Automated first order natural deduction

Bolotov, A., Bocharov, V., Gorchakov, A. and Shangin, V.

A paper published in the proceedings of the 2nd Indian International Conference on Artificial Intelligence, Pune, India, December 20-22, 2005. IICAI 2005 ISCAI. pp. 1292-1311.

Automated First Order Natural Deduction

Alexander Bolotov*, Vyacheslav Bocharov**, Alexander Gorchakov** and Vasilyi Shangin**

* Harrow School of Computer Science
University of Westminster
Watford Road, Harrow HA1 3TP, UK.
A.Bolotov@wmin.ac.uk
http://www2.wmin.ac.uk/bolotoa/index.html
** Department of Logic, Moscow State University, Moscow, 119899, Russia.
{shangin,gorchakov}@philos.msu.ru

Abstract. We present a proof-searching algorithm for the classical first order natural deduction calculus and prove its correctness. For any given task (if this task is indeed solvable), a searching algorithm terminates, either finding a corresponding natural deduction proof or giving a set of constraints, from which a counter-example can be extracted. Proofs of the properties which characterize correctness of the searching algorithm are given. Based on a fully automatic goal-directed searching procedure, our technique can be efficiently applied as an automatic reasoning tool in a deliberative decision making framework across various AI applications.

1 Introduction

Natural deduction calculi (abbreviated in this paper by 'ND') originally were developed in the mid-thirties by Gentzen [11] and Jaskowski [13] and we speak of a natural deduction of G-style and J-style, correspondingly. In this paper we concentrate on the latter approach where the construction of a proof is given as a *synthetic* procedure.[1] Jaskowski-style natural deduction was improved by Fitch [9] and simplified by Quine [17].

Further development of such systems was controversial. On the one hand, there has been an obvious interest in these ND formalisms as they represent a 'natural' way of reasoning. On the other hand, ND was often considered as inappropriate for an algorithmic representation [10], even in the simplest case of classical propositional logic. This scepticism is not surprising because the application of *introduction rules* allows us to introduce into the proof arbitrary formulae. In other words, an introduction rule of this type violates *the subformula property*, which requires that in a derivation, any formula which occurs in the conclusion of a rule, is a subformula of its premises.

As a consequence, ND systems of this type have been primarily studied within the framework of philosophical logic, being widely used in teaching (but again,

[1] We consider sequent, or Gentzen style, calculi as purely *analytic* since they are based upon purely analytic procedures.

mostly in philosophy curriculum) and have been ignored by the automated theorem-proving community, where research has been mostly concentrated on purely *analytic* methods such as *resolution* and *tableau* based approaches [1].

Recently, ND systems have been studied within a wider community. One of the most recent examples of the interest in natural deduction is the areas of *logical frameworks* [15], where the notion of *hypothetical* judgements, i.e.reasoning from hypothesis, as in the natural deduction, is essential. Here, in particular, ND systems have been developed for intuitionistic linear logic [16].

In this paper we present a proof searching algorithm for the ND system of classical first order logic (FOL) and establish its correctness. Our ND-based theorem proving engine is based upon a *goal-directed* searching procedure. We believe that this makes it relevant in the areas of simulation of a deliberative decision making process. Additionally, since our procedure is fully automatic, if refined, it can be potentially applied as an efficient reasoning tool in modelling complex knowledge based systems.

It is now commonly accepted [8] that if we think about possible practical use of an ND system, we would significantly benefit if the particular ND calculus is built *systematically*. In other words, it must be built in such a way that the technique developed would allow us to cover a number of logics by a manipulation of the ND rules.

In our analysis of the ND calculi, we follow this systematical approach. The particular ND-calculus we are interested in is described in detail in [5]. It is a modification of Quine's representation of subordinate proof [17] developed for classical propositional and first-order logic. It has been recently extended to intuitionistic logic [14]. The proof-searching strategies for propositional intuitionistic logic [5], which have been recently developed, are based upon the proof-searching strategies for classical propositional natural deduction calculus [3]. In this paper not only do we extend our approach to FOL and present an improved version of the algorithm, but more importantly, bridge the gap contained in [4] justifying the correctness of the proposed system [18], [5].

We believe that the goal-directed nature of our proof searching technique opens broad perspectives of application of these techniques in many AI areas, and first of all, in agent engineering [20]. Moreover, though the complexity of our proof searching techniques is still an open problem, we know that the complexity is not a function of the length of the input but of the **type** of the input. In other words, once a proof for some input, say, for a formula $((p \supset q) \supset p) \supset p$ (famous Peirce law) has been obtained, its length **will not increase** for a longer input of the same type, $((p \supset (q \wedge r)) \supset p) \supset p$. These observations will be clearer when the reader becomes familiar with our developments and we will return to these problems in conclusion.

The paper is organized as follows. In §2 we review a natural deduction system for classical FOL. In §3 we describe the proof-searching technique giving several underlying searching strategies in §3.1 and the algorithm itself in §3.2. Subsequently, in §4, we establish correctness of the algorithm and in §5 we demonstrate

its application. Finally, in §6, we provide concluding remarks and identify future work.

2 Natural Deduction System

Here we overview the underlying natural deduction system, following [2].[2] To simplify the presentation we consider a standard formulation of first-order logic without functional symbols and equality. The only technical agreement we enforce is that negation is the strongest logical connective.

Notation

- By a *literal* we understand a proposition or its negation.
- By an *elementary quantifier free formula* we understand a literal or a formula of the type $P^n(t_1, \ldots, t_n)$, where P^n is a n-ary predicate and $t_1, \ldots, t_n$ are terms (i.e. individual variables or constants).
- We will use the symbols '$\vdash$' and '$\models$' as follows. By writing $\Gamma \vdash B$ we mean a task to establish a natural deduction derivation of a formula B from a set of assumptions Γ. If Γ, in $\Gamma \vdash B$, is empty then the task is to prove that B is a theorem, and in this case we will simply write $\vdash B$. The abbreviation $\Gamma \models B$ stands for establishing that B is a logical consequence of a set of assumptions Γ. If Γ, in $\Gamma \models B$, is empty then the task is to show that B is a valid formula and in this case we will simply write $\models B$.

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

Specifically for an ND calculus, in constructing an ND derivation, we are allowed to introduce arbitrary formulae as new assumptions. Consequently, any formula in a derivation is either an assumption or a formula which is obtained as a result of the application of one of the inference rules.

Further, the set of rules is divided into the two classes: *elimination* and *introduction* rules. Rules of the first group allow us to simplify formulae to which they are applied. These are rules for the 'elimination' of logical constants. Rules of the second group are aimed at 'building' formulae, introducing new logical constants. In Figure 1 we define sets of elimination and introduction rules, where prefixes '*el*' and '*in*' abbreviate an elimination and an introduction rule, respectively.

- In the formulation of the rules '$\supset$ *in*' and '$\neg$ *in*' the formula C must be the most recent non discarded (see below) assumption occurring in the proof.
- In the formulations of the rules for the quantifiers $A(t_1/t_2)$ abbreviates the result of the correct substitution of any occurrence of a variable t_1 by the term t_2 in formula A.

