Computing and Informatics, Vol. 34, 2015, 337–366

PROOF SIMPLIFICATION IN THE FRAMEWORK OF COHERENT LOGIC

Vesna Marinković

Faculty of Mathematics
University of Belgrade
Studentski trg 16
11000 Belgrade, Serbia
e-mail: `vesnap@matf.bg.ac.rs`

Abstract. The problem of proof simplification draws a lot of attention to itself across various contexts. In this paper, we present one approach for simplifying proofs constructed in the framework of coherent logic. This approach is motivated by the need for filtering-out "clean" and short proofs from proof-traces, which typically contain many irrelevant steps, and which are generated by automated theorem provers – in this case, theorem provers based on coherent logic. Such "clean" proofs can then be used for producing readable proofs in natural-language form. The proof simplification procedure consists of three transformation steps. The first one is based on the elimination of inference steps which are irrelevant for the present proof, also allowing some irrelevant branchings to be eliminated, the second one consists of lifting-up steps through the branching steps, followed by elimination of repeated steps, while the third one serves to convert proof fragments into the reductio ad absurdum form, if possible. In contrast to general simplification procedures, our proof simplification procedure is specific for a fragment of first order logic and therefore simple and easy to implement, and allows simple generation of object level proofs. We proceed to prove that this procedure is correct and terminating, and also that it never increases the size of a proof. Finally, we implement the proof simplification procedure, and provide several example proofs.

Keywords: Proof simplification, coherent logic, readable proofs, automated theorem provers, reductio ad absurdum

Mathematics Subject Classification 2010: 03-F07, 03-F20, 68-T15, 68-T27

1 INTRODUCTION

In recent decades, numerous automated theorem provers (ATPs) based on logics of different expressive powers have been developed. Their main task is the automated checking of the validity of a conjecture, and doing so as efficiently as possible[1]. As a result, they often produce only a yes/no answer to the question whether the given conjecture is a theorem or not (i.e. resolution provers, algebraic provers in geometry, etc.). For some provers this process is accompanied by certain proof-traces or genuine proof objects. However, the style of the proofs generated by ATPs is different from the style used by human mathematicians – the language used to express problems and solutions as well as the nature of solutions differ. The former generates long proofs, based on low-level arguments, while the latter would use high-level arguments [10]. As these automatically generated proofs cannot typically be read and scrutinized by a human mathematician, it can be argued that their soundness greatly depends on a piece of software [3]. Even when these proofs are made readable [5, 13, 22], they often consist of thousands of steps, many of which are irrelevant, making these proofs *de facto* useless. For instance, it may happen that automatically proving a conjecture which is effectively equal to one of the axioms results in a proof-trace containing tens of steps. In this case, obviously, the proof-trace could be shortened to the length of just one step – which would be the application of the axiom that this conjecture is equal to. All of this indicates that extraction of short and readable proofs from proof-traces would be of paramount importance for many applications, such as for:

educational purposes – "clean" formally verified proofs, could be used for automatically producing proofs in natural-language form. Proofs could be given in a block-structured natural-deduction format, or, by doing some additional transformations, in the format which is more similar to the proofs found in textbooks (however, this would be still far from the proof that a human mathematician would write). In this way we would not just get a certificate of the truth, but also a proof which would be more comprehensible for humans.

the formalization of mathematical knowledge – the main application of formalized mathematics is verification of the mathematical correctness. This is usually done using proof assistants and the ones that are today the most in use are Coq [24], Mizar [26], Isabelle/Isar [15], etc. Apart from having knowledge which is formally proven correct, it is important that this formalized knowledge is also human-understandable, and not only machine-verifiable. Automatically generated proofs, which are likely to consist of thousands of steps, are not the most desirable for building the corpus of formalized mathematical knowledge.

applications in industry – although the main goal of industry applications is simply determining whether some conjecture is a theorem or not, the industry

[1] The TPTP [23] provides a repository of problems for automated theorem provers for their testing, evaluation and comparison.

could also benefit from having short and readable proofs. For instance, a readable proof could provide an insight into where the problem appeared in either software or hardware.

It is interesting that even Hilbert in his 24th problem [25] asked for a criterion of simplicity in mathematical proofs, and the development of a theory with the power of proving that a proof is the simplest possible under given conditions[2]. In the meanwhile, the problem of proof simplification for different kind of logics was recognized as important by many researchers, and they employed different techniques in order to solve it.

In this paper, we present a simplification procedure, which transforms a correct proof in the framework of coherent logic into a proof of the same conjecture which is usually quite shorter (but not necessarily the shortest possible). This "cleaning" procedure does not lead to a smarter strategy of proof construction, but is rather syntactic – it represents an addition to the proving procedure. In contrast to general simplification procedures, since our proof simplification procedure is specific for a coherent logic, it is simple and easy to implement, and object level proofs are easily generated. Transformations which the procedure conducts are neither complex nor unexpected – actually, they do the work which humans would find tedious, or even impossible for larger proofs. Therefore, the presented transformation does not bring new proof-theory results, but rather focuses on making ATPs more suitable for certain applications.

Organization of the paper. In Section 2, some background information on coherent logic and automated theorem proving in the framework of coherent logic is given. In Section 3, a proof simplification procedure is presented and its properties are proven, while in Section 4, the implementation is briefly discussed, followed by a couple of examples. In Section 5, the related work is presented, and in Section 6, final conclusions are drawn, and some ideas for further work are given.

2 BACKGROUND

In this section, coherent logic, as the underlying logic of our proof simplification procedure, is briefly described, and a short overview of different ATPs developed for a coherent logic is given, with an emphasis on the ArgoCLP prover [22].

2.1 Coherent Logic

Coherent logic (CL) is a fragment of first-order logic in which all of the formulae are implicitly universally quantified, and are of the form:

$$A_1(\vec{x}) \wedge \ldots \wedge A_n(\vec{x}) \Rightarrow \exists \vec{y}_1 \; B_1(\vec{x}, \vec{y}_1) \vee \ldots \vee \exists \vec{y}_m \; B_m(\vec{x}, \vec{y}_m)$$

[2] For some reason, Hilbert left this problem out of his seminal list of 23 important mathematical problems.

where $n \geq 0$, $m \geq 0$, $\vec{x}$ and $\vec{y}_j, 1 \leq j \leq m$ denote vectors of variables, $A_i, 1 \leq i \leq n$ denote atomic formulae, while $B_j, 1 \leq j \leq m$ denote conjunctions of atomic formulae. It can be considered as an extension of resolution logic, but in contrast to resolution, in CL the conjecture is not being changed (using clausification or Skolemization), but directly proved. This is specially important in case when we are not only interested in the truth value of the statement, but rather in the proof of the statement. In principle, in CL negations are not allowed; however they can be used in a limited way – for each predicate symbol R additional predicate symbol $\overline{R}$ is introduced and the following additional axioms are used: $R(x) \vee \overline{R}(x)$ and $R(x) \wedge \overline{R}(x) \rightarrow \bot$.

Skolem was the first to describe CL and the corresponding proof procedure [21]. In recent years, CL attracted new attention because of its beneficial features. It is well-suited for producing readable, as well as formal proofs, since it has a constructive proof system based on forward reasoning, which enables easy generation of proof objects [5, 9]. Its significance lies in the fact that a number of problems in different theories can be formulated directly in CL [5]. On the other hand, every first order logic formula can be translated into CL and then solved using solvers for CL [18]. As a corollary of the translation, CL is undecidable but semidecidable [5].

We will call a coherent formula branching (with m branches) if $m \geq 2$, and non-branching otherwise[3]. In the following text, we refer to formulae $A_1(\vec{x}), \ldots, A_n(\vec{x})$ as the premises, and to the formula $\exists \vec{y}_1\ B_1(\vec{x}, \vec{y}_1) \vee \ldots \vee \exists \vec{y}_m\ B_m(\vec{x}, \vec{y}_m)$ as the conclusion of a given formula.

The following set of inference rules (basically, a subset of Gentzen natural deduction rules, and also another version of the rules given in [4]) are the only rules used in CL[4]:

$$\frac{A_1(\vec{a}) \wedge \ldots \wedge A_n(\vec{a})}{A_1(\vec{a}), \ldots, A_n(\vec{a})}\wedge E \qquad \frac{A_1 \vee \ldots \vee A_n \quad \begin{array}{c}[A_1]\\ \vdots\; c_1\\ B\end{array} \quad \ldots \quad \begin{array}{c}[A_n]\\ \vdots\; c_n\\ B\end{array}}{B}\vee E \qquad \frac{\bot}{A}\,efq$$

$$\frac{A_1(\vec{a}), \ldots, A_n(\vec{a}) \quad A_1(\vec{x}) \wedge \ldots \wedge A_n(\vec{x}) \Rightarrow \exists \vec{y}_1\ B_1(\vec{x}, \vec{y}_1) \vee \ldots \vee \exists \vec{y}_m\ B_m(\vec{x}, \vec{y}_m)}{B_1(\vec{a}, \vec{w}_1) \vee \ldots \vee B_m(\vec{a}, \vec{w}_m)}\,ax$$

where $\vec{a}$ denotes a vector of constants, and each $\vec{w}_j, 1 \leq j \leq m$ denotes a vector of witnesses (these are fresh constants, neither appearing in axioms used, nor in the conjecture), while $c_i, 1 \leq i \leq n$ denote chains of inference steps. All of the rules are given in natural deduction style. The rule (ax) is applicable only if there are no vectors of witnesses such that the conclusion already holds.

[3] Non-branching formula corresponds to what is typically called Horn formula, while branching formula corresponds to non-Horn.

[4] Prover which uses only these rules can be sound; the prover which, in addition, uses iterative deepening mechanism described in Section 2.2 can be made complete.

A proof-search tree in CL is a tree, for which the following holds:

(P1) each node is assigned a set of *derived ground atomic formulae* (*derived facts*);

(P2) each node, except the leaf-nodes, is assigned an axiom application, such that all of the facts from its premises appear in the set of derived facts of that node. The set of derived facts of a node is the union of the set of derived facts of its parent node and the facts from the conclusion of the axiom application of the parent node (this corresponds to the (ax) rule). If a node is assigned a branching axiom with m branches, then that node has m child-nodes, and the facts from the i^{th} branch of the conclusion of the axiom appear in the set of derived facts of the i^{th} child-node.

Assuming that the proof search procedure is sound and complete, a proof tree exists if and only if a proof-search tree exists. In the text which follows, we will sometimes treat these two concepts equally, while making distinctions between them when necessary.

