Tackling Incomplete System Specifications Using Natural Deduction in the Paracomplete Setting

Alexander Bolotov
Vasilyi Shangin

This is a copy of the author's accepted version of a paper subsequently published in the Proceedings of 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC), pp. 91-96. ISBN 9781479935741. It is available online at:

https://dx.doi.org/10.1109%2FCOMPSAC.2014.15

Tackling Incomplete System Specifications Using Natural Deduction in the Paracomplete Setting.

Alexander Bolotov
Department of Computer Science and Software Engineering
University of Westminster, W1W 6UW
Email: A.Bolotov@wmin.ac.uk

Vasilyi Shangin
Department of Logic, Lomonosov Moscow State University,
Moscow, 119991, Russia.
Email: shangin@philos.msu.ru

***Abstract*—In many modern computer applications the significance of specification based verification is well accepted. However, when we deal with such complex processes as the integration of heterogeneous systems, parts of specification may be not known. Therefore it is important to have techniques that are able to cope with such incomplete information. An adequate formal set up is given by so called paracomplete logics, where, contrary to the classical framework, for some statements we do not have evidence to conclude if they are true or false. As a consequence, for example, the law of excluded middle is not valid. In this paper we justify how the automated proof search technique for paracomplete logic PComp can be efficiently applied to the reasoning about systems with incomplete information. Note 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. Here, 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.**

***Keywords* - incomplete information, automated natural deduction, paracomplete logic, requirements engineering, assume-guarantee reasoning, component based system assembly.**

I. Introduction

When we speak about the reasoning tools associated with different software engineering tasks related to modern computer systems we must take into account that these systems are complex, dynamic and heterogenous and often the specifications are not fully given. Consider, for example,

(a) the problem of reduction of complex software requirements to simpler ones in the lack of complete specifications,
(b) a typical integration task of various resources, which could be the problem of formation 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.

To tackle such heterogeneous, incomplete information, so called classical reasoning is not suitable as it validates the famous principle of excluded middle which assumes that truth values of every primitive statement are fully defined. Therefore, there is a need to develop high level specification languages and the corresponding deductive methods suitable in cases where some statements in the specification are not completely defined.

We concentrate on paracomplete logic PComp, [1], [19] where the principle of excluded middle and some corresponding reasoning cases (such as from $A \supset B$ to infer $\neg A \vee B$) are prevented. We restrict our considerations by propositional reasoning without tackling temporal/dynamic aspects. To tackle the reasoning applicable to the problem setting of cases (a) - (c) above, we need efficient deductive techniques. Our reasoning on the selection of existing formalisms and deductive techniques is based upon the following:

(i) Need to keep track on the assumptions, to know which assumptions and how have been involved into the proof.
(ii) Presence of the automated proof search should enable the implementation.
(iii) The deductive techniques and proof search should be generic enough to enable re-use and adaptability to various types of formal specifications, for example, not only to incomplete specifications but also to inconsistent or to those that are both incomplete and inconsistent, and would enable further extensions to richer formalisms, e.g. dynamic systems.

From the point of view of these criteria, we believe that natural deduction represents a very attractive framework. Natural deduction allows not only to establish that there exists a desired proof but also allows for the explicit construction of the proof. Logic PComp has been given natural deduction formulation and algorithmic proof search [9], thus meeting our requirements (i) and (ii). We have not found any other natural deduction formulation of paracomplete setting. Note that most research on paracomplete logic (and paraconsistent logic) is devoted to philosophical problems thus traditionally tackling axiomatic constructions [16] or tableau techniques [4] or sequent constructions [14]. At the same time, in [3] the authors introduce natural deduction systems for paracomplete setting defining the translation techniques to Isabelle, hence no direct deductive method is given.

Moreover, PComp has its dual system, PCont, which tackles dual, inconsistent systems. This logic was also given natural deduction formulation and algorithmic proof search [7], based on the same generic foundation as PComp which meets our criterion (iii). Having in mind that the algorithmic proof search

has the core goal-directed procedure adapted in both cases it also meets our adaptivity element in our requirement (iii).

The main contribution of this paper is justifying how the automated proof search technique for PComp can be efficiently applied to the reasoning about systems with incomplete information.

