# Theorem Proving as Constraint Solving with Coherent Logic

Predrag Janičić, Julien Narboux

**Noname manuscript No.**
(will be inserted by the editor)

# Theorem Proving as Constraint Solving with Coherent Logic

**Predrag Janičić · Julien Narboux**

Received: date / Accepted: date

**Abstract** In contrast to common automated theorem proving approaches, in which the search space is a set of some formulae and what is sought is again a (goal) formula, we propose an approach based on searching for a proof (of a given length) as a whole. Namely, a proof of a formula in a fixed logical setting can be encoded as a sequence of natural numbers meeting some conditions and a suitable constraint solver can find such sequence. The sequence can then be decoded giving a proof in the original theory language. This approach leads to several unique features, for instance, it can provide shortest proofs. In this paper, we focus on proofs in coherent logic, an expressive fragment of first-order logic, and on SAT and SMT solvers for solving sets of constraints, but the approach could be tried in other contexts as well. We implemented the proposed method and we present its features, perspectives and performances. One of the features of the implemented prover is that it can generate human understandable proofs in natural language and also machine verifiable proofs for the interactive prover Coq.

**Keywords** Automated theorem proving · interactive theorem proving · constraint solving · coherent logic · SAT/SMT solving

## 1 Introduction

The central task of automated theorem proving – proving a conjecture given some premises – can be pursued in a range of ways, all of which are some form of search. In almost all of them, the search space is a set of some formulae and what is sought

Predrag Janičić
Department for Computer Science, Faculty of Mathematics, University of Belgrade, Studentski trg 16, 11000 Belgrade, Serbia
E-mail: janicic@matf.bg.ac.rs
ORCID: 0000-0001-8922-4948

Julien Narboux
UMR 7357 CNRS, University of Strasbourg, Pôle API, Bd Sébastien Brant, BP 10413, 67412 Illkirch, France
E-mail: narboux@unistra.fr
ORCID: 0000-0003-3527-7184

is again a formula (often a contradiction). There is no assumption on the size of the proof, and the size of the proof is known only after the target formula has been found (if it has been found).

We propose a completely different approach: instead of searching for some formula, we propose searching for a proof as a whole. A (finite, fixed length) proof can be encoded, represented as a sequence of natural numbers or Boolean constants meeting some constraints. For given premises and a given conjecture, that representation yields a set of concrete constraints and solving them gives a needed proof. Hence, this approach can be called "theorem proving as constraint solving". The approach has some unique features, including yielding *short* and *shortest* possible proofs (in terms of the given axioms, without additional lemmas). It is often the case that we know there is some relatively short proof (or we may be interested in such proofs only), for instance, in development of mathematical theories. Then, we can focus on searching for a short proof, while the shortest proof can be especially appreciated. Another unique feature are *hints*: it can be easily required that some specific inference step or some specific axiom *must* be used within the proof (all other proving approaches, when they are provided with some axioms, *may* use them, but are not obliged to). Hints may be useful, for example, in the following context: given an informal proof of some theorem or fragments of a formal proof in some specific representation, we can use that information as *hints* for helping our system in reconstructing the concrete proof under consideration.

The approach could be tried for various logics and calculi – if one needs an automated theorem prover for some theory, he/she needs to specify only the notion of proof (and the way the proofs are mapped into sequences of natural numbers). We concentrate on coherent logic (CL), an expressive fragment of first-order logic. The absence of function symbols in CL allows a trivial encoding of matching of axiom arguments. CL proofs are easily transformed into formal proofs of interactive theorem provers. In addition, CL can serve well as a vehicle for readable proofs, its proofs are simple and natural, which is achieved by combining several natural deduction rules into one rule. CL proofs can also suitably serve as a basic for further post-processing so they can be more human-style.

We implemented the proposed approach and here we present that implementation too. We translate the proof constraints into propositional logic or into some decidable theories of first-order logic (such as linear arithmetic), and solve them using SAT and SMT solvers. From the obtained models, we generate proofs that are machine verifiable or that are given in a natural language form. The approach and the system we present can be subject to different improvements and optimisations. Hence, we don't see this paper as a final, but rather as a first step in this new direction of research.

*Overview of the paper:* In Section 2 we give some background information on coherent logic and on a constraint solver URSA. In Section 3 we present key ideas and techniques within the proposed approach, primarily how is a proof represented as a sequence of natural numbers. In Section 4 we present our prototype implementation. In Section 5 we present some evaluation results and some analysis of performance of our implementation. In Section 6 we discuss some features of the proposed approach and its implementation and some scenarios of usage. In Section 7 we briefly discuss related work. In Section 8 we discuss some possible directions for future work and in Section 9 we draw some final conclusions.

## 2 Background

We assume the reader is familiar with basic notions of automated theorem proving (ATP) and interactive theorem proving (ITP). In particular, we assume some basic knowledge of SAT and SMT solvers [9], of the interactive provers Coq [68] and Isabelle [54], and their interconnections [20].

*Coherent Logic.* A formula of first-order logic is said to be *coherent* if it has the following form:

$$A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee B_{m-1}(\vec{x},\vec{y}))$$

where universal closure is assumed, and where $0 \leq n$, $0 \leq m$, $\vec{x}$ denotes a sequence of variables $x_0, x_2, \ldots, x_{k-1}$ $(0 \leq k)$, $A_i$ (for $0 \leq i \leq n-1$) denotes an atomic formula (involving zero or more variables from $\vec{x}$), $\vec{y}$ denotes a sequence of variables $y_0, y_2, \ldots, y_{l-1}$ $(0 \leq l)$, and $B_j$ (for $0 \leq j \leq m-1$) denotes a conjunction of atomic formulae (involving zero or more of the variables from $\vec{x}$ and $\vec{y}$). If $n = 0$, then the left hand side of the implication is assumed to be $\top$ . If $m = 0$, then the right hand side of the implication is assumed to be $\bot$ . There are no function symbols with arity greater than zero. Coherent formulae do not involve the negation connectives. A coherent theory is a set of sentences, axiomatised by coherent formulae, and closed under derivability.[1]

A number of theories and theorems can be formulated directly and simply in coherent logic (CL). Several authors independently point to CL (or rules similar to those of CL) as suitable for expressing (sometimes – also automating) significant portions of mathematics [4, 29]. In contrast to resolution-based theorem proving, in forward reasoning for CL, the conjecture being proved is kept unchanged and proved without using refutation, Skolemization and clausal form. Thanks to this, CL is suitable for producing readable synthetic proofs [7] with "structure of ordinary mathematical arguments better retained" [26] and also for producing machine verifiable proofs.