A coherent formula

$$A_1(\vec{x}) \wedge \ldots \wedge A_n(\vec{x}) \Rightarrow \exists \vec{y}_1\ B_1(\vec{x}, \vec{y}_1) \vee \ldots \vee \exists \vec{y}_m\ B_m(\vec{x}, \vec{y}_m)$$

is a *CL-theorem* if there exist a proof-search tree, for which the following holds:

(T1) the set of derived facts of the root-node consists of instantiated premises $A_1(\vec{a})$, $\ldots, A_n(\vec{a})$ of the conjecture being proven, where $\vec{a}$ denotes a vector of fresh constants;

(T2) the set of derived facts of each leaf-node contains either all of the conjuncts of the formula $B_j(\vec{a}, \vec{w_j})$, for some j $(1 \leq j \leq m)$ and for some vector of constants $\vec{w_j}$, or a contradiction.

2.2 Automated Theorem Proving in CL

In recent years, Bezem and his coauthors revived an interest for CL. Bezem and Coquand [5] developed, in Prolog, a CL prover which generates proof objects in Coq. Berghofer and Bezem developed, in ML, an internal prover for CL in Isabelle. To our knowledge, the first ATP using CL was developed by Janičić and Kordić [13], and it was used for solving theorems from the field of Euclidean geometry. Its extension, but also a significant improvement, is the ArgoCLP prover [22]. It is an ATP based on CL, which uses the inference system given in 2.1. Its proving procedure consists of a forward chaining mechanism with iterative deepening. Constants are enumerated according to the point of their appearance during the proof search. A dedicated counter is introduced, and is assigned a value of the maximal permitted value of the constant which can be used within an axiom application. Initially, it is set to the number of constants appearing in the premises of the conjecture, and it increases each time, when there is no axiom that can be applied. The prover is generic, and it outputs a formal proof in Isabelle/Isar form, as well as a proof in the natural-language form.

3 TRANSFORMATION PROCEDURE

The goal of the transformation procedure is to transform the correct proof in the CL framework into a proof of the same conjecture, typically much shorter than the original one. The procedure consists of three steps:

- eliminating irrelevant nodes (*EIN transformation*),
- lifting-up nodes through the branching nodes and eliminating repeated nodes (*LUP & ERN transformation*),
- reductio ad absurdum (*RAA transformation*).

The *EIN* transformation falls into the category of contracting transformations, since it removes irrelevant nodes from the proof, thus possibly reducing the size of a proof. The *LUP & ERN* transformation lifts up nodes through the branching nodes to the highest possible position in the proof tree, and then, if there is multiple occurrence of a node in a part of the proof between two branching nodes, it is being reduced to only one appearance. In this way if there is a node appearing in different branches of the same branching and if it does not depend on any of branching assumptions, it would occur just once in a simplified proof tree above the branching node, thus reducing the size of the proof. So, it is a contracting transformation, as well as a restructuring one. On the other hand, the *RAA* transformation belongs to the category of restructuring transformations, as it rearranges the structure of the proof tree in order to make it more readable and more alike to traditional proofs. It converts the proof tree into the reductio ad absurdum form, if possible.

The proof simplification transformation (*PST transformation*) is defined as a composition of *RAA*, *LUP & ERN* and *EIN* transformation:

$$PST = RAA \circ LUP\,\&\,ERN \circ EIN$$

It takes place post-festum, once a conjecture has already been proven.

To improve the readability of the paper, we introduce the following definitions:

Definition 1. *proof*(ϕ, Δ) denotes that Δ is a proof of the theorem ϕ.

Definition 2. The *size* of a proof tree is the number of nodes in the tree.

3.1 The EIN Transformation

Let us suppose that a proof-trace constructed using inference rules from 2.1 is given. In the following text, instead of considering a proof-trace as a sequence of inference steps, we will be considering its corresponding proof tree. For each node in the proof tree, it is determined whether it is relevant for the proof or not. All of the nodes which are irrelevant for the present proof are eliminated from it.

Definition 3. A node in a proof tree is *relevant* if the axiom application occurring within it derives at least one fact (in each of the branches, if the node is branching) which is needed for deriving the conclusion of the conjecture being proven (or a contradiction) in all of the corresponding leaf-nodes. Otherwise, we say that the node is *irrelevant.*

In order to determine if a node is relevant, the set R of relevant objects in the set of currently processed nodes of the proof tree is maintained. Objects which are allowed to appear in the set R are only atomic ground formulae (facts). Initially, R is set to be an empty set. Since for each node it has to be determined if the facts it derives are used in its subtree, it is necessary to gather all of the information about its subtree before processing a node. So, the *EIN* procedure starts from the last node in the proof tree (the right-most leaf-node of the tree). Afterwards, the proof tree is traversed backwards, from the last node of the proof tree to the first one, i.e. opposite to DFS traversal. Each node is processed according to its type – whether it is non-branching or branching.

Processing a non-branching node: the axiom application of a non-branching node is of the following form:

$$A_1(\vec{a}) \wedge \ldots \wedge A_n(\vec{a}) \Rightarrow B^1(\vec{a},\vec{w}) \wedge \ldots \wedge B^k(\vec{a},\vec{w})$$

where $n \geq 0$, $k \geq 0$, $\vec{a}$ denotes a vector of constants, $\vec{w}$ denotes a vector of witnesses (which can also be an empty vector), and A_i, $1 \leq i \leq n$, as well as B^j, $1 \leq j \leq k$, denote atomic formulae. In this case, if, for some j, it holds that the fact $B^j(\vec{a},\vec{w})$ is found in the set of relevant objects R at the moment of node traversal, then this node is found to be relevant, and the set R is being updated in the following manner: all of the facts from its conclusion ($B^j(\vec{a},\vec{w})$, $1 \leq j \leq k$) are removed from R (if they are found there), while all of the facts from its premises ($A_i(\vec{a})$, $1 \leq i \leq n$) are added to the set R. Otherwise, the node is denoted as irrelevant, and no changes are made to the set R.

As soon as the theorem is proven in a certain branch, that branch is closed. Therefore, all of the leaf-nodes of the proof-search tree (ones deriving the conclusion or a contradiction) are *a priori* marked as relevant and, accordingly, all of the facts from their premises are added to the set R in the moment of their traversal.

Example 1. A proof of a theorem $T : A(x) \wedge B(y) \rightarrow F(x,y)$ using the axiom system: $A1 : A(x) \rightarrow C(x)$, $A2 : A(x) \rightarrow D(x) \wedge E(x)$, $A3 : C(x) \wedge B(y) \rightarrow F(x,y)$ is presented in Figure 1 a). For clearer presentation, a node is presented by its axiom application only (along with the axiom label), while its set of derived facts is omitted. Initially, the conjecture is instantiated by introducing two new constants a and b and replacing universally quantified variables x and y by these constants. The goal is to prove the conclusion $F(a,b)$ using given set of axioms and the two assumptions $A(a)$ and $B(b)$. The nodes are enumerated in the order they are visited during a depth-first traversal of the proof tree.

$$\begin{array}{c}
A(a), B(b) \vdash F(a,b) \\ \hline
n_1 : A(a) \rightarrow C(a) \ (A1) \\
| \\
n_2 : A(a) \rightarrow D(a) \wedge E(a) \ (A2) \\
| \\
n_3 : C(a) \wedge B(b) \rightarrow F(a,b) \ (A3) \\
\text{a)}
\end{array}
\qquad
\begin{array}{c}
A(a), B(b) \vdash F(a,b) \\ \hline
n_1 : A(a) \rightarrow C(a) \ (A1) \\
| \\
n_3 : C(a) \wedge B(b) \rightarrow F(a,b) \ (A3) \\
\\
\\
\text{b)}
\end{array}$$

Figure 1. Eliminating non-branching nodes: a) the non-simplified proof, b) the simplified proof

The *EIN* procedure starts from the leaf-node n_3 of the proof tree, and the set R is initialized to $\{C(a), B(b)\}$. Next, node n_2 is processed, and it is deemed irrelevant, as none of the facts from its conclusion (neither $D(a)$ nor $E(a)$) are present in the set R. Finally, the root-node n_1 is relevant since the fact $C(a)$ appears in the set R, and the fact $C(a)$ is substituted by the fact $A(a)$ in R. So, at the end, the set R contains the facts $\{A(a), B(b)\}$, which are the premises of the theorem T. The simplified proof of the theorem T is given in Figure 1 b).

Processing a branching node: the axiom application of a branching node is of the following form:

$$A_1(\vec{a}) \wedge \ldots \wedge A_n(\vec{a}) \Rightarrow B_1(\vec{a}, \vec{w}_1) \vee \ldots \vee B_m(\vec{a}, \vec{w}_m)$$

where $n \geq 0$, $m \geq 2$, $\vec{a}$ denotes a vector of constants, $\vec{w_i}, 1 \leq i \leq m$, denote vectors of witnesses (which can be also an empty vector), $A_i, 1 \leq i \leq n$, denote atomic formulae, while $B_j, 1 \leq j \leq m$, denote conjunctions of atomic formulae: $B_j = B_j^1 \wedge \ldots \wedge B_j^{k_j}$. A branching node has the same number of child-nodes as the number of disjuncts in its axiom, and we will refer to each child-node as an *assuming node* (as it "assumes" that B_j holds), and to B_j as a *branching assumption.* A branching node is relevant only if the derivations performed in every branch need their branching assumptions (B_j) for deriving the conclusion of the conjecture (or a contradiction). This actually means that, for each of the branches, it holds that at least one fact from its branching assumption is found in the set R at the moment of processing the assuming node of that branch. If that is not the case, then the branching is obsolete, the branching node along with other branches can be eliminated from the proof tree, and from that point, only the derivation conducted in the branch which is not using its branching assumption can be kept in the proof tree[5]. In the case in which that branch is closed by a contradiction, it can be concluded that the starting conjecture is contradictory. In this case, no changes are made to the set R.

[5] If there is more than one branch which is not using its branching assumption, any one of these branches can be chosen to be kept in the proof.

On the other hand, if the branching node is marked as relevant, then all of the facts from the conclusion (from each $B_j(\vec{a}, \vec{w}_j)$, $1 \leq j \leq m$) are erased from the set R, while all of the facts from its premises ($A_i(\vec{a})$, $1 \leq i \leq n$) are added to the set R.