[2] This natural deduction system has, for many years, been a subject of the main undergraduate logic course taught at the Faculty of Philosophy, Moscow State University and at the Harrow School of Computer Science, the University of Westminster.

Elimination Rules :		Introduction Rules :	
$\wedge\ el_1$	$\dfrac{A \wedge B}{A}$	$\wedge\ in$	$\dfrac{A, \quad B}{A \wedge B}$
$\wedge\ el_2$	$\dfrac{A \wedge B}{B}$	$\vee\ in_1$	$\dfrac{A}{A \vee B}$
$\vee\ el$	$\dfrac{A \vee B \quad \neg A}{B}$	$\vee\ in_2$	$\dfrac{B}{A \vee B}$
$\supset\ el$	$\dfrac{A \supset B \quad A}{B}$	$\supset\ in$	$\dfrac{B}{C \supset B}$
$\neg\ el$	$\dfrac{\neg\neg A}{A}$	$\neg\ in$	$\dfrac{B \quad \neg B}{\neg C}$
$\forall\ el$	$\dfrac{\forall \alpha A(\alpha)}{A(\alpha/t)}$	$\forall\ in$	$\dfrac{A(\alpha/\beta, \gamma_1, \ldots, \gamma_n)}{\forall \alpha A(\alpha, \gamma_1, \ldots, \gamma_n)}$
$\exists\ el$	$\dfrac{\exists \alpha A(\alpha, \gamma_1, \ldots, \gamma_n)}{A(\alpha/\beta, \gamma_1, \ldots, \gamma_n)}$	$\exists\ in$	$\dfrac{A(\alpha/t)}{\exists \alpha A(\alpha)}$

Fig. 1. ND-rules

– In rules $\forall\ in$ and $\exists\ el$, variable β is absolutely flagged and any free variables $\gamma_1, \ldots, \gamma_n$ in $\forall \alpha A(\alpha)$ or $\exists \alpha A(\alpha)$ are considered as relatively flagged (relatively bounded) variables.

Definition 1 (ND-derivation). *An* ND-derivation *of a formula B from a (possibly empty) set of assumption Γ, abbreviated by $\Gamma \vdash_{ND} B$, is a finite sequence of formulae $A_0, A_1, \ldots, A_n$, abbreviated as* list_proof *which satisfies the following conditions:*

(i) *Every A_i $(0 \leq i \leq n)$ is either an assumption, a member of Γ, or the conclusion of one of the ND-rules applied to some formulae which occur in the derivation before A_i.*
(ii) *If A_m $(0 < m \leq n)$ is a conclusion of the application of either of the following ND rules: '$\supset$ in' or '$\neg$ in', and A_k $(0 \leq k < m)$ is the most recent assumption of the derivation, then A_k and all formulae from A_k to A_{m-1}, inclusively, are* discarded.
(iii) *No variable is absolutely flagged twice or relatively binds itself.*
(iv) *The last formula, A_n, in the sequence* list_proof *is identical with B.*

The condition (ii) in the definition above means that if an assumption has been discarded then every formula which depends on this assumption (including the assumption itself) is also considered as discarded, i.e. it can not be involved in any further derivation.

Definition 2 (Completed ND-derivation). *An ND-derivation of a formula B from a set of assumptions Γ is* completed *if, and only if, no absolutely flagged variable occurs free in Γ and B.*

Definition 3 (ND-Proof). *An* ND-proof *is a completed ND-derivation where the set of non-discarded assumptions is empty. The last formula of a proof is a* theorem.

In our graphical representation of the subordinate proof we adopt the technique where a hierarchy of boxes is represented by the nesting of square brackets which are written on the left hand side of the derivation.

The following theorems establish meta-theoretical properties of the ND system defined above. The corresponding proofs can be found in [5].

Theorem 1 (ND Soundness). [5] *Let $B_1, B_2, \ldots, B_n$ be a completed ND derivation and let Γ be a set of non-discarded assumptions which occur in this derivation. Then the following propositions are valid.*

- *If $\Gamma \neq \emptyset$ then $\Gamma \models_{ND} B_n$.*
- *If $\Gamma = \emptyset$ then $\models_{ND} B_n$, i.e. B_n is a valid formula.*

Theorem 2 (ND Completeness). [5] *For any set of FOL, Γ, and any formula, G, if $\Gamma \models G$ then there exists $\Gamma \vdash G$, a completed ND derivation of G from the assumptions Γ.*

Example. In Figure 2 we illustrate the ND-derivation, providing the proof for the following formula

$$\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x)) \tag{1}$$

0. $\forall x(P(x) \wedge Q(x))$	assumption
1. $P(y_1) \wedge Q(y_1)$	$\forall$ *el*, 0
2. $P(y_1)$	$\wedge$ el_1, 1
3. $\forall x P(x)$	$\forall$ *in*, 2, y_1 flagged
4. $P(y_2) \wedge Q(y_2)$	$\forall$ *el*, 0
5. $Q(y_2)$	$\wedge$ *el*, 4
6. $\forall x Q(x)$	$\forall$ *in*, 5, y_2 flagged
7. $\forall x P(x) \wedge \forall x Q(x)$	$\wedge$ *in*, 3,6
8. $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$	$\supset$ *in*, 7

Fig. 2. ND-proof of $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$

Here, taking into account the structure of the given formula, $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$, we aim to derive its consequent, $\forall x P(x) \wedge \forall x Q(x)$,

from its antecedent, $\forall x(P(x) \wedge Q(x))$. Now, as our goal is a conjunctive formula, we are trying to derive both conjuncts, $\forall x P(x)$, and $\forall x Q(x)$. Thus, step 1 is obtained from 0 by eliminating the $\forall$ quantifier. Step 2 is derived by eliminating conjunction from 1. Introducing $\forall$ quantifier to formula 2 we deduce 3 and variable y_1 becomes absolutely flagged (in our comments to the proof we simply write "y_1 flagged"). Subsequent steps, 4-7, are analogous to 1-3: again we eliminate $\forall$ from 0 deriving 4, then eliminating conjunction from the latter we get 5, and introducing the $\forall$ obtain 6, provided that another variable, y_2 is absolutely flagged. At step 7 we introduce conjunction to 3 and 6, and, finally, introduce implication to 7 deriving the desired formula (1).

3 ND-proof Searching Strategy

The proof searching strategy is *goal-directed*. The core idea behind it is the creation of the two sequences of formulae: *list_proof* and *list_goals*. The first sequence represents formulae which form a derivation (see Definition 1). In the second sequence we keep track of the list of goals. Here, each goal is either a formula or two arbitrary contradictory formulae. We will abbreviate this designated type of goal by **false**. An *algo-derivation*, $\mathrm{ND}_{\mathrm{alg}}$, is a pair (*list_proof*, *list_goals*) whose construction is determined by the searching procedure described below. On each step of constructing an $\mathrm{ND}_{\mathrm{alg}}$, a specific goal is chosen, which should be reached at the *current stage*. Thus, the appropriate name for such a goal would be a *current_goal*. The first goal of *list_goals* is extracted from the given task, we will refer to this goal as to the *initial goal*.

