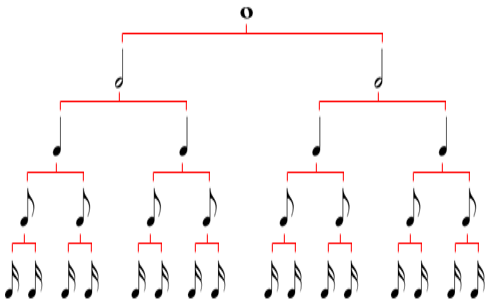
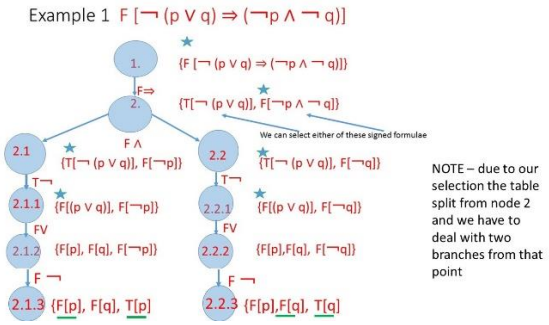
**Student Co-Creators Report**

# Project Title:

# HANDEL: Heuristics for Natural Deduction Layout






# Research Team:

**STUDENTS**
Olimpia-Maria Moisiu, MEng Software Engineering, Level 5
Natalia Yerashenia, PhD researcher, Software Systems Engineering Group

**STAFF**
Dr Alexander Bolotov
Dr Klaus Draeger

**College of Design, Creative and Digital Industries,**
**School of CS & Engineering**

## Academic Year 2018-2019

## Section 1. Executive Summary (300-400 words)

The project team undertook research in the area of formal specification and verification. We studied the deductive approach, based on the Natural Deduction style proof. Natural deduction systems (NDS) were originally aimed to simulate the way the mathematicians think, mainly in proving various statements. The research community in computer science has become more interested in NDS approach as the one reflecting the natural way of the reasoning process. Moreover, recently NDS found their applications in various areas of computer science such as reasoning for rational agents or formal verification of security protocols. NDS has been the centre of the research within Dr Bolotov's Software Systems Engineering group where Dr Draeger is a member.

The main aim of the project has been to investigate how changing the basic set of Booleans affects the ND proof. We developed an efficient approach to tackle this problem by forming three sets of test formulae – with the full set of Booleans (Conjunction, disjunction, negation, implication), and two restricted sets – the one with negation and implication and the other one with negation and disjunction. We then ran the proof search programme on these sets as well as manual theorem proving.

The results of our journey were very mixed. Our main conjecture before undertaking this study was that the basic set of Booleans in the formulae does affect the complexity of the proof significantly, and, in particular, that a system using only implication and negation would have shorter proofs. The proofs we have obtained and shown below illustrate that this is not, in fact, true! For simple examples, the conjecture was approved; however, for more complex examples the proofs of the equivalent formulae formulated with {=>, ¬} and {V, ¬} are similar.

We still do not know WHERE exactly the complexity grows and WHAT are the remedies. The main lessons learned from the research were: 1) In many cases, the proof of a formula using the full set of Boolean connectives could be closely mimicked using the chosen restricted set, meaning that the complexity did not usually increase substantially. 2) The behaviour of a translated formula could depend strongly on choices made during the translation. For example, one of the operations which tends to be problematic in the natural deduction approach, **disjunction,** is associative; so, we can simply write (aVbVc) for either ((aVb) Vc) or (aV(bVc)). If we represent disjunction by translating (xVy) as (~x>y) throughout, those two (equivalent) formulae become (~(~a => b) => c) and (~a => (~b => c)), which behave very differently. On a similar note, commutativity means that (aVb) and (bVa) are equivalent; but if a and b are themselves complex formulae, the complexity of proving the corresponding translated formulas (~a>b) and (~b>a) can differ. 3) As a consequence of the previous point, one important direction of future research is the investigation of suitable strategies for choosing an appropriate translation among the possible equivalent ones.

Our results will be useful for the research within the Software Engineering research group, specifically, for the ongoing research project on the automation of the natural deduction theorem proving. We also envisage these results will be useful to inform our CS&Engineering students in assisting their choice of final year projects.

## Section 2. Background and Aims (200-300 words)

In this section, after a short introduction, we present the general problem setting, give an overview of the natural deduction systems, the basic proof search procedure we have investigated, and the basic proof search algorithm. This will help to understand the context of our research, our methodology and our interpretation of results.

1. **Intro.** Computers become more and more involved in our daily life. We trust them to take care of our homes and we enjoy the comfort they bring. An aircraft autopilot takes us across the ocean and the vehicle computer automatically parks your car. Let's just imagine what will happen if suddenly services that are crucial for our society go wrong – the consequences could be even catastrophic. Consider what a tiny mistake in a program controlling a flight or an artificial heart can cause. Imagine an absolute waste of billions of pounds if an auto-navigating space shuttle would miss its target flying into the infinite depth of the Universe as an extremely expensive but useless toy.
   To avoid these unwanted consequences, we need to guarantee that the hardware and relevant computer programs are doing exactly what we want them to do. In this project, we are interested in using mathematical modelling. The idea is rather transparent. If we have a mathematical model of a computer system which we are analyzing, then we run experiments with this model to identify where things can go wrong. However, generally, the programs we mentioned above are so complex that neither an individual nor even a group of experts are able to test them. Is there a solution to this problem? Yes, there is an elegant solution – to use computer power. It means that we need special programs designed for checking other programs, and we need to organise some reasoning to guide this process.
   Our project, HANDEL, has been undertaken to tackle some aspects of this reasoning mechanism. We have studied a particular reasoning technique called Natural Deduction.

2. **General Problem Setting.** The significance of formal specification with subsequent verification in Software Engineering is well accepted [2,4]. 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. We represent the task of deductive verification, DV, of a system Sys with its specification Spec by the following signature [2]:
   DV :: Sys × Spec → B × [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 × [Proof].

3. **Context.** In the scheme above, we treat DV resulting in the Natural Deduction style proof. Natural deduction systems (NDS) were originally aimed to simulate the way the mathematicians think, mainly in proving various statements [3]. The indicative feature of natural deduction is an explicit representation of the proof. The research community in computer science for a long time underestimated NDS considering NDS not suitable for automation. This led to a rather paradoxical situation - NDS were taught as part of computer science or mathematics curriculum, in many of the world universities, but only as a convenient framework exemplifying the reasoning process. Moreover, recently NDS found their applications in various areas of computer science such as reasoning for rational agents or formal verification of security protocols. NDS has been the centre of the research within Dr Bolotov's Software Systems Engineering group where Dr Draeger is a member. Much of this work has also been done in collaboration with Moscow State University where a fundamental problem has been recently found – developed NDS algorithms cannot efficiently cope with so-called Pigeonhole Principle, discovered by Dirichlet in 1834, and thus often called Dirichlet Principle.
   This principle is very simple and intuitive - if n + 1 pigeons sit in n holes then some hole contains more than one pigeon.