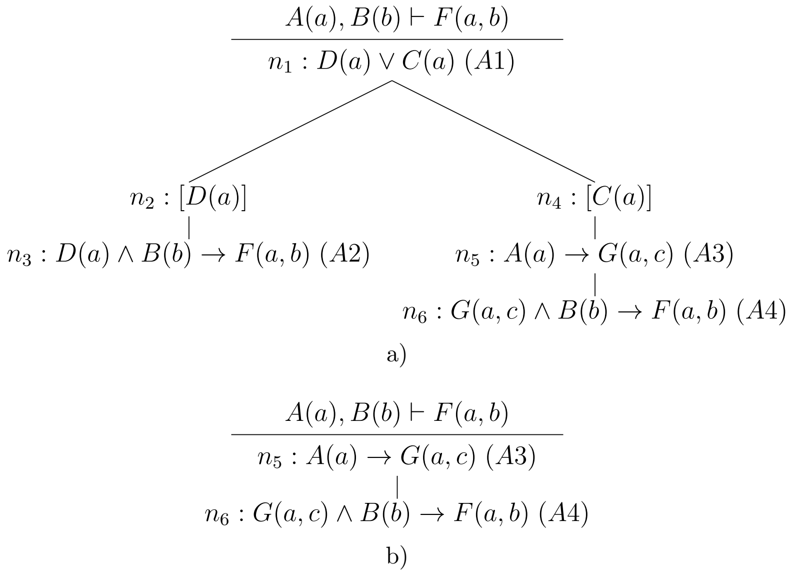
$A(a), B(b) \vdash F(a, b)$
$n_1 : D(a) \vee C(a)$ $(A1)$
$n_2 : [D(a)]$
$n_3 : D(a) \wedge B(b) \to F(a, b)$ $(A2)$
$n_4 : [C(a)]$
$n_5 : A(a) \to G(a, c)$ $(A3)$
$n_6 : G(a, c) \wedge B(b) \to F(a, b)$ $(A4)$
a)
$A(a), B(b) \vdash F(a, b)$
$n_5 : A(a) \to G(a, c)$ $(A3)$
$n_6 : G(a, c) \wedge B(b) \to F(a, b)$ $(A4)$
b)

Figure 2. Eliminating branching nodes: a) nonsimplified proof, b) simplified proof

Example 2. Let us consider a proof of the same theorem $T : A(x) \wedge B(y) \to F(x, y)$ as in the previous example, within another axiom system $A1 : D(x) \vee C(x)$, $A2 : D(x) \wedge B(y) \to F(x, y)$, $A3 : A(x) \to G(x, y)$, $A4 : G(x, y) \wedge B(z) \to F(x, z)$, which is given in Figure 2 a). Initially, R is set to $\{G(a, c), B(b)\}$ and during the traversal of the node n_5, the conclusion fact $G(a, c)$ is erased and the fact $A(a)$ is added to the set R. It can be seen that, at the moment of processing the assuming node n_4, its branching assumption $C(a)$ is not found in the set R, therefore the whole proof can be substituted by a shorter, non-branching one, which consists only from the nodes n_5 and n_6. Once the transformation procedure has terminated (the root-node of the proof tree has been processed), the set R contains only instantiated premises of the conjecture being proven (meaning that all of the necessary information needed to prove the conjecture are given in its premises). More precisely, the set R is equal to a subset of instantiated premises; if it is a proper subset, then some premises of the conjecture are redundant.

In Algorithm 1, we give the pseudocode of the *EIN* transformation. We assume that each node has a field *children_num* containing number of child-nodes, as well as a vector *child* containing child-nodes. Also, R is a global variable initially set to be the empty set.

Algorithm 1 $EIN(r)$

```
Require: $r$ - root-node of a proof tree
Ensure: simplified proof tree of the same conjecture
  $children\_num \leftarrow r.children\_num$
  $A \leftarrow \bigcup_{i=1..n} A_i(a)$ {$A_i(a)$, $1 \le i \le n$ are the premises of the axiom application of the node $r$}
  if $children\_num = 0$ then {a leaf-node}
    $R \leftarrow R \cup A$ {updating the set of relevant objects}
    return $r$ {all of the leaf-nodes are initially marked as relevant}
  else if $children\_num = 1$ then {a non-branching node}
    $r_0 \leftarrow EIN(r.child[0])$ {$r_0$ is a root-node of the simplified subtree}
    $B \leftarrow \bigcup_{j=1..k} B^j(a,w)$ {$B = \bigwedge_{1 \le j \le k} B^j(a,w)$ is the conclusion of the axiom application of the node $r$}
    if $R \cap B \neq \emptyset$ then {node $r$ is relevant}
      $r.child[0] \leftarrow r_0$
      $R \leftarrow (R \setminus B) \cup A$ {updating the set of relevant objects}
      return $r$
    else {node $r$ is irrelevant}
      return $r_0$
    end if
  else {a branching node}
    for $j = children\_num - 1$ downto 0 do
      $R \leftarrow \emptyset$
      $B_j \leftarrow \bigcup_{k=1..k_j} B_j^k(a,w_j)$ {$B_j = \bigwedge_{1 \le k \le k_j} B_j^k(a,w_j)$ is the $j^{\text{th}}$ branching assumption of the axiom application of the node $r$}
      $r_j \leftarrow EIN(r.child[j])$ {$r_j$ is a root-node of the simplified $j^{\text{th}}$ subtree}
      if $R \cap B_j \neq \emptyset$ then {this branch needs its assumption}
        $(r.child[j]) \leftarrow r_j$
        $R_j \leftarrow R$ {the set of relevant facts for this branch is kept in variable $R_j$}
      else {this branch does not need its assumption}
        return $r_j$ {other branches and the branching node are erased from the proof tree}
      end if
    end for
    $R \leftarrow \bigcup_{j=1..children\_num} R_j \setminus A$
    return $r$ {all of the branches are relevant, the branching node is kept}
  end if
```

Example 3. A set of axioms $(A1)$–$(A9)$ and a conjecture (C) are given:

$(A1)$: $\forall x\, A(x) \rightarrow E(x)$

$(A2)$: $\forall x\, A(x) \rightarrow I(x)$

$(A3)$: $\forall x\, B(x) \rightarrow H(x)$

(*A*4**):** $\forall x\, C(x) \wedge I(x) \rightarrow J(x)$

(*A*5**):** $\forall x\, \forall y\, \forall z\, G(x,y) \wedge B(z) \rightarrow\ F(x,z)$

(*A*6**):** $\forall x\, \forall y\, D(x) \wedge H(y) \rightarrow F(x,y)$

(*A*7**):** $\forall x\, D(x) \vee C(x)$

(*A*8**):** $\forall x\, (B(x) \wedge K(x)) \vee D(x)$

(*A*9**):** $\forall x\, A(x) \wedge B(x) \rightarrow \exists y\, G(x,y)$

(*C***):** $\forall x\, \forall y\, A(x) \wedge B(y) \rightarrow F(x,y)$

A proof of conjecture C, using the axiom system given above, is shown in Figure 3. In this example, all of branches are closed by the derivation of the conclusion of the conjecture ($F(a,b)$). As it can be seen, the root-node n_1 in the generated proof-search tree is not relevant for the proof since the derived fact $E(a)$ is not used anywhere further in the proof. Similarly, the node n_8 is not relevant. So, the proof could be simplified.

The transformation procedure would find the first branching node n_4 irrelevant, since the branching assumption $C(a)$ is not used for deriving the conclusion in the second branch (however, it is used in the node n_8, but, as it is already noticed, this node is irrelevant for the proof). On the contrary, the second branching node n_9 would be marked as relevant since there exists a fact in both branching assumptions ($B(a)$ in the first one, and $D(a)$ in the second) which are used in the derivation of the formula $F(a,b)$ (each of them in its own branch). Therefore, only the second branch of the first branching is to be kept in the “clean” proof, with all of the irrelevant non-branching nodes (n_1, n_2, n_8, n_{11}) eliminated. Finally, after the *EIN* transformation stops, the proof would have the tree-structure shown in Figure 4.

Alternatively, a derivation conducted in the simplified proof can be given also in natural deduction style, as in (1):

$$\frac{(B(a) \wedge K(a)) \vee D(a) \quad (2) \quad (3)}{F(a,b)}\ \vee E \tag{1}$$

where (2) and (3) denote the following proof-fragments:

$$\cfrac{\cfrac{A(a) \quad \cfrac{[B(a) \wedge K(a)]}{B(a)}\ \wedge E \quad A(x) \wedge B(x) \Rightarrow G(x,y)}{G(a,c)}\ ax \quad B(b) \quad G(x,y) \wedge B(z) \Rightarrow F(x,z)}{F(a,b)}\ ax \tag{2}$$

$$\cfrac{[D(a)] \quad \cfrac{B(b) \quad B(x) \Rightarrow H(x)}{H(b)}\ ax \quad D(x) \wedge H(y) \Rightarrow F(x,y)}{F(a,b)}\ ax \tag{3}$$

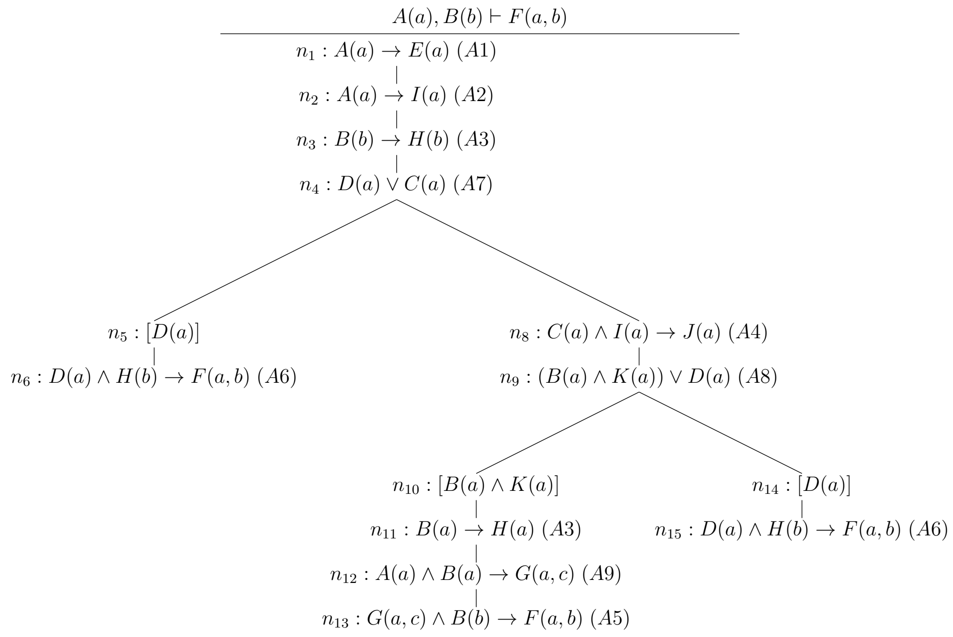
$A(a), B(b) \vdash F(a,b)$
$n_1 : A(a) \to E(a)\ (A1)$
$n_2 : A(a) \to I(a)\ (A2)$
$n_3 : B(b) \to H(b)\ (A3)$
$n_4 : D(a) \vee C(a)\ (A7)$
$n_5 : [D(a)]$
$n_6 : D(a) \wedge H(b) \to F(a,b)\ (A6)$
$n_8 : C(a) \wedge I(a) \to J(a)\ (A4)$
$n_9 : (B(a) \wedge K(a)) \vee D(a)\ (A8)$
$n_{10} : [B(a) \wedge K(a)]$
$n_{11} : B(a) \to H(a)\ (A3)$
$n_{12} : A(a) \wedge B(a) \to G(a,c)\ (A9)$
$n_{13} : G(a,c) \wedge B(b) \to F(a,b)\ (A5)$
$n_{14} : [D(a)]$
$n_{15} : D(a) \wedge H(b) \to F(a,b)\ (A6)$

Figure 3. Initial proof

3.2 Properties of the EIN Transformation

In this section, some important properties of the *EIN* transformation are shown. They state that the *EIN* transformation outputs a valid proof of the same conjecture, which is at most the same size as the initial one, and which has no irrelevant steps.