Every first-order theory has a coherent conservative extension [26, 61], i.e., any first-order theory can be translated into CL, possibly with additional predicate symbols. This translation process is called "coherentisation" or, sometimes, "geometrisation" [26]. Translation of FOL formulae into CL involves elimination of the negation connectives: negations can be kept in place and new predicates symbols for corresponding sub-formula have to be introduced, or negations can be pushed down to atomic formulae [61]. In the latter case, for every predicate symbol $R$ (that appears in negated form), a new symbol $\overline{R}$ is introduced that stands for $\neg R$, and the following axioms are introduced: $\forall \vec{x}(R(\vec{x}) \wedge \overline{R}(\vec{x}) \Rightarrow \bot)$, $\forall \vec{x}(R(\vec{x}) \vee \overline{R}(\vec{x}))$. In order to enable more efficient proving, some advanced translation techniques are used. Elimination of function symbols, sometimes called *anti Skolemization*, is also done by introducing additional predicate symbols [56].

[1] A coherent formula is also known as a "special coherent implication", "geometric formula", "basic geometric sequent" [26]. A coherent theory is sometimes called a "geometric theory" [47]. However, much more widely used notion of "geometric formula" allows *infinitary disjuctions* (but only over finitely many variables) [72]. Coherent formulae involve only finitary disjunctions, so coherent logic can be seen as a special case of geometric logic, or as a first-order fragment of geometric logic.

If a CL formula can be classically proved from a set of CL formulae, then it can also be intuitionistically proved from that set (this statement is known as the first-order Barr's Theorem [26]). However, translation from FOL to CL is not necessarily constructive.

The problem of provability in CL is semi-decidable. CL admits a simple proof system, a sequent-based variant is as follows [65]:

$$\frac{\Gamma, ax, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}), \underline{B_0(\vec{a},\vec{b}) \vee \ldots \vee B_{m-1}(\vec{a},\vec{b})} \vdash P}{\Gamma, ax, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash P}\ \text{MP}$$

$$\frac{\Gamma, \underline{B_0(\vec{c})} \vdash P \quad \ldots \quad \Gamma, \underline{B_{m-1}(\vec{c})} \vdash P}{\Gamma, B_0(\vec{c}) \vee \ldots \vee B_{m-1}(\vec{c}) \vdash P}\ \text{QEDcs (case split)}$$

$$\frac{}{\Gamma, \underline{B_i(\vec{a},\vec{b})} \vdash \exists \vec{y}(B_0(\vec{a},\vec{y}) \vee \ldots \vee\ B_{m-1}(\vec{a},\vec{y}))}\ \text{QEDas (assumption)}$$

$$\frac{}{\Gamma, \bot \vdash P}\ \text{QEDefq (ex falso quodlibet)}$$

In the rules given above, it is assumed: $ax$ is a formula $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee\ B_{m-1}(\vec{x},\vec{y}))$;[2] $\vec{a}$, $\vec{b}$, $\vec{c}$ denote sequences of constants (possibly of length zero); in the rule MP (*extended modus ponens*), $\vec{b}$ are fresh constants; $\vec{x}$ and $\vec{y}$ denote sequences of variables (possibly of length zero); $A_i(\vec{x})$ ($B_i(\vec{x},\vec{y})$) have no free variables other than from $\vec{x}$ (and $\vec{y}$); $A_i(\vec{a})$ are ground atomic formulae; $B_i(\vec{a},\vec{b})$ and $B_i(\vec{c})$ are conjunctions of ground atomic formulae; $\underline{\Phi}$ denotes the list of conjuncts in $\Phi$ if $\Phi$ is conjunction, and otherwise $\Phi$ itself. In the proving process, the rules are read from bottom to top, i.e., by a rule application one gets the contents (new subgoals) above the line.[3]

For a set of coherent axioms $AX$ and the statement $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee\ B_{m-1}(\vec{x},\vec{y}))$ to be proved, within the above proof system one has to derive the following sequent (where $\vec{a}$ denotes a sequence of new symbols of constants): $AX, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash \exists \vec{y}(B_0(\vec{a},\vec{y}) \vee \ldots \vee\ B_{m-1}(\vec{a},\vec{y}))$.

Notice that in the above proof system case split may occur only at the end of a (sub)proof. However, it is not a substantial restriction: any proof with unrestricted use of case split can be transformed to a proof in the above system.

*Example 1* Consider the following set of axioms:
ax1: $\forall x\ (p(x) \Rightarrow r(x) \vee q(x))$
ax2: $\forall x\ (q(x) \Rightarrow \bot)$
and the following conjecture that can be proved as a CL theorem:

[2] Notice the hidden link between the formulae $B_i(\vec{a},\vec{b})$ from the rule MP and the formula $ax$: the formulae $B_i(\vec{a},\vec{b})$ from the rule are instances of the formulae $B_i(\vec{x},\vec{y})$ from $ax$.

[3] The rule ¬Intro:

$$\frac{\Gamma, R(\vec{x}) \vdash \bot}{\Gamma \vdash \overline{R}(\vec{x})}\ \neg\text{Intro}$$

could be added to the given set of inference rules. In the intuitionistic setting, it can be more suitable (i.e. weaker) than the axiom $\forall \vec{x}(R(\vec{x}) \vee \overline{R}(\vec{x}))$ (for concrete $R$).

$\forall x\ (p(x) \Rightarrow r(x))$

Using the above rule system, the theorem can be proved as follows:

$$\dfrac{\dfrac{\dfrac{}{ax1, ax2, p(a), r(a) \vdash r(a)}\ \text{QEDas} \quad \dfrac{\dfrac{}{ax1, ax2, p(a), q(a), \bot \vdash r(a)}\ \text{QEDefq}}{ax1, ax2, p(a), q(a) \vdash r(a)}\ \text{MP(ax2)}}{ax1, ax2, p(a), r(a) \vee q(a) \vdash r(a)}\ \text{QEDcs}}{ax1, ax2, p(a) \vdash r(a)}\ \text{MP(ax1)}$$

The same proof can be given in a forward manner, in a natural language form (this proof was generated by our prover, described in the rest of the paper):

Consider an arbitrary $a$ such that: $p(a)$. It should be proved that $r(a)$.