The rest of the paper is organized as follows. In §II we describe the natural deduction calculus and in §III we describe the proof searching algorithm. In §IV we discuss efficiency of the proof search algorithm. In §V we consider simple scenarios of requirements engineering and component based system synthesis and assume-guarantee technique. We apply paracomplete logic to tackle incomplete specifications and the proof search algorithm for PComp as deductive verification. Finally, in §VI, we provide concluding remarks and identify future work.

II. Natural deduction system $\mathbf{N}PComp$

As the specification framework we utilise the formal language of paracomplete logic PComp. PComp syntax is based upon the propositional syntax with the standard set of Boolean operations. PComp has the matrix semantics with three values 1, f, 0 with the designated value 1 such that $0 < f < 1$ and $A \vee B = max(A, B)$ and $A \wedge B = min(A, B)$. The matrices for $\neg$ and $\supset$ are given below:

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

A	$\neg\ A$
1	0
f	f
0	1

The axiomatics for PComp can be found in [1].

Now, as it is indicative for the natural deduction construction, we define two classes of rules of inference: *elimination* and *introduction* rules in Figure 1.

- $\supset_{in}$ discards each formula starting from the last alive assumption up to the conclusion of this rule;
- $\vee_{el}$ discards each formula starting from assumption A up to formula C, inclusively, as well as each formula starting from assumption B up to formula C, inclusively;
- $\supset_p$ discards each formula starting from assumption $A \supset B$ up to formula A, inclusively. This, as before, is abbreviated by enclosing relevant discharged and discarded formulae into square brackets.

Definition 1 (Inference, Proof): An *inference* in the system NPComp is a finite non-empty sequence of formulae where each formula is an assumption or is derived from the previous ones via a NPComp-rule. A *proof* in the system NPComp is an inference from the empty set of assumptions.

It has been shown [20] that the presented natural deduction calculus is sound and complete, below $\models$ stands for logical consequence:

Theorem 1: $\Gamma \vdash_{\mathbf{N}PComp} A \Leftrightarrow \Gamma \models A$.

Elimination Rules :

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

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

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

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

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

Introduction Rules :

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

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

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

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

$$\supset_p\ \frac{[A \supset B]\quad A}{A} \qquad PComp_{\neg\ in}\ \frac{A,\ \neg A}{B}$$

Fig. 1. NPComp-ND-rules

III. Proof Searching for $\mathbf{N}PComp$

In [9] we defined the proof search technique for the logic PComp. Here we only describe the algorithm behind this search referring an interested reader to [9] for details. The proof search strategy is *goal-directed*, it runs over two sequences: list_proof which lists formulae in the proof and list_goals which lists goals to be reached. Each step of the algorithmic proof is associated with a specific goal, called *current_goal*. Checking the reachability of the current goal is one of the core procedures.

Definition 2 (Current goal reachability): Current goal, G_n, $0 \leq n$, occurring in list_goals= $\langle G_0, G_1, \ldots, G_n \rangle$, is reached if

- G_n is some formula B and there is a formula $A \in$ list_proof such that A is not discarded and $A = B$ or
- G_n is of the form $[A]B$ and there is a list_proof(B^+) such that a non-discarded assumption $A \in$ list_proof and B is the last formula of list_proof.
- G_n is a contradiction and there are two contradictory statements, $A \in$ list_proof and $\neg A \in$ list_proof.

A. Proof-Searching Algorithm NPComp$_{\mathsf{ALG}}$

The proof search algorithm defined below utilises searching procedures Procedure 1 - Procedure 4 defined in [9].

Algorithm NPComp$_{\mathsf{ALG}}$.

list_proof $=$ list_goals $= \emptyset$. Given a task $\Gamma \vdash G$,

(1) $G_{cur} = G$.
$(\Gamma \neq \emptyset) \longrightarrow$ (list_proof $= \Gamma$, list_goals $= G$, *go to* (2)) else
list_goals $= G$, *go to* (2).

(2) $Procedure(3)(G_{cur}) = \mathbf{true}$ (check the reachability of the current goal)
$\forall G_i(0 \leq i) \in$ list_goals $((G_i = G_{cur}) \longrightarrow (Procedure\ (3)(G_i) = \mathbf{true}))$.
(2a) $Reached(G_{cur}) \longrightarrow$ *go to* (3) else
(2b) *go to* (4).

(3) $Procedure(4)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$ (apply relevant introduction rule)
(3a) $(Reached(G_{cur}) = \mathbf{true})AND\ (G_{cur} = G) \longrightarrow$ *go to* $(6a)$ else
(3b) $Procedure\ (4)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$, *go to* 2.