Lemma 1 (Correctness of EIN)**.** If $proof(\phi, \Delta)$ then $proof(\phi, EIN(\Delta))$.

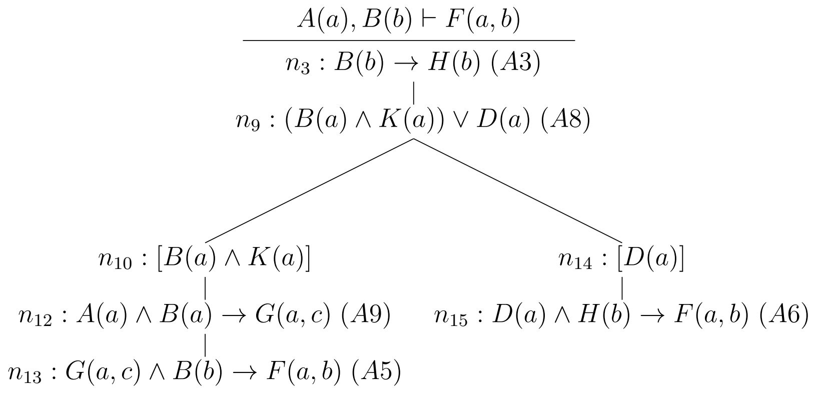
$A(a), B(b) \vdash F(a,b)$
$n_3 : B(b) \to H(b)\ (A3)$
$n_9 : (B(a) \wedge K(a)) \vee D(a)\ (A8)$
$n_{10} : [B(a) \wedge K(a)]$
$n_{12} : A(a) \wedge B(a) \to G(a,c)\ (A9)$
$n_{13} : G(a,c) \wedge B(b) \to F(a,b)\ (A5)$
$n_{14} : [D(a)]$
$n_{15} : D(a) \wedge H(b) \to F(a,b)\ (A6)$

Figure 4. Simplified proof

Proof. First, we need to prove that the newly constructed object $EIN(\Delta)$ is a proof. We will show that $EIN(\Delta)$ satisfies the properties (P1) and (P2). The property (P1) trivially holds, since it already holds for Δ, and the set of nodes of $EIN(\Delta)$ is a subset of the set of nodes of Δ.

Let us suppose that $EIN(\Delta)$ contains a node such that there exists a fact F from the premises of its axiom, which is not found in the set of derived facts of that node. Since the node belongs to $EIN(\Delta)$, it means that during the transformation process and the backward traversal of the tree, after it had been processed, the fact F has been added to the set of relevant objects. Accordingly, a node which contains the application of an axiom which concludes F somewhere earlier in Δ (and it has to exist in Δ, since it is a valid proof tree) would be added to $EIN(\Delta)$. Hence, the fact F is found in the set of derived facts of that node, which leads to a contradiction. Therefore, our assumption was wrong, meaning that for each node in $EIN(\Delta)$ all of the facts from the premises of its axiom are found in the set of derived facts of that node. So, $EIN(\Delta)$ satisfies the property (P2) also, and is, indeed, a valid proof.

Is it a proof of a given conjecture ϕ? We need to show that properties (T1) and (T2) hold for the proof $EIN(\Delta)$ and the formula ϕ.

Let us denote the root-node of $EIN(\Delta)$ with n_1 and the root-node of Δ with n_2. Assume that the set of derived facts of the node n_1 contains a fact F which is not from the set of premises of the conjecture ϕ. Since the set of derived facts of the node n_2 does not contain F, it holds for proof Δ that the node n_1 is a descendant of the node n_2 and one of the nodes on the path from the node n_1 to the node n_2 concluded the fact F. But in that case, according to the transformation specification, that node would be added to $EIN(\Delta)$, and since it is an ancestor of the node n_1, the node n_1 would not be a root-node of $EIN(\Delta)$. A contradiction. Therefore, the root-node of $EIN(\Delta)$ contains only the facts from the premises of the conjecture ϕ, thus $EIN(\Delta)$ satisfies the property (T1).

According to the specification of the transformation, the set of leaf-nodes of $EIN(\Delta)$ is a subset of the set of leaf-nodes of Δ (since each leaf-node is initially set to be relevant and can only be eliminated from the simplified proof tree together with the entire branch), hence $EIN(\Delta)$ satisfies the property (T2). So, $EIN(\Delta)$ is a proof of the conjecture ϕ. □

Lemma 2 (Termination of EIN)**.** The procedure of eliminating irrelevant nodes is terminating.

Proof. The *EIN* transformation is based on proof tree traversal and the processing of each node of the tree is done only once, and in constant time. Moreover, in the case when there is a branching node in the proof tree which is not relevant and the branch which is kept is not the first one, the nodes from the branches preceding the one kept in the proof tree will not be traversed. More precisely, if we denote by $T(n)$ the total number of recursive calls for the tree having n nodes, then the following holds:

- $T(1) = 1$,
- $T(n) \leq \sum_i T(n_i) + 1$, where $\sum n_i = n - 1$.

This gives us that $T(n) \leq n - 1$ and this can be showed using induction. Since each node is processed in constant time this implies that the whole procedure works in $O(n)$ time. So, the *EIN* procedure does terminate, and its time complexity is linear w.r.t. the size of the input proof tree. □

Lemma 3 (Idempotence of EIN)**.** The procedure of eliminating irrelevant nodes does not introduce irrelevant nodes, therefore: $EIN(EIN(\Delta)) = EIN(\Delta)$

Proof. Let us denote $EIN(\Delta)$ with Δ_1. Our goal is to prove that $EIN(\Delta_1) = \Delta_1$, i.e. that all of the nodes in the proof Δ_1 are relevant for the proof Δ_1. This can be proven by using mathematical induction on the height of the node in the proof tree.

1. base case: all of the leaf-nodes (nodes of height 0) of Δ_1 are relevant, by the specification of the procedure
2. inductive step: assuming that this property holds for all of the nodes of height k, we show that it holds for a node n of height $k + 1$. Recall that the set R is not changed during the processing of irrelevant nodes, and that the relevance of a node depends only on its descendants. All of the descendants of the node n are relevant by inductive hypothesis (since their height in the proof tree is smaller), which gives that during the construction of $EIN(\Delta_1)$ at the moment of traversal of node n the set R contains the same facts it had contained during the construction of Δ_1. Considering that the node n was set to be relevant then, the same is concluded now. Therefore, the node n (of height $k + 1$) is relevant.

Therefore, all steps in the proof Δ_1 are relevant for the proof Δ_1, meaning that $EIN(\Delta_1) = \Delta_1$. □

Lemma 4 (Non-increasing size of EIN)**.** $\mathit{size}(EIN(\Delta)) \leq \mathit{size}(\Delta)$.

Proof. The *EIN* transformation affects the proof in the sense that it can erase some of the nodes from the proof tree (sometimes even complete branches), but never add a node. Therefore, it holds that $\mathit{size}(EIN(\Delta)) \leq \mathit{size}(\Delta)$. □

3.3 The LUP & ERN Transformation

We will call a subtree $n_i, n_{i+1}, \ldots, n_k$ of the proof tree $n_1, n_2 \ldots, n_s$ *linear* if none of the nodes in the subtree is a branching node. Linear subtree is *maximal* if it cannot be extended by any of its subsequent nodes in a proof tree (n_{i-1} or n_{k+1}) and still remain linear.

After the *EIN* transformation stops, there will be no more irrelevant nodes in the proof tree. However, the principle of parsimony could still be violated since the same fact could be derived in different branches of the same branching. In case when the node which derives that fact does not use neither the branching assumption, nor any conclusion derived on the path from the branching assumption to that node, it could be lifted-up above the branching node. This becomes especially useful when there are more occurrences of the same node in different branches, since once they are lifted-up above the branching, their multiple occurrence could be replaced with just one, decreasing the size of the proof tree.

For this step of simplification procedure we will use a dependency matrix M_d, which is a quadratic matrix whose dimension is equal to the size of proof tree obtained after the *EIN* transformation. It is initially set to zero matrix. Assume that nodes are enumerated in the order they are visited during a depth-first traversal of the proof tree. For each node n_i and for each fact from its set of premises, the node n_j preceding it which derives that fact is found and $M_d(i, j)$ is set to 1. Afterwards, for each row r of the matrix M_d the maximal index c of the column containing 1 is found, and in case when there exists a branching node between nodes n_c and n_r the node n_r is erased, and inserted at the position preceding that branching node. This is repeated as far as there are candidate nodes for switching their position. Finally, for each maximal linear subtree of the proof tree possible multiple occurrences of nodes are identified and eliminated, leaving just one occurrence of each node.

Example 4. Let us consider a proof of the theorem $T : A(x) \wedge B(y) \to F(x, y, z)$, within the axiom system $A1 : A(x) \to H(x)$, $A2 : C(x) \vee D(x)$, $A3 : C(x) \wedge E(y, z) \to F(x, y, z)$, $A4 : D(x) \wedge E(y, z) \to F(x, y, z)$, $A5 : B(x) \to E(x, y)$, which is given in Figure 5. The node n_4 could be lifted up to the position above the branching node n_2, since its set of instantiated premises $(B(b))$ does not depend neither on the branching assumption $(C(a))$, nor any of the nodes in between the branching assumption and the node itself. The same argument applies to the node n_7. So, the intermediate proof tree given in Figure 6 would be obtained. Afterwards, the two occurrences of the same node (n_2 and n_3 in new notation) would be shortened to only one, thus obtaining the proof tree shown in Figure 7.