4. **Importance of the Principle.** Surprisingly, this simple and intuitive principle has vast applications – in mathematics, mostly in combinatorics, in computing, for example, in analysing algorithms or hashing. Various formulations of this principle depend on the context of the problem setting. It is notable that even in its simplest formulation, in classical propositional logic, the Pigeonhole Principle causes problems for proof methods. As an example, in the implementation of the proof search developed by Dr Bolotov's group, the proof for the case with 6 pigeons in the existing NDS, after 41 hours of work, generated 3 001 000 steps with 1 226 000 interim goals. Surely, this is something unrealistic for a human being. It is not surprising that the automated reasoning research community has been using the Pigeonhole Principle as a valuable test for proof search algorithms and their complexity.

**5. Initial Research Aims:**

- To search for an algorithmic solution to the ND proof capable to proving the Pigeonhole Principle in classical propositional logic.
- To investigate the computational complexity of the proposed solution.

6. **Initial Assumptions:** at the stage of project design our assumption was that the Pigeonhole Principle is manageable by the existing algorithms. However, a scrutinised investigation into the algorithm has led to the conclusion that things are much more complex and the initial research aims are not feasible to achieve. One of the problems on the way to achieve these aims was to understand how the change in the set of basic Boolean operations affects the ND proof. Therefore, the project team re-evaluated the project aims in this context.

**7. Revised Research Aim:**
To investigate how changing the basic set of Booleans affects the ND proof.

8. **Revised Objectives:**

1. Form a set of test formulae with the following sets of Booleans:
   a. Conjunction, disjunction, negation, implication
   b. Negation, implication
   c. Negation, disjunction
2. Run the proof search programme on (a)
3. Manually prove formulae from lists (b) and (c)

## 9. Formulation of the ND system and proof search algorithm. [1]

The figure below reflects the **CPLND** rules for the system formulated with the full set of Booleans

$$\&in\ \frac{A,\ B}{A\ \&\ B} \qquad \vee in\ \frac{A}{A \vee B} \qquad \frac{B}{A \vee B} \qquad \supset in\ \frac{B}{C => B} \qquad \neg in\ \frac{B,\ \neg B}{\neg C,}$$

$$\&el\ \frac{A\ \&\ B}{A} \qquad \frac{A\ \&\ B}{B} \qquad \vee el\ \frac{A \vee B,\ \neg A}{B} \qquad \supset el\ \frac{A => B,\ A}{B} \qquad \neg el\ \frac{\neg\neg A}{A}$$

where C is the last alive assumption.

**Figure 1. CPL$_{ND}$ for the full set of Booleans.**

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

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

***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, NDalg, is a pair (list proof, list goals) whose
construction is determined by the searching procedure described below. On each step of constructing an NDalg, a specific goal is chosen, which should be reached at the current stage. Thus, the appropriate name for such a goal would be the current goal. The first goal of list goals is extracted from the given task; we will refer to this goal as to the initial goal.

**Definition 3** (Reachability of a current goal). A current goal, Gn, 0 ≤ n, occurring in list goals= {G1,G2, . . . ,Gn} is reached if the following conditions are satisfied:
– If Gn is not false then Gn is reached if there is a formula A in list proof such that Gn = A and A is not discarded
ELSE

– Gn is false and it is reached if there are formulae A and ¬A in list proof.

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.

***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.
*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, Gn, is not reached. The type of Gn 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 Gn is not false. Here the structure of the current goal, Gn, 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= P1, . . . Pk and list goals= G1, . . . ,Gn, where Gn is the current goal. If it is impossible, at the present stage, to infer Gn then, looking at its structure, we derive a new goal Gn+1 and set the latter as the current goal.

Below we identify various cases of applying sub-procedure 2.1, where Gn = F| ¬A|A ∧ B|A ∨ B|A =>B

*Procedure 2.2.* This sub-procedure is invoked when Gn 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.

(2.2.1) Γ, ¬A ⊢ Ψ, false −→ Γ, ¬A ⊢ _ Ψ, false, A

(2.2.2) Γ,тA ∨ B ⊢ Ψ, false −→ Γ,A ∨ B ⊢ Ψ, false, ¬A

(2.2.3) Γ, A => B ⊢ Ψ, false −→ Γ,A => B ⊢ Ψ, false, A

Applying the rule (2.2.1) we have ¬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 ∨ B and our target is to derive ¬A. If we are successful then we would be able to derive B by ∨ el rule. Similarly, applying rule (2.2.3) we already have A => B in the proof and we aim at deriving A as this would enable us to apply the => el rule.

*Procedure 3.* This procedure checks the reachability of the current goal in the sequence list goals. If, according to Definition 4, the current goal Gn is reached then the sequence list goals is updated by deleting Gn and setting Gn−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.

***Proof-Searching Algorithm***

Given a (decidable) task Γ ⊢ G, where Γ is possibly empty, we already have the first goal, G0 = G, which is the initial goal and the current goal at the same time. At this stage, we apply Procedure 3, checking if G0 is reached.

Assume that G0 is not reached. Then we apply Procedure 1, obtaining all possible conclusions of the elimination ND rules checking if G0 can be reached in this way. If we fail, then Procedure 2 is invoked, and, dependent on the structure of the goals G0, then G1, 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 the 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, G0, 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.

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 preventing looping either in the application of elimination rules or in searching. The aims of marking are:

– to keep track of formulae that were used as premises of the elimination rules
invoked in procedure 1;

– to keep track of formulae in list proof that was 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∨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∨B, set up a new assumption ¬(A∨B) and a new goal, false.
– to inform the searching algorithm that no more elimination rules are applicable.

**Formulation of the algorithm.**

(1) Given a task Γ ⊢ 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, Gn, and the applicability of elimination rules, (see Procedure 3 described in §3.1).
(2a) If Gn 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 Gn (reached) is the initial goal then terminate, the desired ND proof has been found, EXIT, ELSE
(3b) (Gn 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 Gn 6= false then apply Procedure 2.1 (analysis of the structure of Gn),
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.