Definition 4 (Reachability of a current goal). *A current goal,* G_n, $0 \leq n$, *occurring in* list_goals$= \langle G_1, G_2, \ldots, G_n \rangle$, *is reached if the following conditions are satisfied:*

- *If* $G_n \neq$ **false** *then* G_n *is reached if there is a formula* A *in* list_proof *such that* A *is not discarded and there exists a unification* σ *such that* $\sigma(G_n, A)$, ELSE
- G_n *is* **false** *and it is reached if there are formulae* A *and* $\neg B$ *in* list_proof, *they are not discarded and there exists a unification* σ *such that* $\sigma(A, B)$.

In general, when we construct a derivation, we check whether the *current_goal* has been reached. If it has been reached then we apply the appropriate introduction rule, and this is *the only reason* for the application of introduction rules. As we will see later, such an application of an introduction rule is absolutely *determined* by the structure of the previous goal and by the formulation of introduction rules. Alternatively, (if the *current_goal* has not been reached), we continue searching for how we can update *list_proof* and *list_goals*. Note that the construction of these sequences is determined either by the structure of the *current_goal*, or by the structure of complex formulae occurring within *list_proof*. Additionally, we introduce a mechanism of marking formulae within *list_proof* and *list_goals* to indicate formulae that have been already analyzed as sources for new goals and new formulae in *list_proof*.

Now we describe a set of *procedures* which guide the construction of an algo-proof.

3.1 Proof-Searching Procedures

Procedure 1. This procedure updates a sequence *list_proof* by searching (in a breadth-first manner) for an applicable elimination ND-rule. If it finds in *list_proof* a formula, or two formulae, which can serve as premises of an elimination ND-rule, this rule is enforced and the sequence *list_proof* is updated by the conclusion of this rule. Here the elimination rules apply in the following order: any rule to eliminate a Boolean operation, $\exists$ *el*, and, finally, $\forall$ *el*.

Procedure 2. Here a new goal is synthesized in a backward chaining style. This procedure applies when Procedure 1 terminates, i.e. when no elimination ND-rule can be applied, and the current goal, G_n, is not reached. The type of G_n determines *how* the sequences *list_proof* and *list_goals* must be updated. In fact, we have here two sub-procedures.

Procedure 2.1. This sub-procedure is invoked when G_n is not **false**. Here the structure of the current goal, G_n, tells us which formulae must be added into the sequence *list_proof* and which goals into the sequence *list_goals*. In general, this procedure applies as follows. Suppose that *list_proof*$= P_1, \ldots P_k$ and *list_goals*$= G_1, \ldots, G_n$, where G_n is the current goal. If it is impossible, at the present stage, to infer G_n then, looking at its structure, we derive a new goal G_{n+1} and set the latter as the current goal. Below we identify various cases of applying sub-procedure 2.1, where $G_n = F|\neg A|A \wedge B|A \vee B|A \supset B|\forall \alpha A|\exists \alpha A$, and F is an elementary quantifier-free formula and A, B are any formulae. Those rules which require additional comments are marked by $\star$. Let $P_1, \ldots P_k = \Gamma$ and $G_1, \ldots, G_{n-1} = \Delta$.

$$
\begin{array}{rll}
(2.1.1)\ \Gamma \vdash \Delta, F & \longrightarrow & \Gamma, \neg F \vdash \Delta, F, \mathbf{false} \\
(2.1.2)\ \Gamma \vdash \Delta, \neg A & \longrightarrow & \Gamma, A \vdash \Delta, \neg A, \mathbf{false}^{\star} \\
(2.1.3)\ \Gamma \vdash \Delta, A \wedge B & \longrightarrow & \Gamma \vdash \Delta, A \wedge B, B, A \\
(2.1.4.1)\ \Gamma \vdash \Delta, A \vee B & \longrightarrow & \Gamma \vdash \Delta, A \vee B, A \\
(2.1.4.2)\ \Gamma \vdash \Delta, A \vee B & \longrightarrow & \Gamma \vdash \Delta, A \vee B, B^{\star\star} \\
(2.1.5)\ \Gamma \vdash \Delta, A \supset B & \longrightarrow & \Gamma, A \vdash \Delta, A \supset B, B \\
(2.1.6)\ \Gamma \vdash \Delta, \forall \alpha A & \longrightarrow & \Gamma \vdash \Delta, \forall \alpha A, A(\alpha/\beta)^{\star\star\star} \\
(2.1.7)\ \Gamma \vdash \Delta, \exists \alpha A & \longrightarrow & \Gamma \vdash \Delta, \exists \alpha A, A(\alpha/\gamma)^{\star\star\star\star}
\end{array}
$$

where

- $\star$ A can be a literal, disjunction, or an $\exists$ quantified formula.
- $\star\star$ searching rule (2.1.4.2) applies when rule (2.1.4.1) fails, i.e. when applying rule (2.1.4.1), we have not managed to reach the left disjunct of the goal $A \vee B$, in which case the subroutine invoked into this attempt is deleted and rule (2.1.4.2) is fired.
- $\star\star\star$ in the rule (2.1.6) β is an absolutely flagged variable and for any free variable δ in $\forall \alpha A$, δ is marked as relatively flagged in any formula of the *list_proof*.

$\star\star\star\star$ in the rule (2.1.7) γ is not an absolutely flagged variable.

Procedure 2.2. This sub-procedure is invoked when G_n is **false**. It searches for complex formulae in the sequence *list_proof* which can serve as a source for new goals. If such a formula is found then its structure will determine the new goal to be generated. Below by Γ, Ψ we understand a list of formulae in *list_proof* with the designated formula Ψ which is considered as a source for new goals. Obviously, Ψ can have the structure of $\neg A$, $A \vee B$ or $A \supset B$.

(2.2.1) $\quad \Gamma, \neg A \vdash \Delta, \mathbf{false} \longrightarrow \quad \Gamma, \neg A \vdash \Delta, \mathbf{false}, A$
(2.2.2) $\Gamma, A \vee B \vdash \Delta, \mathbf{false} \longrightarrow \Gamma, A \vee B \vdash \Delta, \mathbf{false}, \neg A$
(2.2.3) $\Gamma, A \supset B \vdash \Delta, \mathbf{false} \longrightarrow \Gamma, A \supset B \vdash \Delta, \mathbf{false}, A$

Applying the rule (2.2.1) we have $\neg A$ in the proof and are aiming to derive, A itself. If we are successful then this would give us a contradiction. When we apply rule (2.2.2), the proof already contains $A \vee B$ and our target is to derive $\neg A$. If we are successful then we would be able to derive B by $\vee$ *el* rule. Similarly, applying rule (2.2.3) we already have $A \supset B$ in the proof and we aim at deriving A as this would enable us to apply the $\supset$ *el* rule. Note also that the selection of new goals determined by Procedure 2.2 is directly linked to the fact that an exhausted algo-derivation forms a Hintikka set [12] (see also §4 for details).