1. $r(a) \vee q(a)$ (by MP, from $p(a)$ using axiom ax1; instantiation: $X \mapsto a$)
   2. Case $r(a)$:
      3. Proved by assumption! (by QEDas)
   4. Case $q(a)$:
      5. $\bot$ (by MP, from $q(a)$ using axiom ax2; instantiation: $X \mapsto a$)
      6. Contradiction! (by QEDefq)
7. Proved by case split! (by QEDcs, by $r(a), q(a)$)

The above, forward directed form will be used for modelling proofs in the rest of the paper.

*Constraint programming systems URSA.* The constraint solving system URSA[4] [36] is based on reduction to the propositional satisfiability problem (SAT). In URSA, the problem is specified in a language which is imperative and similar to C, but at the same time, is declarative, as the user does not have to provide a solving mechanism for the given problem. URSA allows two types of variables: natural numbers (with names beginning with `n`, e.g., `nX`) and Booleans (with names beginning with `b`, e.g., `bX`), with a wide range of C-like operators (arithmetic, relational, logical, and bitwise). Variables can have concrete (ground) or symbolic values (in which case, they are represented by vectors of propositional formulae). There is support for procedures and there are control-flow structures (in the style of C). Loops must be with known bounds and there is no `if-else` statement, but only `ite` expressions (corresponding to `?:` in C). An URSA specification is symbolically executed and the given constraint corresponds to a propositional formula. It is then transformed into CNF and passed to one of the underlying SAT solvers. If this formula is satisfiable, the system can return all its models.

[4] `https://github.com/janicicpredrag/URSA`

## 3 Proof Encoding for Coherent Logic and Proof Search

In this section we discuss how a proof in a fragment of CL can be represented as a sequence of natural numbers. The basic idea of the encoding is to encode individual proof steps as fixed-size sequences of numbers with appropriate conditions, and to add conditions constraining the global proof structure. Individual proof steps are represented, among others, by sequences of numbers that correspond to facts derived in that step, corresponding inference rules and, possibly, axioms along with instantiations used. We skip some details, unsubstantial ideas or tricks in order to have the central ideas clearer. A number of aspects of this representation (i.e., of the proof encoding), can be defined in some other way, but we don't discuss all alternatives. Also, there are some redundancies in the encoding, introduced for more convenient representation and proof decoding.

The proposed approach can be applied to the full CL, but it is simpler and more convenient to apply it to its fragment that we call CL2. This restriction is not substantial. Indeed, for proving any formula in CL it is sufficient to prove a corresponding formula in CL2. CL formulae are of the form (universally closed):

$$A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee \ B_{m-1}(\vec{x},\vec{y}))$$

while CL2 formulae meet the following restrictions:

- $m = 1$ or $m = 2$;
- each formula $B_i$ consists of only one conjunct.

### 3.1 From CL to CL2 and Back

Transformation from CL to CL2 is based on the following steps (universal closure is assumed for all formulae):[5]

- An axiom
$$P \Rightarrow \exists \vec{y}(Q \vee (C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}) \vee R),$$
if $k > 1$, is replaced by the following axiom:
$$P \Rightarrow \exists \vec{y}(Q \vee C_{0,1,\ldots,k-1} \vee R),$$
using a shorthand with a new predicate symbol:
$$C_{0,1,\ldots,k-1} \quad \equiv \quad C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$$
($C_{0,1,\ldots,k-1}$ involves all variables that occur in $C_0$, $C_1$, ..., $C_{k-1}$), and requiring additional axioms:
$$\begin{array}{lcl} C_{0,1,\ldots,k-1} & \Rightarrow & C_0 \\ C_{0,1,\ldots,k-1} & \Rightarrow & C_1 \\ \ldots & & \\ C_{0,1,\ldots,k-1} & \Rightarrow & C_{k-1} \end{array}$$

[5] Transformation to CL2 differs for axioms and for conjectures, because they have different polarity in the proving context: *axioms* $\vdash$ *conjecture*.

– An axiom

$$P \Rightarrow \exists \vec{y}(Q_0 \vee Q_1 \vee \ldots \vee Q_{k-1}),$$

if $k > 2$, and where all $Q_i$ are atomic formulae, is replaced by the following axiom:[6]

$$P \Rightarrow \exists \vec{y}(Q_{0,1,\ldots,k-2} \vee Q_{k-1})$$

using shorthands with new predicate symbols:

$$\begin{array}{lcl} Q_{0,1} & \equiv & Q_0 \vee Q_1 \\ Q_{0,1,2} & \equiv & Q_{0,1} \vee Q_2 \\ \ldots & & \\ Q_{0,1,\ldots,k-2} & \equiv & Q_{0,1,\ldots,k-3} \vee Q_{k-2} \end{array}$$

($Q_{0,1,\ldots,i}$ involves all variables that occur in $Q_{1,2,\ldots,i-1}$ and $Q_i$), and requiring additional axioms:

$$\begin{array}{lcl} Q_{0,1} & \Rightarrow & Q_0 \vee Q_1 \\ Q_{0,1,2} & \Rightarrow & Q_{0,1} \vee Q_2 \\ \ldots & & \\ Q_{0,1,\ldots,k-2} & \Rightarrow & Q_{0,1,\ldots,k-3} \vee Q_{k-2} \end{array}$$

– If the conjecture is of the form

$$P \Rightarrow C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1},$$

then it is replaced by a sequence of conjectures

$$P \Rightarrow C_i, \quad \text{for } 0 \leq i \leq k-1.$$

– If the conjecture is of the form:

$$P \Rightarrow \exists \vec{y}(Q \vee (C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}) \vee R),$$

if $k > 1$, it is replaced by the following conjecture:

$$P \Rightarrow \exists \vec{y}(Q \vee C_{0,1,\ldots,k-1} \vee R)$$

using a shorthand with a new predicate symbol:

$$C_{0,1,\ldots,k-1} \quad \equiv \quad C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$$

($C_{0,1,\ldots,k-1}$ involves all variables that occur in $C_0$, $C_1$, …, $C_{k-1}$), and requiring additional axioms:

$$C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1} \quad \Rightarrow \quad C_{0,1,\ldots,k-1}$$

[6] Transformation based on binary cuts of sets of formula would lead to a smaller set of new short-hands and additional axioms – of size $O(\log k)$ instead of $O(k)$. However, the size of disjunctions in real-world examples is usually not too large, so we keep the simple version and will experiment with the second one in the future.