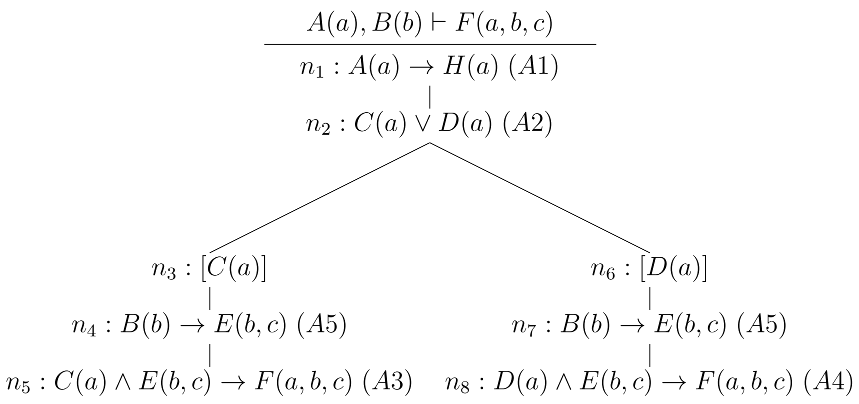
$A(a), B(b) \vdash F(a, b, c)$
$n_1 : A(a) \to H(a)$ $(A1)$
$n_2 : C(a) \vee D(a)$ $(A2)$
$n_3 : [C(a)]$
$n_4 : B(b) \to E(b, c)$ $(A5)$
$n_5 : C(a) \wedge E(b, c) \to F(a, b, c)$ $(A3)$
$n_6 : [D(a)]$
$n_7 : B(b) \to E(b, c)$ $(A5)$
$n_8 : D(a) \wedge E(b, c) \to F(a, b, c)$ $(A4)$

Figure 5. Proof after *EIN* transformation

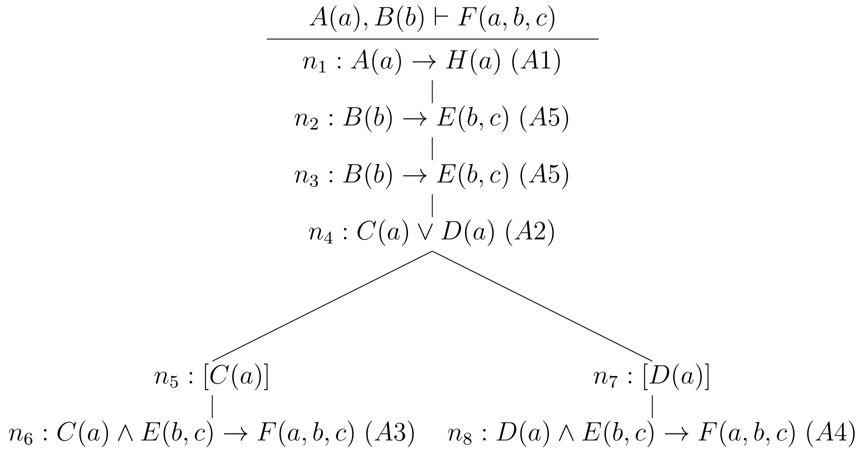
$A(a), B(b) \vdash F(a,b,c)$
$n_1 : A(a) \rightarrow H(a)$ $(A1)$
$n_2 : B(b) \rightarrow E(b,c)$ $(A5)$
$n_3 : B(b) \rightarrow E(b,c)$ $(A5)$
$n_4 : C(a) \vee D(a)$ $(A2)$
$n_5 : [C(a)]$
$n_7 : [D(a)]$
$n_6 : C(a) \wedge E(b,c) \rightarrow F(a,b,c)$ $(A3)$
$n_8 : D(a) \wedge E(b,c) \rightarrow F(a,b,c)$ $(A4)$

Figure 6. Proof after *LUP* transformation

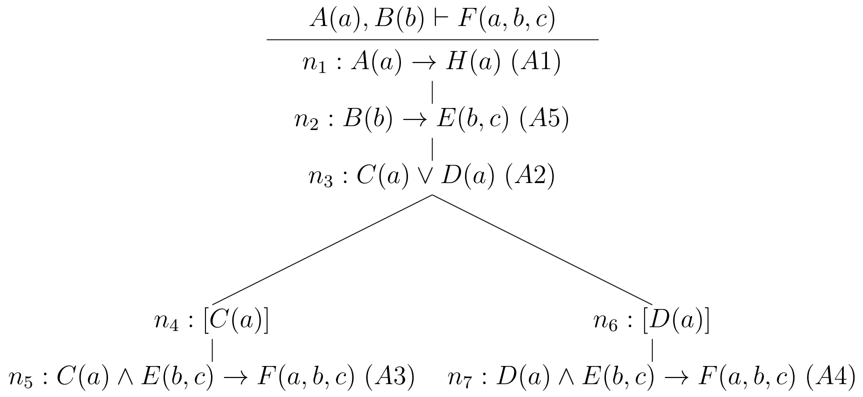
$A(a), B(b) \vdash F(a,b,c)$
$n_1 : A(a) \rightarrow H(a)$ $(A1)$
$n_2 : B(b) \rightarrow E(b,c)$ $(A5)$
$n_3 : C(a) \vee D(a)$ $(A2)$
$n_4 : [C(a)]$
$n_6 : [D(a)]$
$n_5 : C(a) \wedge E(b,c) \rightarrow F(a,b,c)$ $(A3)$
$n_7 : D(a) \wedge E(b,c) \rightarrow F(a,b,c)$ $(A4)$

Figure 7. Proof after *ERN* transformation

3.4 Properties of the LUP & ERN Transformation

In this section, the same properties which have been proven for the *EIN* transformation are shown to be true for *LUP & ERN* transformation as well.

Lemma 5 (Correctness of *LUP* & *ERN*)**.** If *proof*(ϕ, Δ) then *proof*$(\phi, LUP\,\&\, ERN(\Delta))$.

Proof. All nodes of the new tree are nodes of the initial proof tree, therefore property (P1) trivially holds. Property (P2) also holds since all nodes of the initial proof tree that used the facts which that node derives are also below it in a new proof tree, and all facts from its set of premises are derived above its new position (which is guaranteed by the *LUP* algorithm itself). The tree obtained after *ERN* transformation procedure is a proof tree since all facts derived by erased (duplicated) nodes are still derived in a new tree (only this time just once). □

Lemma 6 (Termination of *LUP & ERN*)**.** The procedure of lifting-up and eliminating repeated nodes is terminating.

Proof. The procedure terminates since there is a finite number of nodes in the proof tree. Since the *LUP* procedure moves the nodes appearing below the branching node to the position above the branching node, it can be applied at most the number of times equal to the number of non-branching nodes in the proof tree, which is in worst case equal to $O(size(\Delta))$. In each iteration of the *LUP* procedure, the dependency graph M_d is computed, which has the time complexity $O(size(\Delta)^2)$. Then for each row of the matrix M_d the last value equal to 1 is being sought, and if a branching node was found before that value, the corresponding node would be lifted-up to the position above that branching node – this takes $O(size(\Delta))$ time. After the *LUP* procedure stops, repeated nodes are being eliminated – for each maximal linear subtree of the proof Δ all pairs of steps are being tested for equality and if two steps are found equal – one of them is being erased. This takes $O(size(\Delta)^2)$ time. Altogether *LUP & ERN* procedure takes $O(size(\Delta)^3)$ time in worst case. □

Lemma 7 (Idempotence of *LUP & ERN*)**.** The procedure of lifting-up and eliminating repeated nodes is idempotent, i.e.:

$$LUP\,\&\,ERN(LUP\,\&\,ERN(\Delta)) = LUP\,\&\,ERN(\Delta)$$

Proof. The *LUP & ERN* procedure stops when there are no more nodes that could be lifted-up and when all repeated nodes are erased. Therefore, applying the procedure once again would not change the proof tree, since there would be no candidate nodes neither for lifting-up, nor for elimination of repeated nodes. □

Lemma 8 (Non-increasing size of *LUP & ERN*)**.**

$$size(LUP\,\&\,ERN(\Delta)) \leq size(\Delta).$$

Proof. Lifting-up procedure does not change the size of the proof tree, while the elimination of repeated nodes can decrease its size, so altogether the new size of the proof tree is less or equal to the size of the initial proof tree. □

3.5 Reductio ad Absurdum (RAA)

In CL, negation is not used. However, negation of predicate can be “simulated” by introducing a new symbol $\overline{P}$ which has a role of $\neg P$ [18], supported by the axioms $P \wedge \overline{P} \Rightarrow \bot$ and $P \vee \overline{P}$. Note that these axioms are only specific instances of *tertium non datur* (introduced to model the original, possibly classical-logic initial axiom system), but the obtained deductive system still remains intuitionistic.

The *RAA* transformation procedure consists of converting proof fragments into the reductio ad absurdum form. This transformation is particularly significant for the proofs given in the natural-language form, since proofs in reductio ad absurdum form closely resemble proofs found in mathematical textbooks. This is important, as proofs obtained in the framework of coherent logic can easily be translated into natural-language form, as in [22]. In this way, when the *RAA* transformation is applied, the result is a proof, which has no redundancy and which consists of steps which are typical for traditional proofs.

For convenience, the set of inference rules is expanded by the rules:

$$\dfrac{\begin{array}{c}[P]\\ \vdots\\ \bot\end{array}}{\overline{P}}\ \neg I_1 \qquad\qquad \dfrac{\begin{array}{c}[\overline{P}]\\ \vdots\\ \bot\end{array}}{P}\ \neg I_2$$

where P denotes an atomic formula. These two rules are derived from the axiom $P \vee \overline{P}$ and the rule $(\vee E)$.

We will also consider an extended notion of proof in which, instead of having the property (P2), the following property holds:

(P2') for each node, either the property (P2) holds, or an application of the $(\neg I_1)$ or $(\neg I_2)$ inference rule is assigned to the node – in that case, the set of derived facts of that node contains $\bot$, while the set of derived facts of the child-node is the same as the set of derived facts of the ancestor node, with the assumption fact substituted by its negation, or in the case when the assumption is a negation of a fact by the fact itself.