**Example proof based on the above searching algorithms**

| | proof | analysis | goals |
|---|---|---|---|
| 0. | | | ((p => q) => p) => p – reached |
| 1. | (p => q) => p | Assumption | p – reached from 10 |
| 2. | ¬p | Assumption | $\perp_1$ – reached 2, 8 |
| 3. | p | Assumption | p ⊃ q –reached, 7 |
| 4. | ¬q | Assumption | q – reached, 6 |
| 5. | ¬¬q | ¬in, 2, 3 | $\perp_2$ –reached 2 and 3 |
| 6. | q | ¬el 5 | |
| 7. | p ⊃ q | ⊃in 6 | |
| 8. | p | ⊃in 1, 7 | |
| 9. | ¬¬p | ¬in 2, 8 | |
| 10. | p | ¬in 9 | |
| 11. | ((p ⊃ q) ⊃ p) ⊃ p | ⊃in 10 | |

**Figure 2. Proof of Pierce Law ((p => q) => p) => p.**

Potential Impact:

1. *Stakeholders*: Solving the first aim would bring a powerful algorithmic construction while solving the second aim would give a clear understanding of the place of our construction in the hierarchy of computational complexity. We envisage that the success of the proposal will contribute not only to the core research on proofs but also on an emerging area of so-called SAT-provers. The latter are formal techniques used in formal analysis of software systems when we want to check the properties of a computational model – put intuitively, when we have a task to prove that some certain desired property of a computation is SATisfied in the formal mathematical model of the computation. Formalisms that express the Pigeonhole Principle are often used in testing SAT solvers, representing so-called class of hard problems.

2. The project "HANDEL" targets one of the recently discovered flaws in the automation of NDS proof search. Not only we are not aware of any NDS construction that can prove the Pigeonhole Principle efficiently **but have not also seen any research that attempts to study the effects of the certain basic sets of Booleans on the ND Proof search**. Hence, it is important from a theoretical point of view. Achieving the project aims is also essential for Software Systems Engineering research group, which has more than 50 publications on ND proof search.
3. *Benefits for the Project Team.* The project has been an important contribution to the student participants – being solely theoretical, it has been their first experience of working on a substantial problem, in a partnership with academics: it has given more insight to all participants on a dedicated scientific area, developed our student participants' problem-solving skills, independent research, and gave them an outstanding experience of undertaking research in a team.
4. Last but not least, the project formed, tested and approved a working research partnership that can take new challenges in the future.

**References:**

[1] Alexander Bolotov, Daniil Kozhemiachenko, Vasilyi Shangin: Paracomplete Logic K1: Natural Deduction, its Automation, Complexity and Applications. IfCoLog Journal of Logics and their Applications, Volume 5 (1): 221-262 (2018)

[2] 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.

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

[4] 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.

**Section 3. Methods** (150-300 words)

The project has been an adventure. One of the key methodologies we explored involved the representation of the underlying natural deduction reasoning as a goal-directed procedure. This has proven to be a very efficient approach to studying proofs.

To meet the objectives, we developed the following technique:
Form testing sets of formulae based on

1. Full set of Booleans: Conjunction, disjunction, negation, implication
   d. Negation, implication Only
   e. Negation, disjunction Only
4. Run the proof search programme on (a)
5. Manually prove formulae from lists (b) and (c)
6. Compare all proofs with the automated proofs build according to the algorithm

Specifically, our testing methodology involved the analysis of equivalent formulae but formulated with different sets of Booleans. The table below illustrates this methodology.

| Full sets of Booleans | ¬, V | ¬, => |
|---|---|---|
| p => p<br>p => (p V q)<br>(p & q) => q<br>¬p & (p V q) => q | ¬p V p<br>P => (p V q)<br>¬(¬p V ¬q) => q<br>¬(¬p => ¬(pVq)) => q | p => p<br>P => (¬p => q)<br>¬(p => ¬q) => q<br>¬(¬p => ¬(¬p => q)) => q |

**Figure 3. Comparison Methodology.**

The figure below represents a list of formulae on which the ND search algorithm has been tested.

p>(((p&q)>q)&(q>(p&q)))
~pV((~(p&q)Vq)&(~qV(p&q)))
p>((((q>p)&(p>q))>q)&(q>((q>p)&(p>q))))
~pV((((q>p)&(p>q))>q)&(q>((q>p)&(p>q))))
((p>q)&(q>p))V(qVp)
(p>q)>(qV(p>r))
~((p>q)&(q>p))>(qVp)
((pVq)>((p>q)>q))&(((p>q)>q)>(pVq))
(p>q)>(~q>(p>r))
(((~q>q)>q)>(p>q))>(p>q)
(~p>p)>p
(((~q>q)>q)>(p>q))>(~(p>q)>(p>q))
(~p>r)>((p>r)>r)
~(~(p>q)V(r>s))V(~(~pV(t>s))V(r>(t>s)))
((p>r)>r)>(~p>r)
~((p>q)>(r>s))V(~(p>(t>s))V(r>(t>s)))
~(~(p>q)V(r>s))V(~(p>(t>s))V(r>(t>s)))
((p>q)>(r>s))>((p>(t>s))>(r>(t>s)))
~(~(p>q)V(r>s))V(~(~pV(t>s))V(~rV(t>s)))
((p1>r)>(s>p))>(((r>q)>p)>(s>p))
((r>q)>(s>p))>((r>p)>(s>p))
(p>q)>(((p>r)>q)>q)
(((p>r)>q)>q)>((q>r)>(p>r))
(((r>p)>p)>(s>p))>(((p>q)>r)>(s>p))
q>(((~q>q)>q)>(p>q))
(((r>p)>(s>p))>(((p>q)>r)>p1))>(q1>(((p>q)>r)>p1))
q>(((~q>q)>q)>(p>q))
(~r>(p>~s))>((r>(p>q))>((s>p)>(s>q)))

~(~rV(pV~s))V(~(~rV(p>q))V((s>p)>(s>q)))

~(rV(p>~s))V((r>(p>q))>((s>p)>(s>q)))
((p1>(q1>p1))>(((~r>(p>~s))>((r>(p>q))>((s>p)>(s>q))))>p2))>(q2>p2)
((p>((~q>(r>s))>((s>q)>(p1>(r>q)))))>((~q1>(q1>r1))>s1))>s1
((r1>(((p>q)>s)>q1))>(s1>((s>p1)>(~p>~r))))>(s1>((s>p)>(r>p)))
((p>q)>((p>(p&q))&((p&q)>p)))&(((p>(p&q))&((p&q)>p))>(p>q))
(p&(~qV~r))V~((p&(~rV~p))V~((s&q)V~~(~pV~s)))
~(p&(q>~r))>~(~(p&(r>~p))>~((s&q)V(p>~s)))
(p&(~qV~r))V~(~((s&r)V(~pV~s))V(p&(~pV~q)))
(p>~(q>~r))>(((s>~r)>(p>~s))&(p>~(p>~q)))
(p>~(q>~r))>~(((s>~r)>(p>~s))>~(p>~(p>~q)))
~(p&(~qV~r))>~(((s&r)V(~pV~s))>(p&(~pV~q)))
~(p&(~qV~r))>~(~((s&r)V(~pV~s))V(p&(~pV~q)))
~(p&(~qV~r))>~((~(s&r)>(~pV~s))>(p&(~pV~q)))
(p>~(q>~r))>~((p1>p1)>~((s>~q)>(p>~s)))
(p&(~qV~r))V((~p1Vp1)&((s&q)V(~pV~s)))
(p&(~qV~r))V((~p1V~~p1)&((s&q)V(~pV~s)))
(((r>p)>(s>p))>(((p>q)>r)>t))>((p1Vp2)>(((p>q)>r)>t))
~~(p>r)>((sV(qVp))>((rVq)Vs))
~((pV(rVq))>(sVt))V(~(sVt)>(~(p&r)&~(p&q)))
((~(p&q)>r)&(p&q))>r
~(pVq)V(~(qV(rVs))>p)