Procedure 3. This procedure checks reachability of the current goal in the sequence *list_goals*. If, according to Definition 4, the current goal G_n is reached then the sequence *list_goals* is updated by deleting G_n and setting G_{n-1} as the current goal.

Procedure 4. Procedure 4 indicates that one of the introduction ND-rules, i.e. a rule which introduces a logical connective or a quantifier, must be applied. We will see below that any application of the introduction rule is completely determined by the current goal of the sequence of goals. This property of our proof searching technique protects us from inferring an infinite number of formulae in *list_proof*.

Procedure 5 – Unification. We adopt the unification algorithm from Chang-Lee [6, 18]. This unification algorithm analyzes formulae position from left to right (as usual) and prevents a variable from binding itself (which is not a usual feature determined by the properties of the calculus in question). It is not sufficient for the proof searching to find a unification. Additionally, this underlying substitution should not violate clause (iii) of Definition 1. That is why a procedure for checking a substitution for preserving this property is incorporated in our unification algorithm.

The unification algorithm is shown to be complete and always finds the most general unifier. If a formula in *list_proof* unifies with the current goal of a sequence of goals, this goal is deleted from a sequence of goals and the previous goal (if any) becomes the current goal. If the initial goal is reached we obtain the desired ND-inference.

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

3.2 Proof-Searching Algorithm

First, let us explain, schematically, the performance of the proof-searching algorithm by describing its major components. These components correspond to the searching procedures presented in §3.1.

Given a (decidable) task $\Gamma \vdash G$, where Γ is possibly empty, we already have the first goal, $G_0 = G$, which is the initial goal and the current goal at the same time. At this stage we apply Procedure 3, checking if G_0 is reached. Assume that G_0 is not reached. Then we apply Procedure 1, obtaining all possible conclusions of the elimination ND rules checking if G_0 can be reached in this way. If we fail, then Procedure 2 is invoked, and, dependent on the structure of the goals G_0, then G_1, etc the sequence *list_proof* is updated by adding new assumptions and the sequence *list_goals* by adding new goals. Note, that each time we add new formulae to *list_proof*, we check if we can reach the *current_goal* by applying elimination rules. If the *current_goal* is reached, then we determine which introduction rules are to be applied. Suppose, however, that we still have no luck. This could only be in the case, when *current_goal* is set as **false** and we do not have contradictory formulae in *list_proof*. Now we update *list_goals* looking for possible sources of new goals in the complex formulae in *list_proof*. We continue searching until either we reach the initial goal, G_0, in which case we terminate having found the desired derivation, or until *list_proof* and *list_goals* cannot be updated any further. In the latter case we terminate, and no derivation has been found and a (finite) counterexample can be extracted. Note again, that this termination of the algorithm is guaranteed for any given decidable task. However, it is obvious that this procedure might not terminate due to the undecidability of FOL.

Below we formulate the main stages of the proof-searching algorithm referring the reader to [18, 6].

We define a technique to introduce and to eliminate marks for formulae in *list_proof* and *list_goals*. Most of these special marking schemes are devoted to prevent looping either in application of elimination rules or in searching. The aims of marking are:

- to keep track of formulae that were used as premisses of the elimination rules invoked in procedure 1,
- to keep track of formulae in *list_proof* that were considered as sources of new goals when procedure 2.1 applies,
- to deal with specific cases of goals, such as, for example, the case 2.1.4: here, before applying sub-procedure 2.1.4.1 we mark the goal $A \vee B$ and if both this sub-procedure and the subsequent 2.1.4.2, fail then we delete from memory part of the algo-proof starting from the goal $A \vee B$, set up a new assumption $\neg(A \vee B)$ and a new goal, **false**. Similar marking is provided for formulae of the type $\exists x A(x)$ occurring in *list_proof*.
- to inform the searching algorithm that no more elimination rules are applicable.

Formulation of the algorithm.

(1) Given a task $\Gamma \vdash G$, we consider G as the *initial goal* of the derivation and write G into *list_goals*. If the set of given assumptions in Γ is not empty then these assumptions are written in a *list_proof*. Set *current_goal* $= G$, *go to* (2).
(2) Analysis of the reachability of the current goal, G_n, and the applicability of elimination rules, (see Procedure 3 described in §3.1).
  (2a) If G_n is reached then *go to* (3) *ELSE*
  (2b) $*$ (if elimination rules are applicable) *go to* (4) *ELSE*
    $*$ (if no more elimination rules are applicable) *go to* (5).
(3) Based on the structure of the goal reached
  (3a) If G_n (reached) is the initial goal then terminate, the desired ND proof has been found, EXIT, *ELSE*
  (3b) (G_n is reached and it is not the initial goal). Apply Procedure 4 (which invokes introduction rules), *go to* 2.
(4) Apply Procedure 1 (which invokes eliminations rules), *go to* (2).
(5) Apply Procedure 2.
  (5a) If $G_n \neq$ **false** then apply Procedure 2.1 (analysis of the structure of G_n), *go to* (2) *ELSE*
  (5b) Apply Procedure 2.2 (searching for the sources of new goals in *list_proof*), *go to* (2) *ELSE*
  (5c) (if all formulae in *list_proof* are marked, i.e. have been considered as sources for new goals), *go to* (6).
(6) Terminate (see comment below). No ND proof has been found. EXIT.

Recall that the termination is guaranteed for any decidable input, and for these examples the algorithm would reach its termination state (3a) (success) or (6) (failure). On the other hand, for some inputs, nothing prevents us of an infinite looping through different stages of the algorithm never reaching these termination stages.

4 Correctness of the Algorithm

In this section we will sketch proofs of the main theorems which characterize the correctness of the $\mathrm{ND}_{\mathrm{alg}}$ algorithm, namely, its *soundness* and *completeness*.

Before establishing soundness and completeness properties of the $\mathrm{ND}_{\mathrm{alg}}$, we introduce a notion of the exhausted $\mathrm{ND}_{\mathrm{alg}}$.

Definition 5. *An algo-derivation is* exhausted *if none of the elimination ND rules can be applied and none of the formulae in the sequence* list_proof *can serve as the source of a new goal.*

It is easy to see that in the exhausted derivation $\mathrm{ND}_{\mathrm{alg}} =$ (*list_proof*, *list_goals*) built for the task $\Gamma \vdash G$, if $\mathrm{ND}_{\mathrm{alg}}$ has not been found, then the last goal in the sequence *list_goals* is always **false**.

Theorem 3. *Let $A_1, A_2, \ldots, A_n$ be a derivation constructed following the proof-searching algorithm, and let Γ be the set of non-discarded assumptions in this derivation. Then $\Gamma \models_{ND} A_n$.*

Proof. Theorem 3 follows from Theorem 1 (soundness of the ND system). *(End)*

For the proof of completeness theorem we need the following lemma.