– If the conjecture is of the form

$$P \Rightarrow \exists \vec{y}(Q_0 \vee Q_1 \vee \ldots \vee Q_{k-1}),$$

if $k > 2$, and where all $Q_i$ are atomic formulae, it is replaced by the following conjecture:

$$P \Rightarrow \exists \vec{y}(Q_{0,1,\ldots,k-1})$$

using shorthands with new predicate symbols:

$$Q_{0,1,\ldots,k-1} \quad \equiv \quad Q_0 \vee Q_1 \vee \ldots \vee Q_{k-1}$$

($Q_{0,1,\ldots,k-1}$ involves all variables that occur in $Q_0$, $Q_1$, …, $Q_{k-1}$), and requiring additional axioms:

$$\begin{array}{lcl} Q_0 & \Rightarrow & Q_{0,1,\ldots,k-1} \\ Q_1 & \Rightarrow & Q_{0,1,\ldots,k-1} \\ \ldots & & \\ Q_{k-1} & \Rightarrow & Q_{0,1,\ldots,k-1} \end{array}$$

The above rules are applied in iterations until all axioms and the conjecture are in CL2 form. It is obvious that this process terminates.

*Example 2* The axiom:
$\forall x\ (p(x) \Rightarrow (q(x) \vee (r(x) \wedge s(x)) \vee t(x)))$
is replaced by the following axioms:
$\forall x\ (p(x) \Rightarrow (q\_or\_r\_and\_s(x) \vee t(x)))$
$\forall x\ (q\_or\_r\_and\_s(x)(x) \Rightarrow (q(x) \vee r\_and\_s(x))$
$\forall x\ (r\_and\_s(x) \Rightarrow r(x))$
$\forall x\ (r\_and\_s(x) \Rightarrow s(x))$

The conjecture:
$\forall x \exists y\ (a(x) \Rightarrow (b(x,y) \vee (c(y) \wedge d(y)) \vee e(y))))$
is replaced by the conjecture:
$\forall x \exists y\ (a(x) \Rightarrow (b\_or\_c\_and\_d\_or\_e(x,y)))$
and the following additional axioms:
$\forall x\ (c(x) \wedge d(x) \Rightarrow c\_and\_d(x))$
$\forall x,y\ (b(x,y) \Rightarrow b\_or\_c\_and\_d\_or\_e(x,y))$
$\forall x,y\ (c\_and\_d(y) \Rightarrow b\_or\_c\_and\_d\_or\_e(x,y))$
$\forall x,y\ (e(y) \Rightarrow b\_or\_c\_and\_d\_or\_e(x,y))$

A similar transformation could be used to reduce the number of premises in each axiom to at most two (but we do not use it).

A theory $\mathcal{T}'$ obtained this way from a CL theory $\mathcal{T}$ is its conservative extension. Given a proof in CL2, it is easy to obtain a corresponding proof in CL: all introduced predicate symbols have to be eliminated, i.e., defined expressions have to be replaced by their corresponding definitions in terms of the original signature. This means, however, that not all assumption steps (in the final proof) will hold only facts, but possibly also disjunctions or conjunctions.

As said in Section 2, in translation from FOL to CL, for every predicate symbol $R$ that appears in negated form, a new symbol $\overline{R}$ is introduced that stands for $\neg R$. As a final step in constructing a proof, we can translate $\overline{R}$ back to $\neg R$ (keeping the axioms $\forall \vec{x}(R(\vec{x}) \wedge \neg R(\vec{x}) \Rightarrow \bot)$, and $\forall \vec{x}(R(\vec{x}) \vee \neg R(\vec{x}))$).

### 3.2 Intro

As mentioned in Section 2, given the conjecture (the atom $B_1(\vec{x},\vec{y})$ may be absent): $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists\vec{y}(B_0(\vec{x},\vec{y}) \vee B_1(\vec{x},\vec{y}))$, new constants $\vec{a}$ are introduced and the proving task becomes the task of checking whether it holds that:

$$A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash \exists\vec{y}(B_0(\vec{a},\vec{y}) \vee B_1(\vec{a},\vec{y}))$$

### 3.3 Kinds of Proof Steps

Following the definitions of CL and CL2, a class of CL2 proofs can be represented in the following simple way (there are zero or more assumptions – $As$, MP is used zero or more times, QEDcs involves two other objects Proof):

$$\text{Proof} \ ::= \ \text{As}^* \ \text{MP}^* \ \left(\text{QEDcs}\left(\text{Proof}^2\right) \mid \text{QEDas} \mid \text{QEDefq}\right)$$

Therefore, any CL2 proof can be represented as a sequence of steps and each step is one of the following:

Assumption : This step contains a fact $A(\vec{a})$, introduced by the conjecture formulation or by cases in proofs by cases.

MP : An axiom (universally quantified) $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists\vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee B_{m-1}(\vec{x},\vec{y}))$ has been applied. It has been instantiated using an instantiation $\sigma = [\vec{x} \mapsto \vec{a}, \vec{y} \mapsto \vec{b}]$. Constants $\vec{b}$ are new, fresh constants (not present in the original signature), and all the facts $A_0(\vec{a})$, ... $A_{n-1}(\vec{a})$ have been proved previously, on the current proof branch.

FirstCase : There is a preceding MP step with a conclusion $B_0(\vec{a},\vec{b}) \vee B_1(\vec{a},\vec{b})$. This step opens the first case consideration (i.e., the first subproof) and it introduces the assumption $B_0(\vec{a},\vec{b})$. Without loss of generality, we impose the restriction that that MP step is the immediate previous step. It can be easily proved that this does not harm completeness.

SecondCase : There is a preceding MP step with a conclusion $B_0(\vec{a},\vec{b}) \vee B_1(\vec{a},\vec{b})$, and a corresponding FirstCase completed in the previous proof step. This step opens the second case consideration (i.e., the second subproof) and it introduces the assumption $B_1(\vec{a},\vec{b})$.

QEDbyCases : There is a preceding MP step with a conclusion $B_0(\vec{a},\vec{b}) \vee B_1(\vec{a},\vec{b})$ and both cases have been completed (in the previous step). This step closes the proof by cases.

QEDbyAssumption : The goal statement is present in the assumptions or is derived in the previous step.

QEDbyEFQ : If $\bot$ has been derived in the previous step, then the goal statement has been proved as well.

### 3.4 Axioms and Conjecture Representation

We assume that all the given axioms are in the fragment CL2. The axioms are ordered and each is assigned an ordinal number. The same holds for the predicate symbols and for the symbols of constants (from the original signature and those

introduced later, as fresh constants). (The symbols $\bot$ and $\top$ as predicate symbols of arity 0 may be assigned values 0 and 1).