**Figure 4. Extended list of testing formulae**

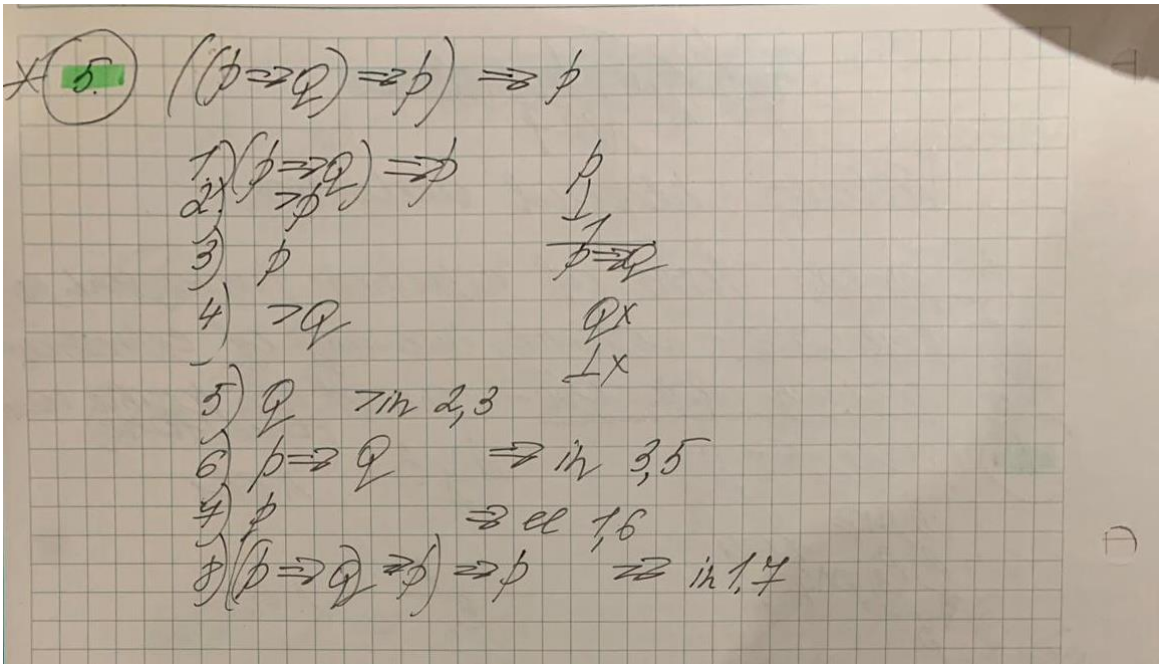
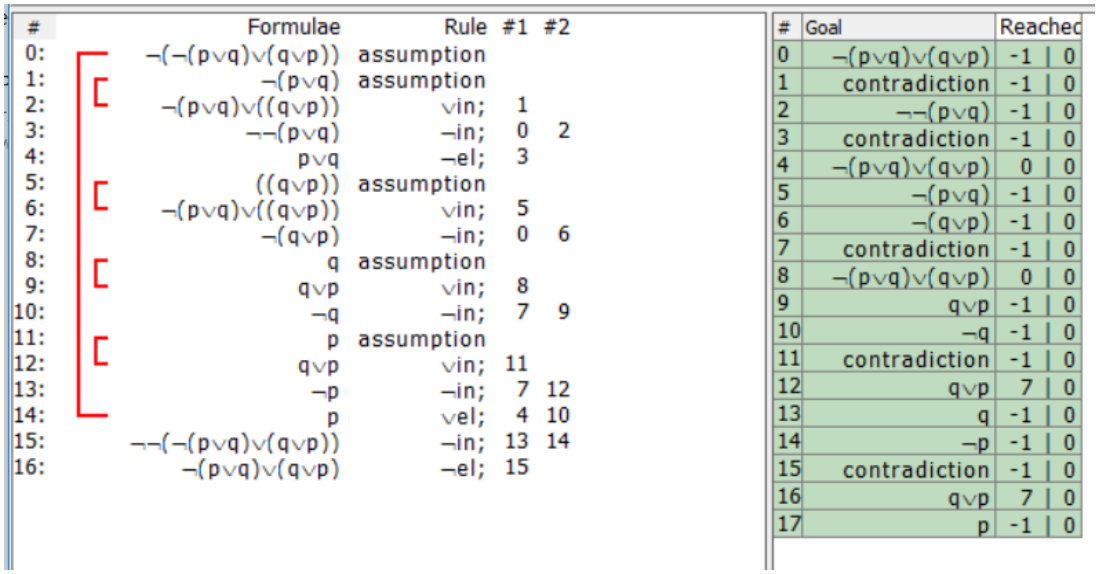


**Figure 5 Natural Deduction Proofs by hand**

Figure 5 illustrates our adventures representing some “hand made” proofs of the formulae from the test set. Our methodology of making proofs by hand is justified by the fact that an implemented proof search follows an algorithm which has been designed for the full set of Booleans and the modification of this set may need a simpler algorithm.

## Section 4. Results

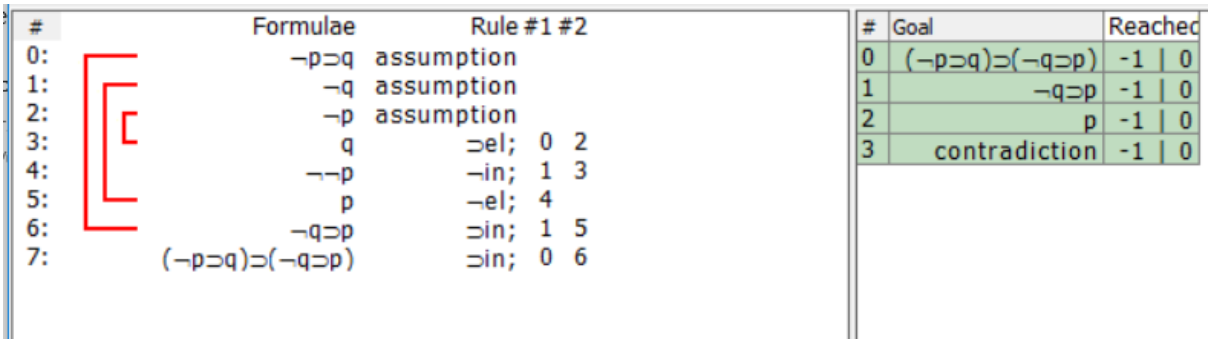
Results were very mixed. Our main conjecture before undertaking this study was that the basic set of Booleans in the formulae does affect the complexity of the proof significantly, and, in particular, that a system using only implication and negation would have shorter proofs. The proofs we have obtained and shown below illustrate that this is not, in fact, true! For simple examples, the conjecture was approved; however, for more complex examples – see the Figures below, the proofs of the equivalent formulae formulated with {=>, ¬} and {V, ¬} are similar.