Lemma 1. *Let Σ abbreviate a set of non-discarded formulae in* list_proof*, $T(\Sigma)$ abbreviate the set of all terms occurring in formulae of* list_proof*. Then in an exhausted algo-derivation obtained for a task $\Gamma \vdash G$, the set, Σ, forms a Hintikka set [12], i.e. for any A, $B \in \Sigma$ the following conditions are satisfied:*

(1) *no formula A and $\neg A$ both occur in Σ;*
(2) *if $\neg\neg A \in \Sigma$ then $A \in \Sigma$;*
(3) *if $A \wedge B \in \Sigma$ then $A \in \Sigma$ and $B \in \Sigma$;*
(4) *if $\neg(A \wedge B) \in \Sigma$ then either $\neg A \in \Sigma$ or $\neg B \in \Sigma$;*
(5) *if $A \vee B \in \Sigma$ then either $A \in \Sigma$ or $B \in \Sigma$;*
(6) *if $\neg(A \vee B) \in \Sigma$ then $\neg A \in \Sigma$ and $\neg B \in \Sigma$;*
(7) *if $A \supset B \in \Sigma$ then either $\neg A \in \Sigma$ or $B \in \Sigma$;*
(8) *if $\neg(A \supset B) \in \Sigma$ then $A \in \Sigma$ and $\neg B \in \Sigma$.*
(9) *if $\forall\alpha A(\alpha) \in \Sigma$ then $A(\alpha/t) \in \Sigma$, for all $t \in T(\Sigma)$.*
(10) *if $\neg\forall\alpha A(\alpha) \in \Sigma$ then $\neg A(\alpha/t) \in \Sigma$, for some $t \in T(\Sigma)$.*
(11) *if $\exists\alpha A(\alpha) \in \Sigma$ then $A(\alpha/t) \in \Sigma$, for some $t \in T(\Sigma)$.*
(12) *if $\neg\forall\alpha A(\alpha) \in \Sigma$ then $\neg A(\alpha/t) \in \Sigma$, for all $t \in T(\Sigma)$.*

Proof.
(1) – immediate, by the construction of Σ.

(2) – if $\neg\neg A \in \Sigma$ then, since the algo-derivation is exhausted, A should occur in *list_proof*, as the '$\neg$ *el*' ND-rule must have been applied to $\neg\neg A$.

(3) – $A \wedge B \in \Sigma$, then again, since the algo-derivation is exhausted, A and B should occur in the *list_proof*, as the '$\wedge$ *el*' ND-rule must have been applied to $A \wedge B$.

(4) – if $\neg(A \wedge B) \in \Sigma$ then, since the algo-derivation is exhausted, by rule (2.2.1) of Procedure 2.2 of the proof-searching algorithm, $\neg(A \wedge B)$, should have served as a source of a new goal, $A \wedge B$. Further, $A \wedge B$, should have generated, by rule (2.1.3) of Procedure 2.1, new goals A and B. If $A \wedge B$ has been reached then it must have been involved in an application of the '$\neg$ *in*' rule, together with $\neg(A \wedge B)$, in which case the latter would have been discarded. Therefore, $A \wedge B$ is not reached, and, so either A or B is not reached. Again, since the algo-derivation is exhausted, either $\neg A$ or $\neg B$ should occur within *list_proof*.

(5) – if $A \vee B \in \Sigma$ then, following rule (2.2.2) of Procedure 2.2, $\neg A$ should appear as a goal. If it has been reached then by '$\vee$ *el*' rule, B must be in the proof. Alternatively, if $\neg A$ has not been reached, since the algo-derivation is exhausted, by rule (2.1.2) of Procedure 2.1, A must have occurred in *list_proof* as an assumption. By the construction of the algorithm, A in this case is not discarded.

(6) – if $\neg(A \vee B) \in \Sigma$ then, following by rule (2.2.1) of Procedure 2.2 of the algorithm, $A \vee B$ should appear as a goal. If it has been reached then, since the algo-derivation is exhausted, $\neg(A \vee B)$ and $A \vee B$ should have been used as premises of the '$\neg$ *in*' rule, in which case $\neg(A \vee B)$ would have been discarded. Thus, the goal $A \vee B$ is not reached. By a special technique linked to the rules (2.1.4.1) and (2.1.4.2) of Procedure (2.1), $\neg A \wedge \neg B$ should have appeared as a new goal which is reached 'automatically'. (This special technique is fired when rules 2.1.4.1 and 2.1.4.2 fail, in which case the subroutine called by these rules is deleted, a new goal, the negation of the goal $A \vee B$ is taken as a new assumption, and a new goal, **false**, is set up.) It is obvious then that from $\neg(A \vee B)$ the algorithm would have derived both $\neg A$ and $\neg B$. Therefore, both $\neg A$ and $\neg B$ are in Σ.

(Cases (7) and (8) can be proved analogously.)

(9) if $\forall\alpha A(\alpha) \in \Sigma$ then, according to the proof searching technique (see rule (2.1.6) of Procedure 2.1) we would generate formulae eliminating the $\forall$ quantifier using every variable from $T(\Sigma)$.
(Cases (10) – (12) can be proved analogously.)
(End)

Now, based on Lemma 1, we can prove the completeness theorem.

Theorem 4 (Completeness). *For any (possibly empty) set of FOL formulae, Γ, and any FOL formula, G, if $\Gamma \models_{ND} G$ then the proof-searching algorithm* $\mathrm{ND}_{\mathrm{alg}}$ *terminates having found the desired derivation $\Gamma \vdash G$.*

Proof. We will sketch the proof of Theorem 4 by contraposition, which shows that for any Γ and G, if no algo-derivation $\Gamma \vdash G$ has been found, then there exists an evaluation $\mathcal{I}$ such that formulae in Γ are true under $\mathcal{I}$ while G is false. Assume we have a task $\Gamma \vdash G$, for which no derivation has been found. In other words, this means that the following derivation cannot be found: $\Gamma, G' \vdash$ **false**, where G' is $\neg G$ if the main connective in G is not negation, otherwise, if G is of the form $\neg H$, then $G' = H$. Let Γ' be a set of non-discarded formulae in the exhausted derivation for Γ, G'. Following Lemma 1, the set Γ' forms a Hintikka set, hence, by the Hintikka lemma [12], Γ' is a satisfiable set of formulae. Therefore, we can find a valuation $\mathcal{I}$ such that all formulae in Γ' are true under $\mathcal{I}$. Therefore, (since $\Gamma, G' \subseteq \Gamma'$) under this interpretation, all formulae in Γ are true while G itself is false. *(End)*