An axiom $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee B_1(\vec{x},\vec{y}))$ (universal closure is assumed) is represented by the following pieces of information:

- the ordinal number;
- the number of universal quantifiers;
- the number of existential quantifiers;
- the number of premises (i.e., the number $n$);
- a flag, true if the axiom is branching (i.e., if there is an atom $B_1$), false, otherwise;
- the ordinal numbers of predicates symbols occurring in the axiom (respectively in the premises and the goal), i.e., the original numbers of dominant symbols in $A_0, \ldots, A_{n-1}, B_0, B_1$;
- a flag saying whether a certain argument in an atomic formulae is a constant symbol, and if yes, the ordinal number of that constant symbol;
- the mapping between variables within the atomic formulae and the quantifiers.

All of the above values can be represented by natural numbers. We use zero-based counting (like in C/C++), for a more compact representation.

*Example 3* Consider the axiom ax1: $\forall x\ (p(x) \Rightarrow r(x) \vee q(x))$ from Example 1. Let us assume that the predicate symbols and their negated versions $p$, $\neg p$, $r$, $\neg r$, $q$, $\neg q$ are represented by the numbers $2, 3, 4, 5, 6, 7$, and the constant $a$ is represented by 0. Then, the axiom is represented by the following values:

- the ordinal number is 1;
- the number of universal quantifiers is 1;
- the number of existential quantifiers is 0;
- the number of premises is 1;
- the axiom is branching;
- the ordinal numbers of predicates symbols are 2, 4, 6;
- no argument in the atomic formula is a constant;
- each variable appearing in the atomic formula maps to the first quantified variable.

### 3.5 Proof Step Representation

The proof is represented as an ordered list of proof steps. The structure of the proof is encoded using some nesting information attached to each proof step. Encoding of proof steps is adjusted to a concrete conjecture to be proved: a maximal number of premises in the axioms is known, and a maximal arity of predicate symbols is known, so fixed-size slots are used for encoding each proof step. Each proof step is represented by the following values (not all are necessarily relevant for each step):

`StepKind` : one of the possible step kinds (see the list in Section 3.3);
`Contents` : one or two atoms, represented by their predicate symbols and argument constants: $P_0(a_0, ..., a_{k-1})$ and, possibly, $P_1(a_0, ..., a_{l-1})$.

`Nesting` : a value that characterises the position of the step in the proof considered as a tree. The initial value is 1. Each MP step has the same nesting as its previous step. In a proof by cases, if the cases are introduced by a MP step with nesting $n$, then the FirstCase step has nesting $2n$, the corresponding SecondCase step has nesting $2n+1$, and the corresponding QEDbyCases has nesting $n$.

`Goal` : says whether the contents of the step $s$ matches the goal (this is redundant information, but convenient for expressing conditions for some proof steps). In one proof, there may be several steps $s$ for which `Goal` $(s)$ is true.

The following pieces of information are relevant for MP steps only:

`AxiomApplied` : the ordinal number of the axiom used;

`From` values: the ordinal numbers of proof steps justifying premises of the axiom applied;

`Instantiation` : the instantiation used (for instance, $\sigma = [\vec{x} \mapsto \vec{a}, \vec{y} \mapsto \vec{b}]$). It is stored as a sequence of ordinal numbers of constants to which the quantified variables are mapped respectively. For axioms that introduce new constants, they map to new, unused natural numbers.

`Cases` : says whether the used axiom is branching or not (this is redundant information, but convenient);

`Premises` : says how many premises the axioms used has (this is redundant information, but convenient).

All of the above values can be represented by natural numbers (again, zero-based counting like in C/C++ is used).

*Example 4* Consider again the CL proof shown in Example 1:

Consider an arbitrary $a$ such that: $p(a)$. It should be proved that $r(a)$.

1. $r(a) \vee q(a)$ (by MP, from $p(a)$ using axiom ax1; instantiation: $X \mapsto a$)
   2. Case $r(a)$:
      3. Proved by assumption! (by QEDas)
   4. Case $q(a)$:
      5. $\bot$ (by MP, from $q(a)$ using axiom ax2; instantiation: $X \mapsto a$)
      6. Contradiction! (by QEDefq)
7. Proved by case split! (by QEDcs, by $r(a), q(a)$)

Its proof steps can be represented by the following numerical values, given that predicate symbols and their negated versions $p$, $\neg p$, $r$, $\neg r$, $q$, $\neg q$ are represented by the numbers $2, 3, 4, 5, 6, 7$, the constant $a$ is represented by 0, and the information on `StepKind` and `AxiomApplied` is packed in one number, as explained in Section 4.2: [7]

[7] The given content, i.e., a proof represented by numbers, was created automatically by our prover. The shown proof representation does not include all variables that were considered during the proving/solving process. Also, some of the given values are redundant (but are still kept for convenience in this representation).

```
0.  1  0 0    2 0     /*** Nesting: 1; Step kind:0 = Assumption;
                           Branching: no; p2(a) ***/
1.  1 13 1    4 0 6 0 /*** Nesting: 1; Step kind:13 = MP-axiom:13;
                           Branching: yes; p4(a) or p6(a) ***/
                    0 /*** From steps: (0) ***/
                    0 /*** Instantiation ***/
2.  2  2 0    4 0     /*** Nesting: 2; Step kind:2 = First case;
                           Branching: no; p4(a) ***/
3.  2 10              /*** Nesting: 2; Step kind:10 = QED by assumption; ***/
4.  3  3 0    6 0     /*** Nesting: 3; Step kind:3 = Second case;
                           Branching: no; p6(a) ***/
5.  3 14 0    0       /*** Nesting: 3; Step kind:14=MP-axiom:14);
                           Branching: no; p0() ***/
                    4 /*** From steps: (4) ***/
                    0 /*** Instantiation ***/
6.  3 11              /*** Nesting: 3; Step kind:11 = QED by EFQ; ***/
7.  1  9              /*** Nesting: 1; Step kind:9 = QED by cases; ***/
```

### 3.6 Goal Representation

Let us suppose that the task is to show (the atom $B_1$ may be absent):

$$A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash \exists \vec{y}(B_0(\vec{a}, \vec{y}) \vee B_1(\vec{a}, \vec{y})).$$

Then the goal, the formula $\exists \vec{y}(B_0(\vec{a}, \vec{y}) \vee B_1(\vec{a}, \vec{y})$, is used to set constraints on the last possible proof step, i.e., on the $MaxL - 1$-th step (recall that zero-based counting is used), where $MaxL$ is the maximal proof length (a concrete number) that has to be provided. The goal is described similarly as other proof steps:

- its `StepKind` is one of QEDbyAssumption , QEDbyCases , QEDbyEFQ ;
- `Contents` $(MaxL - 1)(0) = B_0(\vec{a}, \vec{y})$ (and possibly `Contents` $(MaxL - 1)(1) = B_1(\vec{a}, \vec{y})$, if $B_1$ is present); the variables $\vec{y}$ as arguments have a special treatment;
- `Nesting` $(MaxL - 1)$ equals 1;
- `Cases` $(MaxL - 1)$ is true iff there is the atom $B_1$, i.e., there is a disjunction in the goal).