This type of proof will be called the *extended proof.*

The *RAA* transformation is applicable in cases in which a proof-trace contains an axiom application of the form $P \vee \overline{P}$ to some vector of constants $\vec{a}$, and where one of the two branches is closed by a contradiction. If the first branch is closed by a contradiction, we can, instead of the application of the axiom $P \vee \overline{P}$, assume the assumption of the first branch $P(\vec{a})$, use the same sequence of inference steps used in the first branch, derive a contradiction, and conclude, by the newly introduced rule $\neg I_1$, that $\overline{P}(\vec{a})$ holds. If the second branch is closed by a contradiction, we can assume the assumption of the second branch $\overline{P}(\vec{a})$, use the sequence of inference steps used in the second branch, derive a contradiction, and conclude, by the newly introduced rule $\neg I_2$, that $P(\vec{a})$ holds.

$$\dfrac{\begin{array}{c} G \\ \vdots\; c_1 \\ P(\vec{a})\vee\overline{P}(\vec{a})\end{array} \quad \begin{array}{c}[P(\vec{a})]\\ \vdots\; c_2 \\ \dfrac{\bot}{F}\,eqf\end{array} \quad \begin{array}{c}[\overline{P}(\vec{a})]\\ \vdots\; c_3\\ F\end{array}}{F}\vee E \quad\rightarrow\quad \begin{array}{c} G\\ \vdots\; c_1\\ [P(\vec{a})]\\ \vdots\; c_2\\ \dfrac{\bot}{\overline{P}(\vec{a})}\neg I_1\\ \vdots\; c_3\\ F\end{array} \tag{4}$$

$$\dfrac{\begin{array}{c} G \\ \vdots\; c_4 \\ P(\vec{a})\vee\overline{P}(\vec{a})\end{array} \quad \begin{array}{c}[P(\vec{a})]\\ \vdots\; c_5 \\ F\end{array} \quad \begin{array}{c}[\overline{P}(\vec{a})]\\ \vdots\; c_6\\ \dfrac{\bot}{F}\,eqf\end{array}}{F}\vee E \quad\rightarrow\quad \begin{array}{c} G\\ \vdots\; c_4\\ [\overline{P}(\vec{a})]\\ \vdots\; c_6\\ \dfrac{\bot}{P(\vec{a})}\neg I_2\\ \vdots\; c_5\\ F\end{array} \tag{5}$$

The RAA transformation is shown in (4) and (5).

It should be noticed that, because of the introduction of ($\neg I_1$) and ($\neg I_2$) rules, the tree structure obtained by application of RAA transformation does not satisfy the proof property (P2) any more, but satisfies the property (P2') instead. Therefore, the obtained proof is an extended proof.

3.6 Properties of RAA

In this section, the RAA transformation is proven to be correct, terminating, idempotent and non-increasing in size.

Lemma 9 (Correctness of RAA)**.** If *proof*(ϕ,Δ) then *proof*$(\phi, RAA(\Delta))$.

Proof. This follows straightforwardly from the specification of the transformation. As it has already been mentioned, the obtained proof is an extended proof. $\square$

Lemma 10 (Termination of RAA)**.** The reductio ad absurdum procedure is terminating.

Proof. The number of instances of the axioms of the form $P\vee\overline{P}$ in the proof is finite. Therefore, procedure is terminating. During one traversal of a proof tree, an information whether the node is a candidate for RAA transformation (if all the leaf-nodes in one of two branches are closed by a contradiction) is assigned to all of the branching nodes. After that, it takes one more proof tree traversal to transform a tree into the reductio ad absurdum form. Therefore, the time complexity of the RAA procedure is linear w.r.t. the size of the input proof tree. $\square$

Lemma 11 (Idempotence of RAA)**.** The reductio ad absurdum procedure is idempotent, meaning: $RAA(RAA(\Delta)) = RAA(\Delta)$.

Proof. If we denote $RAA(\Delta)$ with Δ_1, the objective is to prove that $RAA(\Delta_1) = \Delta_1$. The RAA transformation eliminates from the proof all of the instances of the axioms of the form $P \vee \overline{P}$, where one branch is closed by a contradiction. Therefore, Δ_1 does not contain any node which is a candidate for RAA transformation, meaning $RAA(\Delta_1) = \Delta_1$. □

Lemma 12 (Non-increasing size of RAA)**.** $\mathit{size}(RAA(\Delta)) = \mathit{size}(\Delta)$.

Proof. The RAA transformation consists of a simple restructuring of a proof tree. The original tree and the reduced one do have different structures, but still the same number of nodes. So it holds that: $\mathit{size}(RAA(\Delta)) = \mathit{size}(\Delta)$. □

3.7 Properties of PST

Putting together the properties proven for the *EIN*, *LUP & ERN*, and RAA transformations in the previous sections, the PST transformation can be shown correct, terminating and non-increasing in size.

Theorem 1 (Correctness of PST)**.** If $\mathit{proof}(\phi, \Delta)$ then $\mathit{proof}(\phi, PST(\Delta))$.

Proof. The correctness of PST follows directly from the definition of *PST*, Lemma 1, Lemma 5, and Lemma 9. □

Theorem 2 (Termination of PST)**.** Proof simplification transformation is terminating.

Proof. Lemmas 2, 6, and 10 show that EIN, $LUP\,\&\,ERN$, and RAA transformations terminate, so PST, as their composition, terminates as well. □

Theorem 3 (Non-increasing size of PST)**.** $\mathit{size}(PST(\Delta)) \leq \mathit{size}(\Delta)$.

Proof. Using Lemma 4, Lemma 8 and Lemma 12 the following holds:

$$\begin{aligned}\mathit{size}(PST(\Delta)) &= \mathit{size}(RAA(LUP\,\&\,ERN(EIN(\Delta))))\\ &= \mathit{size}(LUP\,\&\,ERN(EIN(\Delta))) \leq \mathit{size}(EIN(\Delta)) \leq \mathit{size}(\Delta)\end{aligned}$$

□

It can be noticed that PST does not have the idempotence property. This is because a proof in reductio ad absurdum form, as the extended proof, does not necessarily satisfy proof property (P2) and, because of that, it is not possible to apply again the EIN transformation to it. However, if redundant nodes would be defined for extended proof analogously as for the basic proofs, it could be easily proven that $PST(\Delta)$ would not contain redundant nodes. Also, new applications of $(\neg I_1)$ and $(\neg I_2)$ rules would not be possible.

4 IMPLEMENTATION AND EXAMPLES

The presented transformation system has been implemented within the ArgoCLP prover [22][6]. It is a generic theorem prover for coherent logic, developed in C++. After the theorem is proven and corresponding proof-trace generated, all of the irrelevant steps are eliminated from the proof-trace using the simplification tool which implements the proposed transformation system. The length of the proof-trace is often significantly shortened. This simplified proof-trace can then be exported to the Isabelle/Isar format, and checked formally for correctness within the Isabelle proof assistant. This is important, as it guarantees that the simplified proof-trace still represents a proof of the same theorem[7].

Proofs are rarely presented in the textbooks in a tree-form (like in natural deduction). To make proofs more alike the proofs from mathematical textbooks, a transformation into the block-structured form is needed[8]. This is done in the following way – whenever an assumption is performed the paragraph is indented by one more tab, and when the branch corresponding to that assumption is closed the indentation is returned to the previous value. In this way fragments of the proof that belong to a subproof are not marked by a line or a rectangle, but logically separated from the rest of the proof. Once a proof-trace is simplified, it is translated in this way to a natural-language form (in English, in LaTeX format).

A prover with this additional simplification tool was tested on theorems from the field of Euclidean geometry within four different axiom systems [22]. In this way 26 proof-traces were generated and afterwards simplified, using our simplification tool. The simplification identified four of them as axioms (since the simplified proof-trace had a length of just one step – which is an application of the axiom they were equivalent to, while the longest of these non-simplified traces was 28 steps long). These four examples were discarded from further analysis, and for the remaining 22 examples, the average percentage of irrelevant steps was calculated. For our corpus, the average percentage of irrelevant steps was 61.8 %, meaning that more than half of the steps in the automatically generated proofs were redundant. Also, the longer the generated proof, the greater its percentual shortening.

Three automatically generated proofs, one from the field of geometry, one from the field of term rewriting theory, and one from the field of metric spaces, given in a natural-language form, are listed below. The first proof was 62 steps long, and simplification process shortened it to 16 steps. The non-simplified version of the second proof has 35 steps, while the simplified version has 21 steps (in the term rewriting example, negations were not allowed as predicates and, therefore, only the

[6] The ArgoCLP prover, together with a proof simplification module, is available on-line from `http://poincare.matf.bg.ac.rs/~vesnap/ArgoCLP_v4.zip`.

[7] However this does not guarantee that the proposed transformation procedure is formally correct – that could be achieved only by verification of the transformation procedure itself within some proof assistant.

[8] The format used is similar to the format used by systems like Isar [15] or Athena [1].

EIN and the *LUP & ERN* transformations were performed). The proof of the third theorem was 141 steps long, and the simplification procedure shortened it to 7 steps.

The following theorem states that if the point A is outside the segment BC, then the point A is outside the segment CB.

Theorem TH_geo:

Assuming that out(A,B,C), show that out(A,C,B).

Proof:

1. There exist a point G such that $\text{bet}(B, A, G)$ and $A \neq G$ (by axiom th_3_14).
2. From the fact that $A \neq G$, it holds that $G \neq A$ (by the equality axioms).
3. It holds that $A = B$ or $A \neq B$ (by axiom ax_g1).
    4. Assume that $A = B$.
        5. From the facts that $\text{out}(A, B, C)$ and $A = B$ it holds that $\text{out}(A, A, C)$.
        6. From the facts that $\text{out}(A, A, C)$, there exist a point I such that $A \neq A$, $C \neq A$, $I \neq A$, $\text{bet}(A, A, I)$ and $\text{bet}(C, A, I)$ (by axiom th_6_3_1).
        7. From the facts that $A \neq A$, and $A = A$ we get a contradiction.
            This proves the conjecture.
    8. Assume that $A \neq B$.
        9. From the fact that $A \neq B$, it holds that $B \neq A$ (by the equality axioms).
            Let us prove that $A \neq C$ by reductio ad absurdum.
            10. Assume that $A = C$.
                11. From the facts that $\text{out}(A, B, C)$ and $A = C$ it holds that $\text{out}(A, B, A)$.
                12. From the facts that $\text{out}(A, B, A)$, there exist a point I such that $B \neq A$, $A \neq A$, $I \neq A$, $\text{bet}(B, A, I)$ and $\text{bet}(A, A, I)$ (by axiom th_6_3_1).
                13. From the facts that $A \neq A$, and $A = A$ we get a contradiction.
                    Contradiction.
            Therefore, it holds that $A \neq C$.
        14. From the fact that $A \neq C$, it holds that $C \neq A$ (by the equality axioms).
        15. From the facts that $B \neq A$, $C \neq A$, $G \neq A$, $\text{bet}(B, A, G)$, and $\text{out}(A, B, C)$, it holds that $\text{bet}(C, A, G)$ (by axiom th_6_2_2).
        16. From the facts that $C \neq A$, $B \neq A$, $G \neq A$, $\text{bet}(C, A, G)$, and $\text{bet}(B, A, G)$, it holds that $\text{out}(A, C, B)$ (by axiom th_6_3_2).
            This proves the conjecture.