5 Application of the Proof Searching Algorithm

As the first example, let us apply the proof searching algorithm to find a proof of the Peirce law, $((p \supset q) \supset p) \supset p$. We apply the algorithm writing this formula as the initial goal, G_0, (the set, Γ, of given assumptions is empty). Next, following step (5a) of the algorithm, we update *list_goals* by the consequent of the goal, $G_1 = p$, and update *list_proof* by its antecedent, $(p \supset q) \supset p$, setting p as the *current_goal*. So we get

list_proof	*list_goals*
	$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$ assumption	$G_0,\ p$

We are at step (2) of the algorithm. Since p is not reached and no elimination rules are applicable, following (5a) of the $\mathrm{ND}_{\mathrm{alg}}$ algorithm, we add a new goal, $G_2 = \mathbf{false}$, and a new assumption, $\neg p$. Thus, we obtain

list_proof	*list_goals*
	$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$ assumption	$G_0,\ p$
1. $\neg p$ assumption	$G_0,\ p,\ \mathbf{false}$

Again, we are at step (2) of the algorithm. The current goal, **false** is not reached and no elimination rules are applicable. Hence, following step (5b) of the $\mathrm{ND}_{\mathrm{alg}}$ algorithm, we analyse complex formulae of the proof. The first complex formula to be analysed is $(p \supset q) \supset p$. Applying (2.2.3) of Procedure 2.2 of the algorithm, we update *list_goals* by a new goal, $G_3 = p \supset q$. Again, since we can not reach this goal, following step (5a), we put p into *list_proof* and q into the *list_goals*, setting $G_4 = q$ as the current goal:

list_proof	*list_goals*
	$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$ assumption	$G_0,\ p$
1. $\neg p$ assumption	$G_0,\ p,\ \mathbf{false}$
	$G_0,\ p,\ \mathbf{false},\ p \supset q$
2. p assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$

Note that although we have a contradiction in the proof, we do not apply the corresponding introduction rule, $\neg$ *in* as the current goal is not **false**. Since q cannot be reached at this stage, we apply (5a), updating *list_proof* by $\neg q$ and *list_goals* by $G_5 = \mathbf{false}$.

list_proof		*list_goals*
		$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$	assumption	$G_0,\ p$
1. $\neg p$	assumption	$G_0,\ p,\ \mathbf{false}$
		$G_0,\ p,\ \mathbf{false},\ p \supset q$
2. p	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$
3. $\neg q$	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q,\ \mathbf{false}$

Since **false** is now the current goal, and the proof contains complementary literals p and $\neg p$, following (3b) of the algorithm, we apply the '$\neg$ *in*' introduction rule to these formulae (respectively formulae 1 and 2 in the proof). This gives us $\neg\neg q$ as the conclusion (formula 4 in the proof). At this stage we mark formula 3 as discarded and delete **false** from *list_goals*. The current goal now is q. Eliminating double negation from $\neg\neg q$, we get q on step 5.

list_proof		*list_goals*
		$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$	assumption	$G_0,\ p$
1. $\neg p$	assumption	$G_0,\ p,\ \mathbf{false}$
		$G_0,\ p,\ \mathbf{false},\ p \supset q$
2. p	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$
3. $\neg q$	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q,\ \mathbf{false}$
4. $\neg\neg q$	'$\neg$ *in*', 1,2	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$
5. q	'$\neg$ *el*', 4	$G_0,\ p,\ \mathbf{false},\ p \supset q$

Thus, we have reached the current goal, q, and, according to the algorithm, we analyze the previous goal, $p \supset q$ in this case. Applying (3b) of the searching algorithm, we invoke '$\supset$ *in*' rule which results in $p \supset q$. We write the latter into the *list_proof* and delete p and $p \supset q$ from *list_goals*. The current goal now is **false**. Recall that the use of '$\supset$ *in*' obliges us to discard all formulae 2–5 from *list_proof*. Next, applying $\supset$ *el* rule to formulae 0 and 6, we obtain p at step 7 of the proof.

list_proof		*list_goals*
		$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$	assumption	$G_0,\ p$
1. $\neg p$	assumption	$G_0,\ p,\ \mathbf{false}$
		$G_0,\ p,\ \mathbf{false},\ p \supset q$
2. p	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$
3. $\neg q$	assumption	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q,\ \mathbf{false}$
4. $\neg\neg q$	'$\neg$ *in*', 1,2	$G_0,\ p,\ \mathbf{false},\ p \supset q,\ q$
5. q	'$\neg$ *el*', 4	$G_0,\ p,\ \mathbf{false},\ p \supset q$
6. $p \supset q$	'$\supset$ *in*', 5	$G_0,\ p,\ \mathbf{false}$
7. p	'$\supset$ *el*', 0,6	$G_0,\ p,\ \mathbf{false}$

This gives us a contradiction, with step 1, and, thus, the current goal, **false** has been reached. Therefore, we apply the '$\neg$ *in*' introduction rule, obtaining $\neg\neg p$

(the negation of the most recent non-discarded assumption), writing it as step 8 of the proof, deleting **false** from the *list_goals* and setting p as the current goal.

list_proof		*list_goals*
		$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$	assumption	$G_0,\ p$
1. $\neg p$	assumption	$G_0,\ p,$ **false**
		$G_0,\ p,$ **false**, $p \supset q$
2. p	assumption	$G_0,\ p,$ **false**, $p \supset q,\ q$
3. $\neg q$	assumption	$G_0,\ p,$ **false**, $p \supset q,\ q,$ **false**
4. $\neg\neg q$	'$\neg$ *in*', 1,2	$G_0,\ p,$ **false**, $p \supset q,\ q$
5. q	'$\neg$ *el*', 4	$G_0,\ p,$ **false**, $p \supset q$
6. $p \supset q$	'$\supset$ *in*', 5	$G_0,\ p,$ **false**
7. p	'$\supset$ *el*', 0,6	$G_0,\ p,$ **false**
8. $\neg\neg p$	'$\neg$ *in*', 1,7	$G_0,\ p$

Eliminating double negation from $\neg\neg p$, we get p on step 9, hence, we have reached the current goal, p. Thus, according to the algorithm, we analyze the previous goal, which in this case is the initial goal. This allows us to apply the '$\supset$ *in*' rule and to derive the desired $((p \supset q) \supset p) \supset p$:

list_proof		*list_goals*
		$G_0 = ((p \supset q) \supset p) \supset p$
0. $(p \supset q) \supset p$	assumption	$G_0,\ p$
1. $\neg p$	assumption	$G_0,\ p,$ **false**
		$G_0,\ p,$ **false**, $p \supset q$
2. p	assumption	$G_0,\ p,$ **false**, $p \supset q,\ q$
3. $\neg q$	assumption	$G_0,\ p,$ **false**, $p \supset q,\ q,$ **false**
4. $\neg\neg q$	'$\neg$ *in*', 1,2	$G_0,\ p,$ **false**, $p \supset q,\ q$
5. q	'$\neg$ *el*', 4	$G_0,\ p,$ **false**, $p \supset q$
6. $p \supset q$	'$\supset$ *in*', 5	$G_0,\ p,$ **false**
7. p	'$\supset$ *el*', 0,6	$G_0,\ p,$ **false**
8. $\neg\neg p$	'$\neg$ *in*', 1,7	$G_0,\ p$
9. p	'$\neg$ *el*', 8	G_0
10. $((p \supset q) \supset p) \supset p$	'$\supset$ *in*', 9	G_0 is reached

Now we will consider how the proof searching technique can build an ND proof for formula (1) $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$. Note that as this proof will be based upon proof searching technique it will differ, due to the searching technique, from the previously presented proof of this formula in §2.