A proof to be found may have a length $L$, less or equal $MaxL$, as ensured by constraints presented in Section 3.8.

### 3.7 Encoding of Proof Steps – Local Constraints

Let us express conditions that have to be met for a proof step of a particular kind. Let the length of the proof be $L$. It must hold that $L \leq MaxL$, and for each proof step $s$ $(0 \leq s < L)$ one of the following groups of conditions holds (i.e., each step has one of the listed step kinds). Notice that $L$ is a symbolic, not a concrete value, so a check if $0 \leq s < L$ is used as a constraint.

Assumption : Given the initial assumptions $A_0(\vec{a})$, ..., $A_{n-1}(\vec{a})$, they constitute the first $n$ steps of the proof (they do not introduce any unknown), for $s < n$:

1. `StepKind` $(s)$ = Assumption ;

2. `Cases` $(s) = \mathit{false}$;
3. `Nesting` $(s) = 1$;
4. `Contents` $(s)(0) = A_s(\vec{a})$

MP : One of the given axioms $AX$ has been applied, i.e., the following conditions are met for some $k$, $0 \leq k < |AX|$, where the $k$-th axiom is: $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x}, \vec{y}) \vee \ldots \vee B_{m-1}(\vec{x}, \vec{y}))$ (where $m$ is 1 or 2):

1. `StepKind` $(s)$ = MP ;
2. `AxiomApplied` $(s) = k$;
3. There is a finite mapping `Instantiation` $(s)$ that maps each quantified variable from $\vec{x}$ to a constant: `Instantiation` $(s)(x_i) = a_i$ and also from $y_i$ to a new, fresh constant: `Instantiation` $(s)(y_i) = b_i$ (other arguments of atomic formulae are constants);
4. `Cases` $(s) = \mathit{false}$, iff $m$ equals 1;
5. `Nesting` $(s)$ = `Nesting` $(s-1)$, if $s > 0$, otherwise `Nesting` $(s) = 1$;
6. `Contents` $(s)(0) = B_0(\vec{a}, \vec{b})$ and, if $m = 2$, `Contents` $(s)(1) = B_1(\vec{a}, \vec{b})$;
7. The mapping `From` $(s)(i)$ maps each $i$, $0 \leq i < n$ to one of numbers $s'$, such that $0 \leq s' < s$ and such that in the $s'$-th proof step the fact $A_i(\vec{a})$ was derived. In addition, it must hold that:
   - `Cases` $(s') = \mathit{false}$ ($s'$ must not be branching step, it must contain a single fact, not a disjunction of two);
   - the steps $s$ and $s'$ are on the same proof branch.[8]

FirstCase :

1. `StepKind` $(s)$ = FirstCase ;
2. $s > 0$;
3. `StepKind` $(s-1)$ = MP ;
4. `Cases` $(s-1) = \mathit{true}$;
5. `Cases` $(s) = \mathit{false}$;
6. `Nesting` $(s) = 2 \cdot$ `Nesting` $(s-1)$;
7. `Contents` $(s)(0)$ = `Contents` $(s-1)(0)$.

SecondCase :

1. `StepKind` $(s)$ = SecondCase ;
2. $s > 0$;
3. the step $s-1$ is one of the QED steps;
4. there is a step $s'$, $0 \leq s' < s-2$ such that;
   - `StepKind` $(s')$ = MP ;
   - `Cases` $(s') = \mathit{true}$;
   - `Contents` $(s)(0)$ = `Contents` $(s')(1)$;
5. `Cases` $(s) = \mathit{false}$;
6. `Nesting` $(s)$ =`Nesting` $(s-1) + 1$.

QEDbyCases :

1. `StepKind` $(s)$ = QEDbyCases ;
2. $s > 0$;
3. the step $s-1$ is one of the QED steps;
4. `Goal` $(s)$;
5. `Nesting` $(s-1)$ is odd;
6. `Nesting` $(s)$ = (`Nesting` $(s-1) - 1)/2$.

[8] The way we represent this condition is described in Section 4.

QEDbyAssumption :
1. `StepKind` $(s)$ = QEDbyAssumption ;
2. $s > 0$;
3. `Goal` $(s-1)$;[9]
4. `Goal` $(s)$;
5. `Nesting` $(s)$ = `Nesting` $(s-1)$.

QEDbyEFQ :
1. `StepKind` $(s)$ = QEDbyEFQ ;
2. $s > 0$;
3. `Contents` $(s-1)(0) = \bot$;
4. `Goal` $(s)$;
5. `Nesting` $(s)$ = `Nesting` $(s-1)$.

Let us comment more on the encoding of the step kind FirstCase , for instance. The given constraints say that such proof step cannot be the very first proof step (i.e., the 0th step), that the previous step is a branching MP step, the step itself is not branching, and its contents is the same as the first atom of the previous step, while the nesting doubles.

*Example 5* Consider again a proof from Example 4, and let us illustrate the above constraints on its two last steps.
One step before the last one (6th step) has a step kind QEDbyEFQ . Indeed, $5 > 0$ (recall that internally we use zero-based counting), the contents of its previous step is $\bot$, the contents of this step (not shown in the presented proofs) is $r(a)$, and its nesting is equal to the nesting of its previous step.
The last step (7th step) has a step kind QEDbyCases . Indeed, $6 > 0$, the contents of its previous step is $r(a)$, the contents of this step (not shown in the presented proofs) is, again, $r(a)$, the nesting of the previous step equals 3, hence it is odd, and finally, the nesting of this proof step equals (3-1)/2.