(4) $Procedure\ (1)(\langle$list_proof$\rangle) = \mathbf{true}$ (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) $Procedure\ (2)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$ (update list_proof and list_goals based on the structure of G_{cur})
(5a) $Procedure\ (2.1)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$ (analysis of the structure of G_{cur}), *go to* (2) else
(5b) $Procedure\ (2.2)(\langle$list_proof, list_goals$\rangle) = \mathbf{true}$ (searching for the sources of new goals in list_proof), *go to* (2) else
(5c) (if all compound formulae in list_proof are marked, i.e. have been considered as sources for new goals), *go to* (6b).

(6) Terminate(NPComp$_{\mathsf{ALG}}$).
(6a) The desired ND proof has been found. EXIT,
(6b) No ND proof has been found. EXIT.

IV. Correctness and Efficiency

When we speak about the efficiency of the proof search we consider two aspects of the procedure - its metatheoretical properties (termination and correctness) and its complexity. In this section we discuss these issues.

The following theorems reflect the metatheoretical properties of the above algorithm [9].

Theorem 2: NPComp$_{\mathsf{ALG}}$ terminates for any input formula.

Theorem 2 guarantees that for any input formula for the NPComp$_{\mathsf{ALG}}$ the sequences list_proof and list_goals are finite.

Theorem 3: NPComp$_{\mathsf{ALG}}$ is sound.

Theorem 3 ensures that every formula for which an ND proof is constructed according with NPComp$_{\mathsf{ALG}}$ is valid.

Theorem 4: NPComp$_{\mathsf{ALG}}$ is complete.

Theorem 4 establishes that for every valid formula, A, NPComp$_{\mathsf{ALG}}$ finds a $PComp_{ND}$ proof. Altogether, Theorems 2, 3 and 4 imply the following fundamental property of our algorithm:

Theorem 5: For any input formula A, the NPComp$_{\mathsf{ALG}}$ terminates either building up a $PComp_{ND}$-proof for A or providing a counter-model.

Before discussing the issue of the efficiency let us concentrate on some core and important features of the proof search. First of all, recall that natural deduction calculi can be used to solve different deductive tasks. For example, we might be given any of the following tasks:

1 to find an ND derivation $\Gamma \vdash B$,
2 to find an ND proof $\vdash B$ or
3 to check the consistency of some given set of formulae.

In the first and third cases, the ND derivation would start with some given set of assumptions Γ. In the second case, i.e. when we need to establish if B is a theorem, we commence our reasoning by introducing some assumptions. As in other ND calculi, in constructing an ND derivation, we are allowed to introduce arbitrary formulae as new assumptions. 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 appear in the proof is well justified serving a specific goal. 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 §V.

Secondly, 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 determines the a relevant introduction rule. Also, though the specific for PComp, $Pcomp_{\neg\ in}$ rule, in general, allows to derive any formula from the contradiction, the application of this rule is strictly determined by the searching procedures, hence the formula that we derived from a contradiction is always the one mentioned in list_goals.

In our discussion of the efficiency of the proof search algorithm we utilise the concept of nested deduction Frege systems (ndF) introduced in [10]. The latter was used in the analysis of the computational complexity of the conventional proof systems for propositional classical logic. Following [10], each line in ndF is a sequent of the form $\Gamma \prec A$ where Γ is a set of formulae and A is a formula. Now the inference rules of the system are:

ndF_{R1} $\prec A$, where A is an instance of an axiom,
ndF_{R2} $\{A\} \prec A$ Hypothesis,
ndF_{R3} If $\Gamma \prec A \supset B,\ \Delta \prec A$ then $\Gamma \cup \Delta \vdash B$, modus ponens
ndF_{R4} If $\Gamma \prec B$ then $\Gamma \backslash \{A\} \prec A \supset B$, deduction rule.

Hypothesis are used in the following fashion. First of all, they are open or closed (by the deduction rule). Secondly, they are used in a nested fashion, i.e. they are closed in the order reversed to their opening. Also, any formula in the scope of the closed hypothesis is no longer available for the proof.