Analysing the structure of the initial goal, $G_0 = \forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$, we set its antecedent as the assumption and its consequent as a new goal.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$

Now we apply all possible elimination rules. Thus, applying $\forall\ el$ we infer the formula $P(x_1) \wedge Q(x_1)$, where x_1 is a new unflagged variable in the deduction.

Additionally, to prevent infinite applications of this rule we check the formula with a special mark.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(x_1) \wedge Q(x_1)$	0, $\forall\ el$	$G_0,\ G_1$
2. $P(x_1)$	1, $\wedge\ el_1$	$G_0,\ G_1$
3. $Q(x_1)$	1, $\wedge\ el_2$	$G_0,\ G_1$

No elimination rules are applicable now. Thus, we consider the last formula in the *list_goals*, i.e. $\forall x P(x) \wedge \forall x Q(x)$. The algorithm reduces this goal to the new goals, $\forall x P(x)$ and $\forall x Q(x)$ and sets up the former as the current goal. Since $\forall x P(x)$ is not reachable, the new goal is $P(y_1)$, where y_1 is a new flagged variable. (We will use variable x_i as unflagged and y_j as flagged.)

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(x_1) \wedge Q(x_1)$	0, $\forall\ el$	$G_0,\ G_1$
2. $P(x_1)$	1, $\wedge\ el_1$	$G_0,\ G_1$
3. $Q(x_1)$	1, $\wedge\ el_2$	$G_0,\ G_1, \forall x Q(x), \forall x P(x), P(y_1)$

Now we are searching the *list_proof* for a formula that unifies with $P(y_1)$. Thus, we find $P(x_1)$ and a substitution $\sigma_1 = x_1/y_1$. The goal $P(y_1)$ is, therefore, reached. We apply σ_1 to all formulae with free variable x_1 in *list_proof*.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall\ el$, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge\ el_1$, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge\ el_2$, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$

Next we use the following proof search rule: if a goal is $A(\beta)$, where β is a new flagged variable, and $\forall \alpha A(\alpha)$ is the last goal in *list_goals*, then by applying $\forall\ in$ to a formula $A(\beta)$ in the *list_proof* we automatically reach this goal. Thus, we apply $\forall\ in$ to formula 2. (Recall that variable y_1 is absolutely flagged.) Since the current goal, $\forall x Q(x)$, is not reachable the algorithm generates a new goal, $Q(y_2)$, where y_2 is a new flagged variable. The derivation now looks as follows:

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall$ *el*, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge$ el_1, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge$ el_2, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$
4. $\forall x P(x)$	2, $\forall$ *in*, y_1 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$

Next, since the algorithm cannot find a unification with $Q(y_2)$ (both y_1 and y_2 are flagged), it puts $\neg Q(y_2)$ as a new assumption and **false** as a new goal. Recall that formula 0 is marked. Now we have the case when we are allowed to get rid of this mark and to re-apply $\forall$ *el* rule to 0 (and mark it again) obtaining 6 and then eliminate conjunction from the latter.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall$ *el*, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge$ el_1, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge$ el_2, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$
4. $\forall x P(x)$	2, $\forall$ *in*, y_1 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
5. $\neg Q(y_2)$	assumption	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
6. $P(x_2) \wedge Q(x_2)$	1, $\forall$ *el*	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
7. $P(x_2)$	6, $\wedge$ el_1	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
8. $Q(x_2)$	6, $\wedge$ el_2	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$

Now $\neg Q(y_2)$ and $Q(x_2)$ are unifiable by a substitution $\sigma_2 = x_2/y_2$. We apply σ_2, obtaining contradictory formulae, $Q(y_2)$ and $\neg Q(y_2)$, and next apply $\neg$ *in* and then $\neg$ *el*.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall$ *el*, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge$ el_1, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge$ el_2, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$
4. $\forall x P(x)$	2, $\forall$ *in*, y_1 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
5. $\neg Q(y_2)$	assumption	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
6. $P(y_2) \wedge Q(y_2)$	1, $\forall$ *el*, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
7. $P(y_2)$	6, $\wedge$ el_1, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
8. $Q(y_2)$	6, $\wedge$ el_2, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2), \mathbf{false}$
9. $\neg\neg Q(y_2)$	5,8, $\neg$ *in*	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
10. $Q(y_2)$	9, $\neg$ *el*	$G_0,\ G_1, \forall x Q(x), Q(y_2)$

Note that once the rule $\neg$ *in* has been applied, assumption 5 and formulae 6-8 became discarded. Now we have reached $Q(y_2)$. The previous goal is $\forall x Q(x)$, which is reachable by $\forall$ *in*, hence, variable y_2 becomes absolutely flagged. Now

the current goal is $\forall x P(x) \wedge \forall x Q(x)$ which is reachable from 4 and 11 by $\wedge$ *in*. Now we are left with the initial goal $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall$ *el*, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge$ el_1, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge$ el_2, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$
4. $\forall x P(x)$	2, $\forall$ *in*, y_1 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
5. $\neg Q(y_2)$	assumption	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
6. $P(y_2) \wedge Q(y_2)$	1, $\forall$ *el*, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
7. $P(y_2)$	6, $\wedge$ el_1, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
8. $Q(y_2)$	6, $\wedge$ el_2, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
9. $\neg\neg Q(y_2)$	5,8, $\neg$ *in*	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
10. $Q(y_2)$	9, $\neg$ *el*	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
11. $\forall x Q(x)$	10, $\forall$ *in*, y_2 flagged	$G_0,\ G_1$
12. $\forall x P(x) \wedge \forall x Q(x)$	4,11, $\wedge$ *in*	G_0

The initial goal $\forall x(P(x) \wedge Q(x)) \supset (\forall x P(x) \wedge \forall x Q(x))$ is reachable from 11 by $\supset$ *in*. Applying this rule, we discard assumption 0 and all formulae 1-12 and, thus, reach the initial goal.