| # | Formulae | Rule | #1 | #2 |
|---|---|---|---|---|
| 0: | ¬(¬(p∨q)∨(q∨p)) | assumption | | |
| 1: | ¬(p∨q) | assumption | | |
| 2: | ¬(p∨q)∨((q∨p)) | ∨in; | 1 | |
| 3: | ¬¬(p∨q) | ¬in; | 0 | 2 |
| 4: | p∨q | ¬el; | 3 | |
| 5: | ((q∨p)) | assumption | | |
| 6: | ¬(p∨q)∨((q∨p)) | ∨in; | 5 | |
| 7: | ¬(q∨p) | ¬in; | 0 | 6 |
| 8: | q | assumption | | |
| 9: | q∨p | ∨in; | 8 | |
| 10: | ¬q | ¬in; | 7 | 9 |
| 11: | p | assumption | | |
| 12: | q∨p | ∨in; | 11 | |
| 13: | ¬p | ¬in; | 7 | 12 |
| 14: | p | ∨el; | 4 | 10 |
| 15: | ¬¬(¬(p∨q)∨(q∨p)) | ¬in; | 13 | 14 |
| 16: | ¬(p∨q)∨(q∨p) | ¬el; | 15 | |

| # | Goal | Reached |
|---|---|---|
| 0 | ¬(p∨q)∨(q∨p) | -1 \| 0 |
| 1 | contradiction | -1 \| 0 |
| 2 | ¬¬(p∨q) | -1 \| 0 |
| 3 | contradiction | -1 \| 0 |
| 4 | ¬(p∨q)∨(q∨p) | 0 \| 0 |
| 5 | ¬(p∨q) | -1 \| 0 |
| 6 | ¬(q∨p) | -1 \| 0 |
| 7 | contradiction | -1 \| 0 |
| 8 | ¬(p∨q)∨(q∨p) | 0 \| 0 |
| 9 | q∨p | -1 \| 0 |
| 10 | ¬q | -1 \| 0 |
| 11 | contradiction | -1 \| 0 |
| 12 | q∨p | 7 \| 0 |
| 13 | q | -1 \| 0 |
| 14 | ¬p | -1 \| 0 |
| 15 | contradiction | -1 \| 0 |
| 16 | q∨p | 7 \| 0 |
| 17 | p | -1 \| 0 |

**Figure 6 Proof for ¬(pVq) V (qVp)**

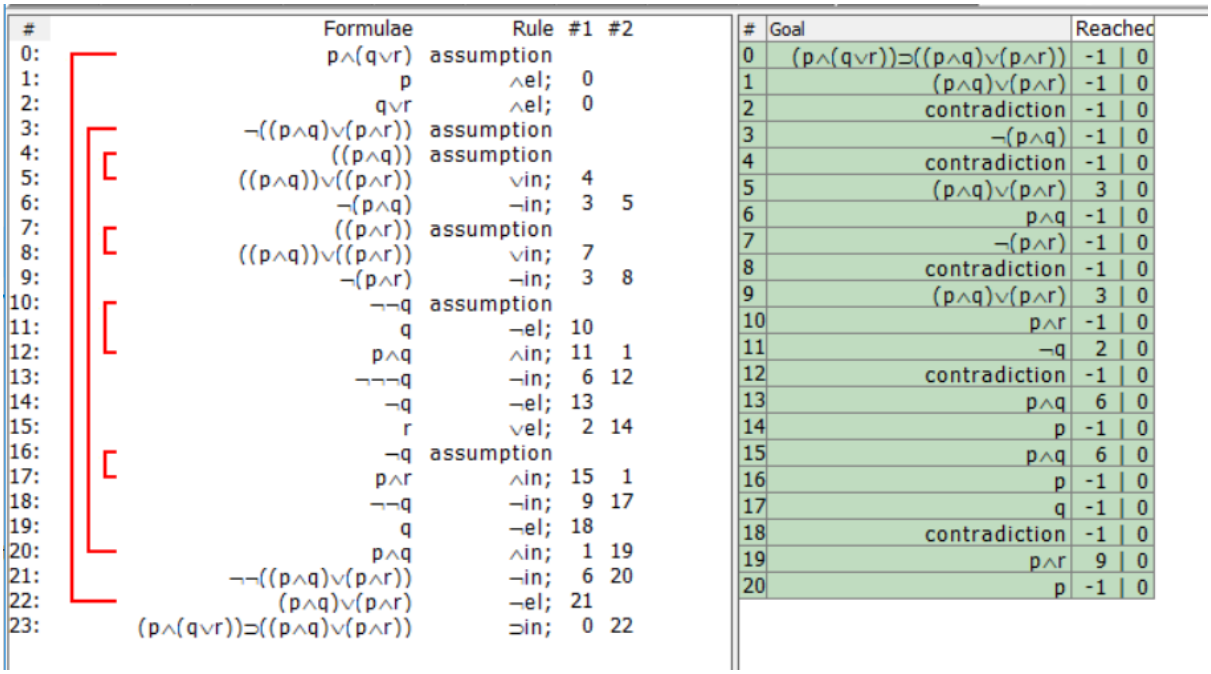


| # | Formulae | Rule | #1 | #2 |
|---|---|---|---|---|
| 0: | ¬p⊃q | assumption | | |
| 1: | ¬q | assumption | | |
| 2: | ¬p | assumption | | |
| 3: | q | ⊃el; | 0 | 2 |
| 4: | ¬¬p | ¬in; | 1 | 3 |
| 5: | p | ¬el; | 4 | |
| 6: | ¬q⊃p | ⊃in; | 1 | 5 |
| 7: | (¬p⊃q)⊃(¬q⊃p) | ⊃in; | 0 | 6 |

| # | Goal | Reached |
|---|---|---|
| 0 | (¬p⊃q)⊃(¬q⊃p) | -1 \| 0 |
| 1 | ¬q⊃p | -1 \| 0 |
| 2 | p | -1 \| 0 |
| 3 | contradiction | -1 \| 0 |