It is shown in [10], that Frege System (almost) linearly simulates ndF. Now we observe that our algorithm NPComp$_{\mathsf{ALG}}$ is a modification of the proof search algorithm for classical ND [7], [8] preserving its style, and most of the procedures. Considering our classical proof search technique, it is possible to show that any ndF proof can be transformed into the proof by the classical ND calculus. Indeed, since every axiom of Frege system is provable by the classical ND proof search algorithm, we can substitute each formula in the proof derived by ndF_{R1} by the corresponding ND proof of the introduced axioms. Similarly, it is a routine to show how to transform any proof by rule ndF_{R4} into an ND proof. Rules ndF_{R2} and ndF_{R3} are simply present in the definition of the ND. Note that in all these proofs which we cannot present here due to the room space, we would essentially use the features of the classical proof ND proof search algorithm (and these are the features of NPComp$_{\mathsf{ALG}}$ as well): that our assumptions are discharged in the reverse order to their introduction due to the nested fashion, pictorial form of our presentation which matches the form used in [10].

Therefore, proofs by our search algorithms in the classical case can be linearly simulated by classical Frege style formulation which puts our Fitch-style natural deduction construction in line of propositional complexity.

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

Concluding this section, we note that these general theoretical discussions should be supported by the study of the practical implementation of the NPComp$_{\mathsf{ALG}}$ which forms part of our future work.

V. Applications in Specification Based Verification

Our development of 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 to the areas listed below:

1) Requirements Engineering.
2) Component Based System Assembly.

In the subsequent sections we will justify the application of the natural deduction to these fields.

A. 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 [15]. In [5] 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 using the advances of automated deduction.

Our searching technique enables to trace the dependencies of the formulae in the proof hence an obvious way of applying it would be to put the specified requirements as the goals for the searching technique so the latter returns the set of assumptions upon which this goal depends. This corresponds to the layer of 'global invariants' mentioned in [15], where the authors give a very reasonable taxonomy of goal patterns (see page 26 of [15]).

Now, our solution looks as follows: setting the requirements Rec 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, $\mathbf{N}PComp_{\mathsf{ALG}}$, 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).

B. Component Based System Assembly

Here we apply the searching algorithm $\mathbf{N}PComp_{\mathsf{ALG}}$ as the deductive verification technique for a component system. We incorporate the notation of [22] to represent the task of deductive verification of a system Sys by the following signature:

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

where the boolean result of deductive verification based on theorem proving is either a proof that a system satisfies a given property or a proof cannot be established.

As an example, let us consider a simple component system interpreted in The Grid Component Model (GCM) based on Fractal [12]. 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 above verification task to establish if the above configuration of components is consistent. We commence the proof (see below) by the given $Spec$ conditions and set up the goal of the procedure to derive the contradiction, abbreviated in the proof annotation below as $\perp$. 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.

list_proof	*annotation*	list_goals
1. $p \supset r$	*given*	$\perp$
2. $\neg p \supset s$	*given*	$\perp$
3. $q \supset (\neg r \wedge \neg s)$	*given*	$\perp$
4. q	*given*	$\perp$
5. $\neg r \wedge \neg s$	$3, 4 \supset$ *el*	$\perp$
6. $\neg r$	$5, \wedge$ *el*	$\perp$
7. $\neg s$	$5, \wedge$ *el*	$\perp$
		$\perp, p$
8. $p \supset (t \wedge \neg t)$	*assumption*	$\perp, p, p$
		$\perp, p, p, \neg p$
9. $\neg p \supset (u \wedge \neg u)$		$\perp, \neg p, \neg p$

At this moment, the procedure stops. A counter-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_proofand s is assigned 0 because $\neg s$ is in the list_proof. Under this valuation, each formula $p \supset r, \neg p \supset s, q \supset (r \wedge s)$ and q is assigned 1. So, this set of formulae in $Spec$ is consistent.

This explicitly shows the nature of the use of paracomplete logic - the given $Spec$ does not have precise information about p - if this component should be bound or not.

C. Assume-guarantee reasoning

We consider here how the natural deduction based reasoning can be applied in the automation of the assume-guarantee reasoning [13], [18] 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 that 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 † in the Figure 2. 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 ‡ in Figure 2.

