Conclusion and Roadmap to Future Work

The contribution of this paper is twofold. On one hand, we provided the complexity analysis of the classical natural deduction system and its modified version, for paracomplete logic **Kl**. This has led us closer to the important question on the efficiency of the presented proof search technique and enables us to speak about the second aspect of the contribution of the paper — application issues. We have shown how paracomplete logic **Kl** can be used in providing high level specifications for incomplete systems and how natural deduction system for this logic, supported by the algorithmic proof search, can be used to reason about obtained specifications. To the best of our knowledge, there is no other similar work on the automation of paracomplete natural deduction systems or on an application of natural deduction techniques in general to the reasoning about incomplete specifications.

We have shown how these developments can be integrated into the existing approaches dealing with component-based system assembly.

It is notable that for many researchers, one of the core features of natural deduction, the 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. In this paper, not only we show the contrary, but we also turned the assumptions management into an advantage showing the applicability of the proposed technique to assume-guarantee reasoning.

The results presented in this paper have important methodological aspects forming the basis for the development of automated goal-directed techniques for more expressive formalisms, for example, temporal and normative extensions. The feasibility of these extensions is based on the systematic, generic nature of the natural deduction construction and algorithmic proof search. This will, in turn, enable the application of the powerful natural deduction based reasoning to tackle dynamic systems defined in heterogeneous environments, with such complicated cases as the combinations of time / paraconsitency / paracompleteness. Thus we envisage the

extensions of the applicability of our methodology to the specification of complex dynamic systems, to the specification of normative systems (i.e. protocols) and to reasoning about systems that are both inconsistent and incomplete.

One specific area, where we have obtained some preliminary results, is *Requirements Engineering*. In a series of works authors indicate the importance of the specification of high-level requirements of a partial model such that these specifications are built incrementally from higher-level goal formulations in a way that guarantees their correctness by construction [23]. In [44] the approach to tackle the problem of reduction of complex software requirements to simpler ones and to reason about the requirements is given.

However, we are not aware of any approach which would tackle this task under the following constraints:

(i) considering this problem in the context of incomplete specifications;

(ii) using the advances of automated deduction.

We argue that the natural deduction searching technique, which enables us to trace the dependencies of the formulae in the proof, opens a very important prospect of finding solutions to the above (i) and (ii). The methodology here is as follows: set the formally specified requirements as the goals for the searching technique so the latter returns the set of assumptions upon which these goals depend.

This corresponds to the layer of 'global invariants' mentioned in [23], where the authors give a very reasonable taxonomy of goal patterns (see [23, P.26]).

Now, our solution looks as follows: setting the requirements Req as goals for the proof searching technique, we aim at finding such global invariants.

Thus, applying to each such requirement $r \in Req$ our proof searching algorithm, $\mathbf{Kl}_{\mathsf{NDALG}}$, we aim at finding the assumptions, $Depend(r)$, on which r depends in the proof. This set of formulae $Depend(r)$ represents the desired set of reduced requirements (global invariants).

Acknowledgements

The authors should acknowledge here Prof. Vladimir Popov (Lomonosov Moscow State University), who acquainted them with matrix definitions for **Kl** which we are utilised in this paper.

References

[1] A. Avron. Natural 3-valued Logics — Characterization and Proof Theory. The Journal of Symbolic Logic, Vol. 56(1): 276–294, 1991.

[2] A. Avron and I. Lev. A formula-preferential Base for Paraconsistent and Plausible Non-monotonic Reasoning. In Proceedings of the Workshop on Inconsistency in Data and Knowledge (KRR-4), Int. Joint Conf. on AI (IJCAI 2001), pages 60-70, 2001.

[3] Basic Features of the Grid Component Model Deliverable D.PM.04. CoreGRID, March 2007 (http://coregrid.ercim.eu/mambo/).

[4] D. Basin, S. Matthews, and L. Vigano. Natural deduction for non-classical Logics. Studia Logica, 60(1), (1998): 119–160.

[5] V. Bocharov and V. Markin, Introduction to Logic. Moscow, Higher Education, 2008 (in Russian).

[6] A. Bolotov, O. Grigoriev and V. Shangin: Automated Natural Deduction for Propositional Linear-Time Temporal Logic. Proceedings of TIME 2007: 47–58.

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

[8] A. Bolotov and V. Shangin. Tackling Incomplete System Specifications Using Natural Deduction in the Paracomplete Setting. Proceedings of COMPSAC 2014: 91–96.

[9] A. Bolotov, V. Bocharov, A. Gorchakov and V. Shangin. Automated First Order Natural Deduction. Proceedings of IICAI 2005: 1292–1311.

[10] A. Bolotov and V. Shangin. Natural Deduction in a Paracomplete Setting. Logical Investigations, Vol. 20, (2014): 224–247.

[11] M. Bonet and S. Buss. The Deduction Rule and Linear and Near-Linear Proof Simulations. Journal of Symbolic Logic, Vol. 58/2 (1993): 688-709.

[12] A. Buchsbaum and T. Tarcisio. A Reasoning Method for a Paraconsistent Logic. Studia Logica, 52(2), (1993): 281-290.

[13] W. Carnielli. Systematization of Finite Many-Valued Logics Through the Method of Tableaux. Journal of Symbolic Logic Volume 52, Issue 2 (1987), 473–493.

[14] W. Carnielli. On Sequents and Tableaux for Many-valued Logics. Journal of Non-Classical Logic 8(1), (1991): 59–76.