**Figure 7 Proof for (¬p => q) => (¬q => p)**

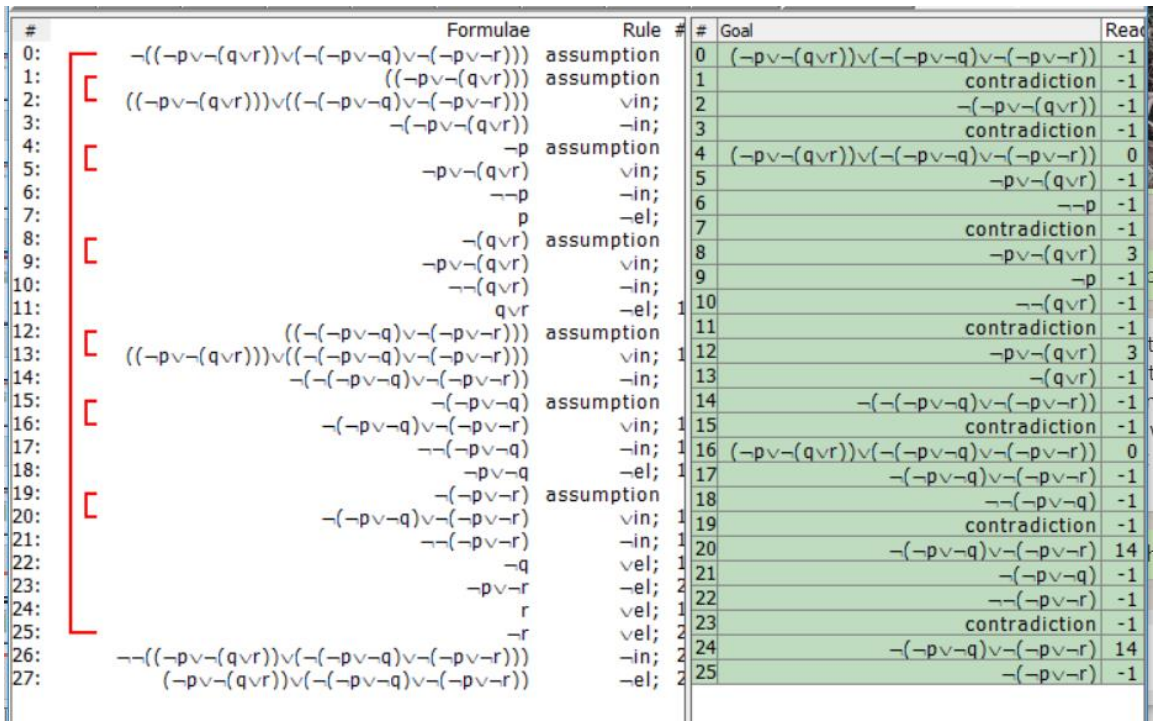
As one can see the proof in Figure 7 is significantly shorter than the one for an equivalent formula in Figure 6



| # | Formulae | Rule | #1 | #2 |
|---|---|---|---|---|
| 0: | p∧(q∨r) | assumption | | |
| 1: | p | ∧el; | 0 | |
| 2: | q∨r | ∧el; | 0 | |
| 3: | ¬((p∧q)∨(p∧r)) | assumption | | |
| 4: | (p∧q) | assumption | | |
| 5: | (p∧q)∨(p∧r) | ∨in; | 4 | |
| 6: | ¬(p∧q) | ¬in; | 3 | 5 |
| 7: | (p∧r) | assumption | | |
| 8: | (p∧q)∨(p∧r) | ∨in; | 7 | |
| 9: | ¬(p∧r) | ¬in; | 3 | 8 |
| 10: | ¬¬q | assumption | | |
| 11: | q | ¬el; | 10 | |
| 12: | p∧q | ∧in; | 11 | 1 |
| 13: | ¬¬¬q | ¬in; | 6 | 12 |
| 14: | ¬q | ¬el; | 13 | |
| 15: | r | ∨el; | 2 | 14 |
| 16: | ¬q | assumption | | |
| 17: | p∧r | ∧in; | 15 | 1 |
| 18: | ¬¬q | ¬in; | 9 | 17 |
| 19: | q | ¬el; | 18 | |
| 20: | p∧q | ∧in; | 1 | 19 |
| 21: | ¬¬((p∧q)∨(p∧r)) | ¬in; | 6 | 20 |
| 22: | (p∧q)∨(p∧r) | ¬el; | 21 | |
| 23: | (p∧(q∨r))⊃((p∧q)∨(p∧r)) | ⊃in; | 0 | 22 |

| # | Goal | Reached |
|---|---|---|
| 0 | (p∧(q∨r))⊃((p∧q)∨(p∧r)) | -1 \| 0 |
| 1 | (p∧q)∨(p∧r) | -1 \| 0 |
| 2 | contradiction | -1 \| 0 |
| 3 | ¬(p∧q) | -1 \| 0 |
| 4 | contradiction | -1 \| 0 |
| 5 | (p∧q)∨(p∧r) | 3 \| 0 |
| 6 | p∧q | -1 \| 0 |
| 7 | ¬(p∧r) | -1 \| 0 |
| 8 | contradiction | -1 \| 0 |
| 9 | (p∧q)∨(p∧r) | 3 \| 0 |
| 10 | p∧r | -1 \| 0 |
| 11 | ¬q | 2 \| 0 |
| 12 | contradiction | -1 \| 0 |
| 13 | p∧q | 6 \| 0 |
| 14 | p | -1 \| 0 |
| 15 | p∧q | 6 \| 0 |
| 16 | p | -1 \| 0 |
| 17 | q | -1 \| 0 |
| 18 | contradiction | -1 \| 0 |
| 19 | p∧r | 9 \| 0 |
| 20 | p | -1 \| 0 |