Theorem proved in 16 steps and in 6.68 *s.*

The Diamond Property in rewriting theory states that if p rewrites to both q and r, then the latter two rewrite both to some s (all in one step). The next theorem states that if some rewrite relation satisfies the diamond property, then its reflexive closure also satisfies the diamond property[9].

---

[9] This is one of the problems listed in Bezem's collection of problems in coherent logic in `http://www.ii.uib.no/~bezem/GL/`.

Theorem TH_rew:

Assuming that $RE0$ rewrites to or is equal to $RE1$, and $RE0$ rewrites to or is equal to $RE2$, show that there exist a rewterm $RE3$ such that $RE1$ rewrites to or is equal to $RE3$ and $RE2$ rewrites to or is equal to $RE3$.

Proof:

1. It holds that $RE1$ is equal to $RE1$ (by axiom ax_ref_e).
2. It holds that $RE2$ is equal to $RE2$ (by axiom ax_ref_e).
3. From the facts that $RE1$ is equal to $RE1$, it holds that $RE1$ rewrites to or is equal to $RE1$ (by axiom ax_e_in_re).
4. From the facts that $RE2$ is equal to $RE2$, it holds that $RE2$ rewrites to or is equal to $RE2$ (by axiom ax_e_in_re).
5. From the facts that $RE0$ rewrites to or is equal to $RE1$, it holds that $RE0$ is equal to $RE1$ or $RE0$ rewrites to $RE1$ (by axiom ax_e_or_r).
   6. Assume that $RE0$ is equal to $RE1$.
      7. From the facts that $RE0$ is equal to $RE1$, it holds that $RE1$ is equal to $RE0$ (by axiom ax_sym_e).
      8. From the facts that $RE1$ is equal to $RE0$, and $RE0$ rewrites to or is equal to $RE2$, it holds that $RE1$ rewrites to or is equal to $RE2$ (by axiom ax_congl).
         This proves the conjecture.
   9. Assume that $RE0$ rewrites to $RE1$.
   10. From the facts that $RE0$ rewrites to or is equal to $RE2$, it holds that $RE0$ is equal to $RE2$ or $RE0$ rewrites to $RE2$ (by axiom ax_e_or_r).
      11. Assume that $RE0$ is equal to $RE2$.
         12. From the facts that $RE0$ is equal to $RE2$, it holds that $RE2$ is equal to $RE0$ (by axiom ax_sym_e).
         13. From the facts that $RE2$ is equal to $RE0$, and $RE0$ rewrites to or is equal to $RE1$, it holds that $RE2$ rewrites to or is equal to $RE1$ (by axiom ax_congl).
            This proves the conjecture.
      14. Assume that $RE0$ rewrites to $RE2$.
         15. It holds that $RE1 = RE2$ or $RE1 \neq RE2$ (by axiom ax_g1).
            16. Assume that $RE1 = RE2$.
               17. From the facts that $RE1$ rewrites to or is equal to $RE1$ and $RE1 = RE2$ it holds that $RE2$ rewrites to or is equal to $RE1$.
                  This proves the conjecture.
            18. Assume that $RE1 \neq RE2$.
               19. From the facts that $RE0$ rewrites to $RE1$, $RE1 \neq RE2$, and $RE0$ rewrites to $RE2$, there exist a rewterm $RE3$ such that $RE1$ rewrites to $RE3$ and $RE2$ rewrites to $RE3$ (by axiom ax_dp_r).
               20. From the facts that $RE1$ rewrites to $RE3$, it holds that $RE1$ rewrites to or is equal to $RE3$ (by axiom ax_r_in_re).

21. From the facts that $RE2$ rewrites to $RE3$, it holds that $RE2$ rewrites to or is equal to $RE3$ (by axiom ax_r_in_re).

This proves the conjecture.

Theorem proved in 21 steps and in 0.32 *s.*

In the theory of metric spaces it holds that if $SP1$ is a complete metric space and $SP2$ is a closed subset of $SP1$ then the metric space $SP2$ is complete. A metric space $SP2$ is complete if and only if every Cauchy sequence in $SP2$ converges in $SP2$. The following theorem is a variant of this statement, and it states that if metric space $SP1$ is complete and the metric space $SP2$ is closed, $AR0$ is Cauchy sequence in $SP1$ which does not converge in $SP1$, then the metric space $SP1$ is not a subspace of metric space $SP2$[10].

Theorem TH_mp:

Assuming that the metric space $SP0$ is complete, the metric space $SP1$ is closed, the sequence $SE0$ is Cauchy, the sequence $SE0$ belongs to the metric space $SP1$, and the sequence $SE0$ does not converge in the metric space $SP1$, show that the metric space $SP1$ is not a subspace of the metric space $SP0$.

Proof:

Let us prove that the metric space $SP1$ is not a subspace of the metric space $SP0$ by reductio ad absurdum.

1. Assume that the metric space $SP1$ is a subspace of the metric space $SP0$.

2. From the facts that the sequence $SE0$ belongs to the metric space $SP1$, and the metric space $SP1$ is a subspace of the metric space $SP0$, it holds that the sequence $SE0$ belongs to the metric space $SP0$ (by axiom ax_1).

3. From the facts that the sequence $SE0$ is Cauchy, the sequence $SE0$ belongs to the metric space $SP0$, and the metric space $SP0$ is complete, it holds that the sequence $SE0$ converges in the metric space $SP0$ (by axiom ax_2).

4. From the facts that the sequence $SE0$ converges in the metric space $SP0$, there exists an element $EL0$ such that the value $EL0$ belongs to the metric space $SP0$ and the sequence $SE0$ converges to the value $EL0$ (by axiom ax_3).

5. From the facts that the metric space $SP1$ is closed, the metric space $SP1$ is a subspace of the metric space $SP0$, the sequence $SE0$ belongs to the metric space $SP1$, and the sequence $SE0$ converges to the value $EL0$, it holds that the value $EL0$ belongs to the metric space $SP1$ (by axiom ax_4).

6. From the facts that the sequence $SE0$ converges to the value $EL0$, and the value $EL0$ belongs to the metric space $SP1$, it holds that the sequence $SE0$ converges in the metric space $SP1$ (by axiom ax_5).

7. From the facts that the sequence $SE0$ converges in the metric space $SP1$, and the sequence $SE0$ does not converge in the metric space $SP1$ we get a contradiction.

[10] This is one of the examples from the paper [10].

Contradiction.

Therefore, it holds that the metric space $SP1$ is not a subspace of the metric space $SP0$.

This proves the conjecture.

Theorem proved in 7 steps and in 0.65 s.

These sorts of proofs are more comprehensible than the proofs given in tree-form, but in order to make them more readable, numeration of the steps should be thrown out from the proof, the steps should be grouped into a paragraph, while some trivial steps should be erased. Also, a human mathematician would rarely write "by axiom". The proof of TH_mp, transformed by hand into this form, is given below:

Proof:

Assume that the metric space $SP1$ is a subspace of the metric space $SP0$. From the facts that the sequence $SE0$ belongs to the metric space $SP1$, and the metric space $SP1$ is a subspace of the metric space $SP0$, it holds that the sequence $SE0$ belongs to the metric space $SP0$. From the facts that the sequence $SE0$ is Cauchy, the sequence $SE0$ belongs to the metric space $SP0$, and the metric space $SP0$ is complete, it holds that the sequence $SE0$ converges in the metric space $SP0$. From the facts that the sequence $SE0$ converges in the metric space $SP0$, there exists an element $EL0$ such that the value $EL0$ belongs to the metric space $SP0$ and the sequence $SE0$ converges to the value $EL0$. From the facts that the metric space $SP1$ is closed, the metric space $SP1$ is a subspace of the metric space $SP0$, the sequence $SE0$ belongs to the metric space $SP1$, and the sequence $SE0$ converges to the value $EL0$, it holds that the value $EL0$ belongs to the metric space $SP1$. From the facts that the sequence $SE0$ converges to the value $EL0$, and the value $EL0$ belongs to the metric space $SP1$, it holds that the sequence $SE0$ converges in the metric space $SP1$. Contradiction. Therefore, it holds that the metric space $SP1$ is not a subspace of the metric space $SP0$.

However, this is still far from the proofs that could be found in mathematical textbooks. The problem of generating fluent sentences in the natural language from primitive constructs is an extremely hard and challenging task, even for very sophisticated NLG systems. The proofs we obtain are a simple abstractions of this.

5 RELATED WORK

Recently, the problem of proof optimization started to attract a lot of attention across various contexts. Approaches which address this problem focus either on the elimination of redundant parts of proofs by doing pruning, folding-up, proof condensation, common subexpression elimination, etc., or on representing proofs in a more compact fashion, by doing i.e. lemma extraction.

A general technique most similar to our transformation procedure is called pruning. It consists of dropping all of the redundant case distinctions and existential eliminations sub-proofs. Pruning was initially introduced by Prawitz [20], and the original variant is conceptually similar to our elimination of branching nodes. Also, any derivation containing an introduction of a connective followed immediately by its elimination can be turned into an equivalent derivation without this detour. This reduces a proof simplification problem to a normalization problem of λ-terms. However, the underlying logics are different. CL, with its constructive proof system, allows an easier and more direct simplification of its proof objects.

In [6, 7], Chiarabini applied pruning to first order minimal logic. It has been shown that a program extracted from a pruned proof can be more efficient than the program extracted from a non-pruned one [11].