### 3.8 Encoding of Proof as a Whole – Global Constraints

The global constraints ensure that the global structure (most notably case splits) is correct and that the goal has been reached at the end of the proof. In the following, the proof length equals $L$ ($L$ is unknown, there is only a constraint $0 < L \leq MaxL$, where $MaxL$ is the given, concrete maximal proof length). In all the rules, it must hold that $0 \leq s < L$:

1. Each proof step $s$ is one of the defined step kinds;
2. If step $s-1$ is one of the QED steps, then `Nesting` $(s-1) \neq$ `Nesting` $(s)$;
3. If `Cases` $(s-1)$ is true, then `StepKind` $(s)$ = FirstCase ;
4. If $s-1$ is one of the QED steps and `Nesting` $(s-1)$ is even, then `StepKind` $(s)$ = SecondCase ;
5. If $s-1$ is one of the QED steps and `Nesting` $(s-1)$ is odd, then `StepKind` $(s)$ = QEDbyCases ;
6. `Nesting` $(L-1) = 1$;

[9] For the very first proof step following the given assumptions, there should be also a constraint corresponding to a posibility that the goal has been met by some of the given assumptions.

7. The step $L-1$ is one of the QED steps;
8. `Contents` $(L-1)$ = `Contents` $(MaxL-1)$.

Notice that, without uninterpreted functions, a condition like `Nesting` $(L-1) = 1$ cannot be expressed directly because $L$ is unknown. Instead, the constraint is constructed as a disjunction over all possible values of $L$.

### 3.9 Proof Search

Given some axioms, a conjecture, and a maximal proof length *MaxL*, a set of constraints (in variables ranging over natural numbers) that describe a proof of the conjecture can be constructed, as explained above. If the set of constraints is satisfiable, its model can be decoded into a concrete proof.

A proof search can be facilitated in the following way: first, a proof of length at most $a$ is initially sought, and if there is no such proof, a proof of length at most $a+b$, $a+2b$, $a+3b$ etc, is sought (in our implementation, there are default values for $a$ and $b$, but also the user can set $a$ and $b$, see Section 4.4). Such policy enables faster proof finding for simple theorems and still preserve completeness for more complex ones (while trying to introduce a small number of constraints). The search stops if a proof is found, or the maximal proof length has been reached, or if the available time has been spent.

### 3.10 Properties

Given a maximal proof length, the presented proving method is trivially terminating. In addition, even without a given maximal proof length, if the input statement is provable, and the method is applied for an increasing sequence of maximal proof lengths, this process is also terminating. Hence, one of the key issues for many proving paradigms – termination – is trivial for the presented approach. The form of axioms does not affect this – termination is trivially ensured also in the presence of axioms that may lead to non-terminating rewrite rules, like commutativity axioms.

The presented approach is sound: what is generated as a proof of the given conjecture is indeed its proof. The presented approach is complete in the following sense: if there is a CL2 proof of length $\leq$ *MaxL* of the given conjecture, it will be eventually found (assuming that the underlying SAT/SMT provers are complete). Even more, we can systematically increase *MaxL* (and the maximal nesting, if needed) and check if there is a proof of length $\leq$ *MaxL*. A CL2 formula is a consequence of the given axioms if and only if there is a *MaxL* such that there is a proof of length $\leq$ *MaxL*. This gives a semi-decision procedure for validity in CL2, as a by-product of the proposed approach. It can be extended to full FOL, since any first-order theory can be translated into CL2.

The above outlined features could be the subject of rigorous proofs and formalisation using an interactive prover and a reflexive approach, but at this, first stage we rather chose to use an approach based on certification of generated proofs using state-of-the-art ITP, instead of proving correctness and completeness of our prototype implementation. This approach has several advantages:

- we can change the prover easily, without changing the proof checker and we do not need to write a new proof;
- we can use efficient implementations, with optimisations which would be hard to verify;
- we can choose the implementation language and we could use parallelisation and native computation (without the layers of abstraction/interpretation involved in a reflexive approach);
- we can use external tools that we do not need to verify (SAT and SMT solvers) nor use certified versions[10].

On the other hand, the drawback is that we cannot be completely sure that the encoding is correct, so our prover may possibly fail.

The complexity of the proposed approach is difficult to estimate. In terms of the number of constraints, the largest portion goes to describing MP steps. If there are $A$ axioms, each of them with maximally $p$ premises, and maximally $a$ arguments, then the number of these constraints for the $l+1$-th proof step can be bounded by $A \cdot a \cdot l^p$. If $MaxL$ is the maximal proof length, by summation for $l = 0, \ldots, MaxL-1$, the number of these constraints belongs to $O(A \cdot a \cdot MaxL^{p+1})$. However, it is very well known that the number of the constraints (in SAT or SMT) is not simply or directly related to the complexity of the solving phase. Thus, we will stick only to some experimental evaluation, rather than on theoretical worst-case or average-case analysis (see Section 5). In practice, the encoding and decoding phases of the prover take negligible time compared to the SAT/SMT solving phase.

### 3.11 Extensions

We considered a number of extensions of the basic approach described above. Some of them were not beneficial, like describing constraints in an incremental manner (for instance, using conditions for a proof step in conditions for the subsequent proof step), using integrated support for equality axioms (instead of external, individual axioms for each predicate symbol), using integrated support for excluded middle axioms (instead of external, individual axioms for each predicate symbol), etc. In the following, we describe the extensions that had positive impact on efficiency of the approach.

*Symmetry Breaking.* As in many other kinds of constraint problems, symmetry breaking can be exploited in the presented approach. Symmetry breaking can lead to more concise constraints and to a more efficient solving phase. However, some care needs to be taken that additional constraints do not disturb the search space, in our context, that is, do not eliminate some proofs of interests.

One of the additional constraints that we use for symmetry breaking imposes a partial ordering of proof steps and reduces a number of possible proofs: if two MP steps $s$ and $s'$ are such that $s+1 = s'$, $\texttt{Nesting}\,(s) = \texttt{Nesting}\,(s')$, and $s'$ does not use $s$, then it must hold that:

[10] Even if there are broadly available SAT and SMT solvers which have been tested on a large amount of problems by a large amount of users and even some integrated in interactive proof assistant using either a reflexive approach or one based on checking certificates, we think it is simpler to use the final reconstructed proof for the original statement as a certificate.

$\texttt{Premises}\ (s') > \texttt{Premises}\ (s)$ or
$(\texttt{Premises}\ (s') = \texttt{Premises}\ (s)$ and $\texttt{AxiomApplied}\ (s') \geq \texttt{AxiomApplied}\ (s))$.

*Axioms Inlining.* We call the axioms of the following forms (universally closed, $\vec{x}$ is a sequence of variables, $A(\vec{x})$ and $B(\vec{x})$ denote atomic formulae):
$B(\vec{x})$
$A(\vec{x}) \Rightarrow B(\vec{x})$
*simple axioms.* In one extension of our basic approach, these axioms are not to be applied through MP steps. Instead, in a MP step with a non-simple axiom applied, premises of the axiom can be justified not only by previous proof steps, but also by suitable, "inline" applications of simple axioms. This makes constraints for MP steps more complex but, on the other hand, enables shorter proofs.