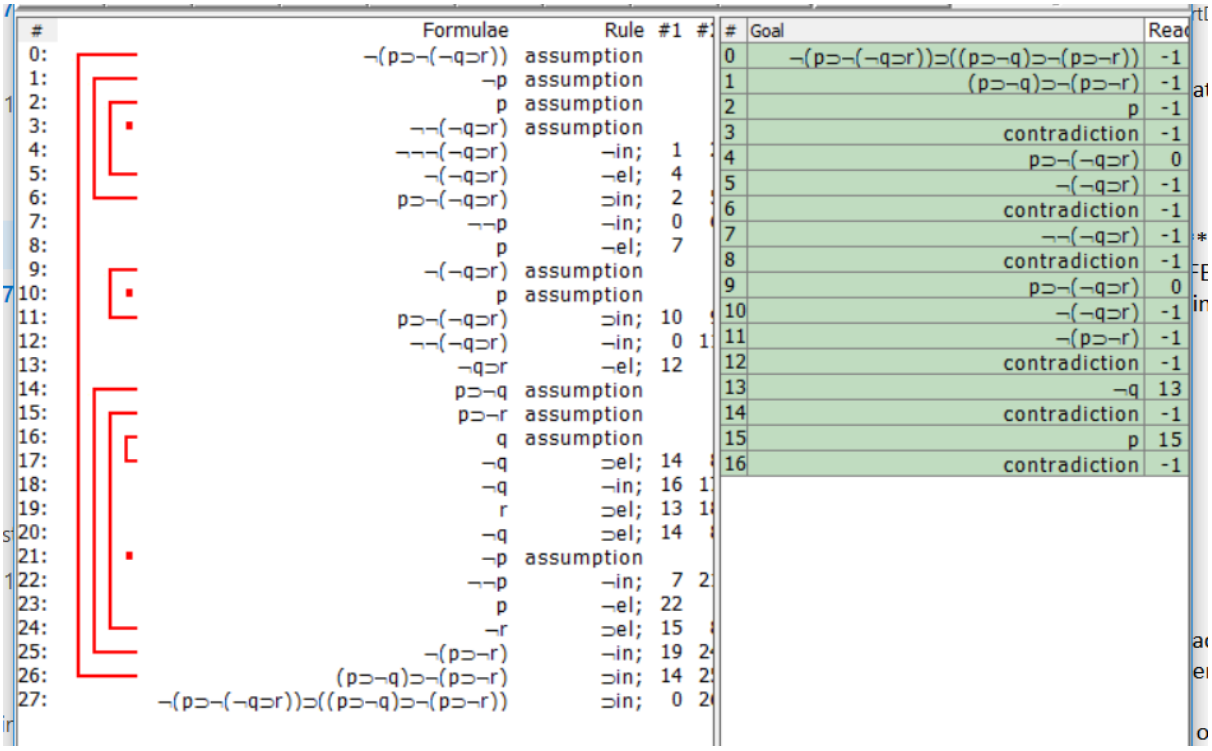
**Figure 8. Proof for p&(qVr) => (p&q)V(p&r)**



| # | Formulae | Rule | # |
|---|---|---|---|
| 0: | ¬((¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r))) | assumption | |
| 1: | ((¬p∨¬(q∨r))) | assumption | |
| 2: | ((¬p∨¬(q∨r)))∨((¬(¬p∨¬q)∨¬(¬p∨¬r))) | ∨in; | |
| 3: | ¬(¬p∨¬(q∨r)) | ¬in; | |
| 4: | ¬p | assumption | |
| 5: | ¬p∨¬(q∨r) | ∨in; | |
| 6: | ¬¬p | ¬in; | |
| 7: | p | ¬el; | |
| 8: | ¬(q∨r) | assumption | |
| 9: | ¬p∨¬(q∨r) | ∨in; | |
| 10: | ¬¬(q∨r) | ¬in; | |
| 11: | q∨r | ¬el; | 1 |
| 12: | ((¬(¬p∨¬q)∨¬(¬p∨¬r))) | assumption | |
| 13: | ((¬p∨¬(q∨r)))∨((¬(¬p∨¬q)∨¬(¬p∨¬r))) | ∨in; | 1 |
| 14: | ¬(¬(¬p∨¬q)∨¬(¬p∨¬r)) | ¬in; | |
| 15: | ¬(¬p∨¬q) | assumption | |
| 16: | ¬(¬p∨¬q)∨¬(¬p∨¬r) | ∨in; | 1 |
| 17: | ¬¬(¬p∨¬q) | ¬in; | 1 |
| 18: | ¬p∨¬q | ¬el; | 1 |
| 19: | ¬(¬p∨¬r) | assumption | |
| 20: | ¬(¬p∨¬q)∨¬(¬p∨¬r) | ∨in; | 1 |
| 21: | ¬¬(¬p∨¬r) | ¬in; | 1 |
| 22: | ¬q | ∨el; | 1 |
| 23: | ¬p∨¬r | ¬el; | 2 |
| 24: | r | ∨el; | 1 |
| 25: | ¬r | ∨el; | 2 |
| 26: | ¬¬((¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r))) | ¬in; | 2 |
| 27: | (¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r)) | ¬el; | 2 |

| # | Goal | Rea |
|---|---|---|
| 0 | (¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r)) | -1 |
| 1 | contradiction | -1 |
| 2 | ¬(¬p∨¬(q∨r)) | -1 |
| 3 | contradiction | -1 |
| 4 | (¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r)) | 0 |
| 5 | ¬p∨¬(q∨r) | -1 |
| 6 | ¬¬p | -1 |
| 7 | contradiction | -1 |
| 8 | ¬p∨¬(q∨r) | 3 |
| 9 | ¬p | -1 |
| 10 | ¬¬(q∨r) | -1 |
| 11 | contradiction | -1 |
| 12 | ¬p∨¬(q∨r) | 3 |
| 13 | ¬(q∨r) | -1 |
| 14 | ¬(¬(¬p∨¬q)∨¬(¬p∨¬r)) | -1 |
| 15 | contradiction | -1 |
| 16 | (¬p∨¬(q∨r))∨(¬(¬p∨¬q)∨¬(¬p∨¬r)) | 0 |
| 17 | ¬(¬p∨¬q)∨¬(¬p∨¬r) | -1 |
| 18 | ¬¬(¬p∨¬q) | -1 |
| 19 | contradiction | -1 |
| 20 | ¬(¬p∨¬q)∨¬(¬p∨¬r) | 14 |
| 21 | ¬(¬p∨¬q) | -1 |
| 22 | ¬¬(¬p∨¬r) | -1 |
| 23 | contradiction | -1 |
| 24 | ¬(¬p∨¬q)∨¬(¬p∨¬r) | 14 |
| 25 | ¬(¬p∨¬r) | -1 |