list_proof		*list_goals*
		G_0
0. $\forall x(P(x) \wedge Q(x))$	assumption	$G_0,\ G_1 = \forall x P(x) \wedge \forall x Q(x)$
1. $P(y_1) \wedge Q(y_1)$	0, $\forall$ *el*, σ_1	$G_0,\ G_1$
2. $P(y_1)$	1, $\wedge$ el_1, σ_1	$G_0,\ G_1$
3. $Q(y_1)$	1, $\wedge$ el_2, σ_1	$G_0,\ G_1, \forall x Q(x), \forall x P(x)$
4. $\forall x P(x)$	2, $\forall$ *in*, y_1 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
5. $\neg Q(y_2)$	assumption	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
6. $P(y_2) \wedge Q(y_2)$	1, $\forall$ *el*, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
7. $P(y_2)$	6, $\wedge$ el_1, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
8. $Q(y_2)$	6, $\wedge$ el_2, σ_2	$G_0,\ G_1, \forall x Q(x), Q(y_2)$, **false**
9. $\neg\neg Q(y_2)$	5,8, $\neg$ *in*, y_2 flagged	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
10. $Q(y_2)$	9, $\neg$ *el*	$G_0,\ G_1, \forall x Q(x), Q(y_2)$
11. $\forall x Q(x)$	10, $\forall$ *in*	$G_0,\ G_1$
12. $\forall x P(x) \wedge \forall x Q(x)$	4,11, $\wedge$ *in*	G_0
13. $\forall x(P(x) \wedge Q(x)) \supset$ $(\forall x P(x) \wedge \forall x Q(x))$	12, $\supset$ *in*	G_0 is reached

6 Discussion

We have presented a proof searching algorithm for classical first-order natural deduction system and established its correctness. Note that most of the automated reasoning tools based on natural deduction are *interactive*. On the contrary, our approach enables potentially fully automatic implementation. We have

implemented the propositional part of the presented searching procedure and are currently extending it to FOL. The technique has been tested on a large number of problems for classical logic taken mostly from [7]. Also, to the best of our knowledge, the only other ND-based theorem prover with a goal-directed procedure similar to that of the $\mathrm{ND}_{\mathrm{alg}}$ presented in this paper, has been developed in CMU [19]. However, the underlying ND system that we use, differs in the set of introduction rules, hence the searching algorithm uses essentially different techniques.

Moreover, we saw our task, first of all, as developing the 'basic' proof searching algorithm. In other words, our aim was to establish the minimal set of procedures which form a complete $\mathrm{ND}_{\mathrm{alg}}$. This is why the underlying ND system was also taken with the minimal set of elimination and introduction rules. As can be expected, in this case, the payoff is a significant extension of the searching space, and the complexity of the searching procedure. Thus, the important part of our future work will be further refinement of the technique developed. We believe that, since our machinery is based upon the minimal set of procedures, it is open to any direction of possible refinement. One of these directions can be seen, for example, in supplying the machinery with a set of derivations that may significantly reduce the search space, and, hence, improve the complexity of its performance.

Complexity, in turn, is another component of future research. Although we have not studied this issue in detail, we believe that the following observation is important. Due to its synthetic character, our searching technique performs almost identically for classes of formulae with similar logical structures. For example, if we substitute p and q by *arbitrarily complex formulae* in Peirce law $((p \supset q) \supset p) \supset p$ the $\mathrm{ND}_{\mathrm{alg}}$ proof for the latter will be schematically identical to the $\mathrm{ND}_{\mathrm{alg}}$ proof presented in our paper. In other words, following our procedure, in general, we are not forced to analyze the structure of complex subformulae *unless needed*.

Things are different when a purely analytic proof searching procedures applies, like a tableau or a resolution based method. Here, unless the machinery is supplied with additional mechanisms, the task is to reduce all complex subformulae to the level of literals. As a consequence, using our method we can have very simple proofs for formulae with a very large number of variables. Similarly, again, as the structure of formulae is the basis for the search strategy, there are formulae which are difficult for analytic methods, but their proof is simple for the $\mathrm{ND}_{\mathrm{alg}}$. For example, the famous formula, $(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q)$, which served as an example showing that analytic tableaux cannot polynomially simulate truth tables [8], is readily proved by the $\mathrm{ND}_{\mathrm{alg}}$. On the other hand, again, due to the synthetic character of ND searching procedures, we can distinguish classes of formulae which are difficult for this approach, while corresponding proofs in purely analytic methods are simple. These are, for example, formulae, whose subformulae are 'irrelevant' to the $\mathrm{ND}_{\mathrm{alg}}$ derivation. Such formulae might lead us to the redundant branches in the searching tree, but no technique is known to distinguish them. This results in an increased complexity

of the algorithm. Moreover, we believe that only some partial solutions to this problem can be found, while the general problem of avoiding such redundancy cannot be solved.

References

1. L. Bachmair and H. Ganzinger. A theory of resolution. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 2. Elsevier, 2001.
2. V. Bocharov and V. Markin. *Principles of Logic.* Infra, Moscow, 1997. (In Russian).
3. A. Bolotov, A. Bocharov, and A. Gorchakov. A proof search algorithm for the natural deduction classical propositional calculus. *Logical Investigations*, 3:181–186, 1995. (in Russian).
4. A. Bolotov, A. Bocharov, and A. Gorchakov. Proof searching algorithm in classical first-order calculus. *Logical Investigations*, 5, 1998. (In Russian).
5. A. Bolotov, V. Bocharov, A. Gorchakov, V. Makarov, and V. Shangin. *Let Computer Prove It.* Logic and Computer. Nauka, Moscow, 2004. (In Russian).
6. C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.
7. A. Church. *Introduction to Mathematical Logic.* Princeton, N.J., 1956.
8. M. D'Agostino and M. J. Mondadori. The taming of the cut. Classical refutations with analytic cut. *Journal of Logic and Computation.*, 4:285–319, 1994.
9. F. Fitch. *Symbolic Logic.* NY: Roland Press, 1952.
10. M. Fitting. *First-Order Logic and Automated Theorem-Proving.* Springer-Verlag, Berlin, 1996.
11. G. Gentzen. Investigation into logical deduction. In *The Collected Papers of Gerhard Gentzen*, pages 68–131. Amsterdam: North-Holland, 1969.
12. J. Hintikka. Notes on the quantification theory. *Societas Scientarium Fennica Commentationes Physico-Mathematicae*, XVII(11), 1957.
13. S. Jaskowski. On the rules of suppositions in formal logic. In *Polish Logic 1920-1939*, pages 232–258. Oxford University Press, 1967.
14. V. Makarov. Automatic theorem-proving in intuitionistic propositional logic. In *Modern Logic: Theory, History and Applications. Proceedings of the 5th Russian Conference*, StPetersburg, 1998. (In Russian).
15. F. Pfenning. Logical frameworks. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter XXI, pages 1063–1147. Elsevier, 2001.
16. J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA'99), Springer-Verlag LNCS, 1581*, L'Aquila, Italy, April 1999.
17. W. Quine. On natural deduction. *Journal of Symbolic Logic*, 15:93–102, 1950.
18. V. Shangin. *Automatic Search for Natural Deduction in Classical First Order Logic.* PhD thesis, Department of Logic, Faculty of Philosophy, Moscow State University, 2004. (In Russian).
19. W. Sieg and J. Byrnes. Normal natural deduction proofs (in classical logic). *Studia Logica*, 60:67–100, 1998.
20. M. Wooldridge. *Reasoning about Rational Agents.* MIT Press, 2000.