The most common way of representing object-level proofs in mathematical textbooks is using graphical block-structured format that was firstly proposed by Jaśkowski [17]. In Jaśkowski's graphical method, every time an assumption is made a new fragment of the proof is started, which is to be enclosed with a rectangle (denoting a subproof). Fitch popularized this method by making small modification, like drawing only left side of the rectangle around subproofs and underlining the assumption, and nowadays this method is called in the literature "the Fitch method". An algorithm for simplification of Fitch-style natural deduction proofs in first-order logic was developed by Arkoudas [2]. It is used to "clean up" proofs obtained by different ATPs based on Fitch-style natural deduction, as well as resolution provers, by converting them to Fitch-style proofs. That system can take as an input proofs that violate the principle of parsimony. Then certain techniques like elimination of redundancies, repetitions and detours are employed in order to eliminate it and, in return, simplified proofs which obey the parsimony principle are obtained. Their system firstly restructures the proof (it does not necessarily return a subtree of a proof tree), and afterwards implements the contracting transformation (which eliminates intermediate conclusions which are not used further in the proof). In contrast to Arkoudas' general simplification procedure, our proof simplification procedure is specific for CL and, since CL has constructive proof system and proof objects are easily generated, simplification procedure is more direct and easy to implement.

Another proof method which originated from Jaśkowski's method is Lemmon's method [8]. Proof format used here consists of a list of enumerated proof steps, while for each step the ordinals of the statements used in that step and the name of the rule applied are given. This format is similar to the format used for presenting object-level proofs in some of the textbooks. Lemmon employed rules of conditional proof and reductio ad absurdum, where the latter resembles steps used in our proofs.

Koshimura and Hasegawa developed a proof simplification procedure for model generation theorem proving [14]. It implements folding-up along with proof condensation by embedding proof simplification mechanism into a model generation. That approach is similar to ours in the way that unnecessary parts of a proof are identified and eliminated. This is achieved by computing relevant atoms which contribute to closing a subproof. However, our approach is different from theirs in the following manner: first, the logic which their system uses does not involve existential quantifications, meaning that underlying logics are not the same. Second, the definition of relevant atoms and method for elimination of irrelevant subproofs are different – in their approach, the subproof trees are deleted only when all the leaf-nodes of the subtree are labeled by $\bot$. Still, they make use of a set of relevant atoms not only for proof simplification after the proof has been completed, but also during the proof search, as well as for lemma generation.

Most ATPs do not put emphasis on producing proofs that are easy to read and understand. However, there are systems such as Theorema [16] which emphasize the importance of having an aesthetically pleasing presentation of proofs. The Theorema system is designed to provide computer support for all aspects of the mathematical exploration cycle (including proving, solving, and computing), in the frame of one uniform logic. An important feature of the system is a simplification of proofs generated by Theorema. Once a proof has been found by a prover, by inspecting the proof object, one may be able to restructure the proof in such a way that it becomes more concise and easier to understand. The simplification of the generated proof tree is performed in a similar manner as in our program, but differences exist in the way proof objects are generated, which is a consequence of the different underlying logics. Also, Ganesalingam and Gowers [10] promote benefits of having readable proofs and for that purpose they also consider coherent logic, although they do not use the name "coherent logic". They developed automatic problem solver with a human-style output which is very hard to distinguish from solutions that a human would write.

None of the systems listed above does a transformation of the proof into a reductio ad absurdum form.

There are many situations in which one needs to manipulate proofs (to store them or transfer) and where any kind of proof simplification, as well as its compact representation, would be beneficial. A method for proof optimization in the context of proof-carrying code, along with a method for lemma extraction that replaces similar subproofs with instances of more general lemmas, is given by Rahul and Necula [19]. These methods obtain substantial reduction in the size of proofs.

6 CONCLUSIONS AND FURTHER WORK

In this paper, we have presented a method for proof simplification in the framework of coherent logic. It is used to extract "clean" proofs from the proof-traces obtained by ATPs (usually containing a number of redundant steps). The presented proof simplification procedure is executed only post-festum, once a conjecture has already been proven. We also provide a method for transformation of the proof-trace into the reductio ad absurdum form. In this way, proofs that are formally proven correct, with no redundant or repeated steps, given in natural-language form, can be automatically generated. This is of importance for educational purposes, as well as for the formalization of mathematics. The approach is implemented within the ArgoCLP prover – a coherent logic prover.

There are several possible directions for future work. Information about relevant and irrelevant facts can be useful, not only in the context of obtaining "clean" and shorter proofs, but also in guiding the remaining proving process (i.e., in future search branches). In that way, a simplification would take place not just post-festum, but on the run as well.

One of possible extensions could also be the lemma extraction, which is a technique for retrieving fragments of a proof which can be matched. This would significantly contribute to the optimization of proofs (like in [14]). However, it requires different techniques from the ones used here.

We are planning to work on proof simplification for other logical frameworks. We will also try to generalize this approach to some richer and more expressive logical fragments. It would be interesting to draw out the general form of the rules (given in natural deduction form) which the given proof simplification transformation can be applied to. In addition, we will focus on applications – generation of textbook-like proofs (in a natural-language form) which could be used to compose mathematical textbooks which closely resemble the ones used in education.

Acknowledgments

I am grateful to Predrag Janičić, Tanja Lutovac, and to Petar Maksimović for useful feedback. I am also grateful to anonymous referee, whose detailed suggestions and insights have contributed to significant improvements of the paper. This work was partially supported by the Serbian Ministry of Education, Science and Technological Development grant 174021 and by Swiss National Science Foundation grant SCOPES IZ73Z0_127979/1.

REFERENCES

[1] Arkoudas, K.: An Athena Tutorial. 2005, available on: `http://people.csail.mit.edu/kostas/dpls/athena/athenaTutorial.pdf`.

[2] Arkoudas, K.: Simplifying Proofs in Fitch-Style Natural Deduction Systems. Journal of Automated Reasoning, Vol. 34, 2005, No. 3, pp. 239–294.

[3] Arkoudas, K.—Bringsjord, S.: Computers, Justification, and Mathematical Knowledge. Minds and Machines, Vol. 17, 2007, No. 2, pp. 185–202.

[4] Bezem, M.—Hendriks, D.: On the Mechanization of the Proof of Hessenberg's Theorem in Coherent Logic. Journal of Automated Reasoning, Vol. 40, 2008, No. 1, pp. 61–85.

[5] Bezem, M.—Coquand, T.: Automating Coherent Logic. In: Sutcliffe, G., Voronkov, A. (Eds.): 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2005), Lecture Notes in Computer Science, Springer-Verlag, Vol. 3835, 2005, pp. 246–260.

[6] Chiarabini, L.: A New Adaptation of the Pruning Technique for the Extraction of Efficient Programs from Proofs. 2008.

[7] Chiarabini, L.: Program Development by Proof Transformation. Dissertation, LMU München, Faculty of Mathematics, Computer Science and Statistics, 2009.

[8] Coburn, B.—Miller, D.: Two Comments on Lemmon's Beginning Logic. Notre Dame Journal of Formal Logic, Vol. XVIII, 1977, No. 4.

[9] Fisher, J.—Bezem, M.: Skolem Machines and Geometric Logic. In: Jones, C. B., Liu, Z., Woodcock, J. (Eds.): 4th International Colloquium on Theoretical Aspects of Computing (ICTAC 2007), Lecture Notes in Computer Science, Springer-Verlag, Vol. 4711, 2007, pp. 201–215.

[10] Ganesalingam, M.—Gowers, T.: A Fully Automatic Problem Solver with Human-Style Output. CoRR abs/1309.4501, 2013.

[11] Goad, C.: Computational Uses of the Manipulation of Formal Proofs. UMI, Dissertation Services, CA, USA, 1980.

[12] Hilbert, D.: Grundlagen der Geometrie, Leipzig, 1899.

[13] Janičić, P.—Kordić, S.: EUCLID – The Geometry Theorem Prover. FILOMAT, Vol. 9, 1995, No. 3, pp. 723–732.

[14] Koshimura, M.—Hasegawa, R.: Proof Simplification for Model Generation and its Applications. LPAR '00 Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning, Springer-Verlag Berlin, Heidelberg, Vol. 1955, 2000, pp. 96–113.

[15] Nipkow, T.—Paulson, L. C.—Wenzel, M.: Isabelle/HOL – A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, Springer-Verlag, Vol. 2283, 2002.

[16] Piroi, F.: Tools for Using Automated Provers in Mathematical Theory Exploration. Ph.D. thesis, RISC, Johannes Kepler University, Linz, Austria, 2004.

[17] Pelletier F. J.—Hazen, A. P.: Natural Deduction. In: Gabbay, D., Woods, J. (Eds.): Handbook of the History of Logic, Vol. 11, 2012, pp. 341–414.

[18] Polonsky A.: Proofs, Types, and Lambda Calculus. Ph.D. thesis, University of Bergen, 2011.

[19] Rahul, S.—Necula G.: Proof Optimization Using Lemma Extraction. Technical Report UCB/CSD-01-1143, University of California, Berkeley, 2001.

[20] Prawitz, D.: Ideas and Results in Proof Theory. Proceedings of the 2nd Scandinavian Logic Symposium, 1971, pp. 237–309.

[21] Skolem, T.: Logico-Combinatorial Investigations in the Satisfiability or Provability of Mathematical Propositions: A Simplified Proof of a Theorem by L. Löwenheim and Generalizations of the Theorem. In: van Heijenport, J. (Ed.): From Frege to Gödel. A Source Book in Mathematical Logic, 1879–1931, Harvard University Press, Cambridge, MA, 1967, 1920, pp. 252–263.

[22] Stojanović S.—Pavlović V.—Janičić P.: A Coherent Logic Based Geometry Theorem Prover Capable of Producing Formal and Readable Proofs. In: Schreck, P., Narboux, J., Richter-Gebert, J. (Eds.): Automated Deduction in Geometry, ADG 2010, Lecture Notes in Artificial Inteligence, Springer, Heidelberg, Vol. 6877, 2011, pp. 200–219.

[23] Sutcliffe G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning, Vol. 43, 2009, No. 4, pp. 337–362.

[24] The Coq development team: The Coq Proof Assistant Reference Manual, Version 8.2. TypiCal Project, 2009.

[25] Thiele R.—Wos L.: Hilbert's Twenty-Fourth Problem. Journal of Automated Reasoning, Vol. 29, 2002, pp. 67–89.

[26] Trybulec, A.: Mizar, The Seventeen Provers of the World. Lecture Notes in Computer Science, Springer, Vol. 3600, 2006, pp. 20–23.

Vesna Marinković is a Ph.D. student at the Faculty of Mathematics, University of Belgrade, doing her Ph.D. thesis on automated solving geometric construction problems. She received her B.Sc. in 2006 and since then she has been working at the Faculty of Mathematics as a teaching assistant. Her research interests include automated theorem proving, automated solving geometric construction problems and computational logic.