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

Fig. 2. NPComp in Assume Guarantee Reasoning

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 [11] 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 incomplete information. The application of the proof search algorithm of the paracomplete logic PComp 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 PComp. 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 $B_1, \ldots, B_m$ $C_1, \ldots, C_n$, respectively. Our task is to find an assumption A, following rule (†) above, such that $B_1, \ldots, B_m, A \vdash P$ and $C_1, \ldots, C_n \vdash A$. Now we commence **N**$PComp$ proof setting list_proof= $B_1, \ldots, B_n$ and list_goals= P:

list_proof	*annotation*	list_goals
1. B_1	*given*	P
$\cdot$	*given*	P
$\cdot$	*given*	P
m. B_m	*given*	P
$m+1$.		$P, \perp$

At step m since the goal P is not reachable, we update list_goals by $\perp$. If at step $m+1$ list_proof contains contradictory elements then the new goal $\perp$ 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 PComp$_{\neg\ in}$ rule:

list_proof	*annotation*	list_goals
1.	*given*	P
.	*given*	P
$i.\ C$	*given*	P
.	*given*	P
$j.\ \neg C$	*given*	P
$m.$	*given*	P
$m+1.$		$P, \bot$
$m+2.$		P
$m+3.\ P$	$i,\ j,\ PComp_{\neg\ in}$	

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 at step m above, when the goal $\bot$ at step m is not reachable. In this case we would have the following continuation of the proof:

list_proof	*annotation*	list_goals
$1.\ B_1$	*given*	P
.	*given*	P
.	*given*	P
$m.\ B_m$	*given*	P
$m+1. P \supset r \wedge \neg r$	*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 $P \supset r \wedge \neg r$ at 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 step $m+1$ to test them in the second task $C_1, \ldots, C_n \vdash A$.

VI. Conclusion and future work

We have shown how the paracomplete logic PComp can be used 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 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 the requirements reduction and component based system assembly. 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 deductive 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.

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] D. Basin, S. Matthews, and L. Viganò. Natural deduction for non-classical logics. *Studia Logica*, 60(1):119–160, 1998.

[4] A. Buchsbaum and T. Tarcisio. A reasoning method for a paraconsistent logic. Studia Logica, 52(2), 1993.

[5] Bo Wei, Zhi Jin, Didar Zowghi, and Bin Yin. Automated reasoning with goal tree models for software quality requirements. In *COMPSAC Workshops*, pages 373–378, 2012.

[6] A. Bolotov, O. Grigoriev and V. Shangin: Automated Natural Deduction for Propositional Linear-Time Temporal Logic. 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, N1, 2012, pp 1-24.

[8] A. Bolotov, V. Bocharov, A. Gorchakov and V. Shangin. *Automated First Order Natural Deduction.* IICAI 2005: 1292-1311

[9] A. Bolotov and V. Shangin. *Natural Deduction in a Paracomplete Setting.* Forthcoming in Logical Investigations, Vol. 20, 2014.

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

[11] J. Cobleigh and G. Avrunin and L. Clarke. *Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Trans. Softw. Eng. Methodol.*: 17(2), 2008.

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

[13] C. Jones. *Specification and design of (parallel) programs. Proceedings of the IFIP 9th World Congress*: IFIP: North Holland, 321332, 1983.

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

[15] E. Letier and A. van Lamsweerde *Deriving Operational Software Specifications from System Goals. SIGSOFT 2002/FSE-10, Charleston, SC, USA*, pages 18-22, 2002.

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

[17] 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, pages 1-8.

[18] A. Pnueli. In transition from global to modular temporal reasoning about programs. *In Logics and Models of Concurrent Systems, K. R. Apt, Ed. NATO ASI*: vol. 13. Springer-Verlag, 123144, 1984.

[19] V. Popov. *Sequence axiomatisation of simple paralogics. Logical Investigations*, Issue 16, 2010, pp 205-220, (in Russian).

[20] V. Shangin. *Natural deduction systems of some logics with truth-value gluts and truth-value gaps. Logical investigation*, StPetersburg, CGI, 2011: 293-308, (in Russian).

[21] Le V. Tien, Quan T. Tho, and Le D. Anh. *Specification-based Verification of Incomplete Programs. ACEEE Int. J. on Information Technology*, Vol. 02, No. 02, April 2012, pages 56-61.

[22] Jörg Kreiker, Andrzej Tarlecki, Moshe Y. Vardi, and Reinhard Wilhelm. *Modeling, Analysis, and Verification - The Formal Methods Manifesto 2010 (Dagstuhl Perspectives Workshop 10482). Dagstuhl Manifestos*, 1(1):21–40, 2011.