[15] J. Cobleigh and G. Avrunin and L. Clarke. Breaking Up Is Hard To Do: An Evaluation of Automated Assume-guarantee Reasoning. ACM Transactions on Software Engineering and Methodology 17(2), (2008): 104–155.

[16] V. Degauquier. Partial and Paraconsistent Three-valued Logics. Logic and Logical Philosophy, Volume 25, (2016): 143–171.

[17] A. Hunter and B. Nuseibeh. Managing Inconsistent Specifications: Reasoning, Analysis and Action. ACM Transactions on Software Engineering and Methodology, 7(4), (1998): 335–367.

[18] A. Hunter and S. Parsons. Introduction to Uncertainty Formalisms. Hunter, A and Parsons, S, (eds.) Applications of Uncertainty Formalisms. (pp. 1-7). Springer. Lecture notes in computer science; Vol. 1455, ISBN 3-540-65312-0, Springer, (2001): 1–7

[19] C. Jones. Specification and Design of (parallel) Programs. Proceedings of the IFIP 9th World Congress: IFIP: North Holland, (1983): 321–332.

[20] N. Kamide. Natural Deduction Systems for Nelson's Paraconsistent Logic and its Neigh-

bors. Journal of Applied Non-Classical Logics. Vol. 15 (4), (2005): 405–435.

[21] S. Kleene. Introduction to Metamathematics. Wolters-Noordhoff Publishing and North Holland Publishing Company, Amsterdam, 1971, 7th Edition.

[22] J. Kreiker, A. Tarlecki, M. Vardi, and R. Wilhelm. Modeling, Analysis, and Verification — The Formal Methods Manifesto 2010 (Dagstuhl Perspectives Workshop 10482). Dagstuhl Manifestos, 1(1), (2011): 21–40.

[23] E. Letier and A. van Lamsweerde. Deriving Operational Software Specifications from System Goals. SIGSOFT 2002/FSE-10, Charleston, SC, USA, (2002): 18–22.

[24] C. Middelburg. A Survey of Paraconsistent Logics. The Computing Research Repository (CoRR), vol. 1103/4324, 2011.

[25] A. Naddeo. Axiomatic Framework Applied to Industrial Design Problem Formulated by Paracomplete Logics Approach: the Power of Decoupling on Optimization-Problem solving. Proceedings of Fourth International Conference on Axiomatic Design, (2006): 1-8.

[26] T. Nipkow, L. Paulson and M. Wenzel. Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Springer, 2002.

[27] D. Li. Using the Prover ANDP to Simplify Orthogonality. Annals of Pure and Applied Logic, 124-1, (2003): 49–70.

[28] J. Noppen, P. van den Broek and M. Aksit. Software Development with Imperfect Information. Soft Computing: 12, (2008): 3–28.

[29] B. Nuseibeh, S. Easterbrook, and A. Russo. Making Inconsistency Respectable in Software Development. Journal of Systems and Software, Vol. 58, No. 2, (2001): 171–180.

[30] D. Pastre. Muscadet2.3 : A Knowledge-based Theorem Prover based on Natural Deduction. International Joint Conference on Automated Reasoning - Conference on Automated Deduction (2001): 685—689.

[31] F. J. Pelletier. Natural Deduction Theorem Proving in THINKER. Studia Logica, 60 (1998): 3–43.

[32] F. J. Pelletier. A Brief History of Natural Deduction. History and Philosophy of Logic, 20 (1999): 1–31.

[33] Y. Petrukhin and V. Shangin Automated correspondence analysis for the binary extensions of the logic of paradox. The Review of Symbolic Logic, 1–26 (doi:10.1017/S1755020317000156).

[34] John L. Pollock. Rational Cognition in OSCAR. Agent Theories, Architectures, and Languages (1999): 71—90.

[35] F. Portoraro. Strategic Construction of Fitch-style Proofs. Studia Logica, 60 (1998): 45–66.

[36] A. Pnueli. In Transition from Global to Modular Temporal Reasoning about Programs. Logics and Models of Concurrent Systems, K. R. Apt, Ed. NATO ASI: vol. 13. Springer-Verlag, (1984): 123–144.

[37] V. Popov. Between the logic Par and the set of all formulae in 'The Proceeding of the

6th Smirnov Readings in logic', Contemporary notebooks, Moscow, 93–95 (In Russian).

[38] V. Popov. Between Int$< \omega, \omega >$ and Intuitionistic Propositional Logic. Logical Investigations, Issue 19, 2013, 197–199 (in Russian).

[39] A. Sette and W. Carnielli. Maximal Weakly-intuitionistic Logics. Studia Logica, 55, 1 (1995): 181—203.

[40] W. Quine. On Natural Deduction. The Journal of Symbolic Logic, 2-15 (1950): 93–102.

[41] R. A. Reckhow. On the Lengths of Proofs in the Propositional Calculus. Ph. D. thesis / Reckhow R.A. ; University of Toronto, 1976.

[42] Shangin V.O. A Precise Definition of an Inference (by the example of natural deduction systems for logics $I < \alpha, \beta >$) Logical investigation 23(1), (2017): 83–104

[43] L. V. Tien, Q. T. Tho, and L. D. Anh. Specification-based Verification of Incomplete Programs. ACEEE Int. Journal on Information Technology, Vol. 02, No. 02, (2012): 56–61.

[44] B. Wei, Z. Jin, D. Zowghi, and B. Yin. Automated Reasoning with Goal Tree Models for Software Quality Requirements. Proceedings of COMPSAC 2012 Workshops, (2012): 373–378.

Received 8 February 2017