**Figure 9. Proof for (¬pV¬(qVr))V(¬(¬pV¬q)V¬(¬pV¬r))**



| # | Formulae | Rule | #1 | #2 |
|---|---|---|---|---|
| 0: | ¬(p⊃¬(¬q⊃r)) | assumption | | |
| 1: | ¬p | assumption | | |
| 2: | p | assumption | | |
| 3: | ¬¬(¬q⊃r) | assumption | | |
| 4: | ¬¬¬(¬q⊃r) | ¬in; | 1 | |
| 5: | ¬(¬q⊃r) | ¬el; | 4 | |
| 6: | p⊃¬(¬q⊃r) | ⊃in; | 2 | |
| 7: | ¬¬p | ¬in; | 0 | |
| 8: | p | ¬el; | 7 | |
| 9: | ¬(¬q⊃r) | assumption | | |
| 10: | p | assumption | | |
| 11: | p⊃¬(¬q⊃r) | ⊃in; | 10 | |
| 12: | ¬¬(¬q⊃r) | ¬in; | 0 | 11 |
| 13: | ¬q⊃r | ¬el; | 12 | |
| 14: | p⊃¬q | assumption | | |
| 15: | p⊃¬r | assumption | | |
| 16: | q | assumption | | |
| 17: | ¬q | ⊃el; | 14 | |
| 18: | ¬q | ¬in; | 16 | 17 |
| 19: | r | ⊃el; | 13 | 18 |
| 20: | ¬q | ⊃el; | 14 | |
| 21: | ¬p | assumption | | |
| 22: | ¬¬p | ¬in; | 7 | 21 |
| 23: | p | ¬el; | 22 | |
| 24: | ¬r | ⊃el; | 15 | |
| 25: | ¬(p⊃¬r) | ¬in; | 19 | 24 |
| 26: | (p⊃¬q)⊃¬(p⊃¬r) | ⊃in; | 14 | 25 |
| 27: | ¬(p⊃¬(¬q⊃r))⊃((p⊃¬q)⊃¬(p⊃¬r)) | ⊃in; | 0 | 26 |

| # | Goal | Reac |
|---|---|---|
| 0 | ¬(p⊃¬(¬q⊃r))⊃((p⊃¬q)⊃¬(p⊃¬r)) | -1 |
| 1 | (p⊃¬q)⊃¬(p⊃¬r) | -1 |
| 2 | p | -1 |
| 3 | contradiction | -1 |
| 4 | p⊃¬(¬q⊃r) | 0 |
| 5 | ¬(¬q⊃r) | -1 |
| 6 | contradiction | -1 |
| 7 | ¬¬(¬q⊃r) | -1 |
| 8 | contradiction | -1 |
| 9 | p⊃¬(¬q⊃r) | 0 |
| 10 | ¬(¬q⊃r) | -1 |
| 11 | ¬(p⊃¬r) | -1 |
| 12 | contradiction | -1 |
| 13 | ¬q | 13 |
| 14 | contradiction | -1 |
| 15 | p | 15 |
| 16 | contradiction | -1 |

**Figure 10. Proof for ¬(p => ¬(¬q => r)) => ((p => q) => (p => r))**

However, proofs for the equivalent formulae in Figures 8-10 are similar: the one in Figure 8 is for the system with the full set of Booleans and proofs in Figures 9 and 10 are for the systems with only {V, ¬} and {=>, ¬}.

**Section 5. Discussion** (300-600 words)

The project has been an adventure and did require the students to gain knowledge in the area. We assumed that the students' maths background gained through the study would suffice. That has been approved during the project. Our organisation of work proved to be efficient – we had regular meetings where the team has discussed proof techniques, brainstormed difficult cases and set up tasks for work in between the meetings. Everybody has access to the implemented algorithm hence could practice examples and do more experiments in their own time. The existence of the implemented algorithm helped tuning our hand made proofs looking for the strategies close to those behind the implementation. The team made significant progress in achieving the aim of the project – to investigate how changing the basic set of Booleans affects the ND proof. However, as mentioned in the "Results" section, our journey, have not proved our conjecture. At the same time, we are not surprised to see this. With no

similar research in the literature, we did not have enough ground material to make a more informed assumption and one of the main achievements of our project is that it actually brought some light on the problem. Moreover, at this stage we can also assume that NONE of the formulations of the ND system – either with the full set of Booleans or with the restricted sets has any advantage in terms of its capability to be automated. Our research within the project has provided evidence that the source of growing complexity of the proof search for various formulations of the ND systems lies deeper, in the internal structures of formulae. This draws grounds for the follow -up research to aim at looking for these patterns of internal structures of formulae.

## **Section 6. Conclusions and Recommendations** (200-300 words)

Our results have approved this in general; however, we still do not know WHERE exactly the complexity grows and WHAT are the remedies. The main lessons learned from the research were:

- In many cases, the proof of a formula using the full set of Boolean connectives could be closely mimicked using the chosen restricted set, meaning that the complexity did not usually increase substantially.
- The behaviour of a translated formula could depend strongly on choices made during the translation. For example, one of the operations which tends to be problematic in the natural deduction approach, **disjunction,** is associative; so, we can simply write (aVbVc) for either ((aVb) Vc) or (aV(bVc)). If we represent disjunction by translating (xVy) as (~x>y) throughout, those two (equivalent) formulae become (~(~a => b) => c) and (~a => (~b => c)), which behave very differently. On a similar note, commutativity means that (aVb) and (bVa) are equivalent; but if a and b are themselves complex formulae, the complexity of proving the corresponding translated formulas (~a>b) and (~b>a) can differ.
- As a consequence of the previous point, one important direction of future research is the investigation of suitable strategies for choosing an appropriate translation among the possible equivalent ones.

## **Section 7. Dissemination** (200-300 words)

The intermediate results of this project were already presented at the Research Seminar of the School of CS & Engineering in Spring 2019.
The parties that are immediately interested in our project are the Research group of ND (UoW-Moscow University) led by Dr Bolotov. The group is currently researching the ways to improve the proof search to tackle complex formulae, such as pigeon hole Principle. The results of our project will inform this work by drawing new prospects – to loop for the patterns in the internal structures of complex formulae of classical propositional logic.
The project's results are also interesting to the research community in ND. We are not at the stage of producing a solid research publication. However, the project has brought enough material to be presented as "Work in Progress" and the project team is looking for a suitable research forum for such presentation.
We *also* envisage the project outputs to be disseminated among the students of CS&Engineering School assisting their selection of final year projects.

## **Section 8. Reflection** (200-300 words)

We believe that the way the project team worked on this type of purely research oriented project – a combination of team meeting and individual research by the project participants – has been an efficient way forward and this can be recommended as an example of good practice.

The project team has been changed from the one initially planned. Two students, who were on their Final Year, found it very difficult to actively involved in the project while performing their core studying and supplementary work. This, of course, has not been expected, as everybody has been fully committed in the beginning of the project. However, the team has reacted by inviting a PhD student, Natalia Yerashenia. Reflecting this experience, we believe that for such type of research project, the participation of Final Year students is problematic, unless the topic of student's Final project is somehow close/relevant.

This has been our first experience of working in a joint student-academics research project, and the lesson learnt is to pay more attention when crating a team, explaining clearly the anticipated student's participation and what it involves.

One other strong feature of our project team was that the project's topic has been an ongoing research of Dr Bolotov research group and two academics members of the team are professionals in this area. This helped to boost the initial stage of the project to transfer necessary knowledge to the students.

**Submission Instructions:**
Please ensure the academic partner has checked a draft report and attach the cover sheet at the front of your report. A final report should be emailed to studentpartnership@westminster.ac.uk by the 24th July 2019.