*Example 6* Let us consider axioms:
ax1: $\forall x\ q(x)$
ax2: $\forall x\ (q(x) \Rightarrow r(x))$
ax3: $\forall x\ (p(x) \wedge q(x) \wedge r(x) \Rightarrow s(x))$
and a conjecture:
$\forall x\ (p(x) \Rightarrow s(x))$.
A proof of this theorem, obtained with inlining (and with the derived lemma $\forall X\ r(X)$)) looks as follows.

> Consider an arbitrary $a$ such that: $p(a)$. It should be proved that $s(a)$.
>
> 1. $s(a)$ (by MP, from $p(a)$, $q(a)$, $r(a)$ using axiom ax3; instantiation: $X \mapsto a$)
> 2. Proved by assumption! (by QEDas)

Since simple axioms cannot be applied on their own within MP steps (all applications of simple axioms are kept implicit within applications of other axioms), it is necessary to derive all their consequences in the form of simple axioms and add them to the set of the original axioms (we call this process "saturation").

This technique contributes to the generation of readable proofs because simple axioms are often omitted in natural language proofs. For example, in geometry (which is our main source of applications), the following axioms, where $Col$ is a predicate to designate colinearity, are simple:
$\forall ABC\ (Col(A,B,C) \Rightarrow Col(B,A,C))$
$\forall AB\ Col(A,A,B)$
$\forall A\ A = A$

A mathematician would hardly feel the need to explicitly state that two identical points are collinear with a third one, or that a point is equal to itself, even if these facts are needed for application of some theorems in some special cases. Readability of generated proofs is further discussed in Section 6.3. Fortunately, keeping such proof steps implicit also improves the efficiency of the prover.

*Memoization.* Memoization is a technique for speeding-up computations by storing results of certain function calls and returning the result when the same inputs occur again. In our context, instead of concrete values about proof steps we have only symbolic values. But something like memoization can still be applied. Some

concrete conditions, for instance "proof steps 3 and 7 are on the same proof branch" may occur in several constraints. For such conditions, we introduce new variables, link the variable to the condition by a constraint, and in other constraints use the variable instead of the full condition. This leads to more concise constraints and can facilitate the solving process[11].

## 4 Implementation

We implemented the above approach as a prover called Larus,[12] a C++ application, with a support of several external tools.[13] Although our implementation is fully functional, it is not fully optimised and should still be considered a prototype implementation, developed along the way of exploring the new automated proving approach. The approach is implemented in several variants, in a form of several proving engines (they have been implemented alongside one proving engine based on using forward reasoning and the data structures available via the STL (Standard Template Library) C++ library ). We have implemented support for encoding proofs in SAT (via the constraint solving system URSA) and in several SMT theories. The solving phase uses SAT and SMT solvers. We implemented support for *filtering* by external provers such as Vampire [44] which eliminates unnecessary axioms (explained in more details in Section 4.1). We also implemented a simplification procedure that eliminates redundant proofs steps (from proofs that are not shortest possible). The implementation consists of around 8000 lines of code (around 7000 lines without the STL-based engine).

The overall proving pipeline in this implementation looks as follows:

1. read the axioms and the conjecture given in the TPTP/fof format;
2. use filtering;
3. transform the axioms and the conjecture from CL to CL2 form;
4. use filtering;
5. encode a proof of the conjecture — on the basis of the given axioms and the conjecture, generate constraints in terms of dedicated variables and possibly (for some SMT-based engines) in terms of interpreted (e.g., +) and uninterpreted functions (e.g., `Nesting` );
6. invoke a suitable solver and retrieve the model found (assuming it exists);
7. read the values of relevant variables (some are redundant) and reconstruct the CL2 proof encoded as a sequence of natural numbers;
8. read the CL2 proof encoded as a sequence of natural numbers, decode it and construct a proof in terms of internal data-structures;
9. simplify the proof (delete redundant proof steps);
10. transform the proof from CL2 to CL form;

[11] This optimisation may be useless when the underlying SAT or SMT prover uses a common sub-expression elimination pre-processing optimisation.

[12] *Larus* is Latin for seagull. A symbolic meaning sometimes attached to seagull is: looking at the problem at hand from a different angle. That may be appropriate for a prover based on a new proving paradigm: looking for a proof as a whole.

[13] The source code of the system and the benchmarks are publicly available from here: `https://github.com/janicicpredrag/Larus`. For running the system, one needs also one of the tools URSA [36] or `z3` [50].

11. transform the proof in terms of internal data-structures into one of supported export formats, such as Coq proofs or natural language LaTeX proofs.

We skip technical steps and discuss only some key design and implementation details.

### 4.1 Pre-processing and Filtering

Sledgehammer is a tool (nowadays – rather a methodology) that enables invoking automated theorem provers from the interactive theorem prover Isabelle [60]. If external provers prove the given goal, they provide information that are used for reconstruction of a proof object within the interactive theorem prover. One of the tasks that external provers ( e.g., Vampire) do is filtering relevant axioms/clauses given. The user can provide hundreds of available axioms/clauses and filtering can, using different techniques, discard many of them, before an attempt to prove the theorem. We use a similar approach for pre-processing the axiom set before trying to prove the conjecture by our prover.

For filtering we use the FOL theorem prover Vampire [44]. In the first filtering phase, the given conjecture with all given axioms (in CL form) is sent to the external prover (Vampire). If it proves the conjecture, the list of used axioms is extracted and forwarded to further proving steps (otherwise, the set of all axioms is used); The second filtering phase, after the axioms are transformed into CL2 form, goes through several stages:

- the given conjecture with all the axioms in CL2 form is sent to the external prover. If it proves the conjecture, the list of used axioms is extracted and forwarded to further proving steps (otherwise, the set of all axioms is used);
- the equality predicate symbol is replaced by a dedicated predicate symbol, and the axioms and the conjecture are sent to the external prover. The point is in avoiding using full native support for equality (in both the external and the presented prover). If the external prover proves the conjecture, it returns only instances of equality axioms that are really needed and only them will be used in the next proving steps. Otherwise, if the external prover fails, all instances of equality axioms will be used.
- the negated predicate symbols are replaced by dedicated predicate symbols, and the axioms, along with the axioms $\forall \vec{x}(R(\vec{x}) \wedge \overline{R}(\vec{x}) \Rightarrow \bot)$, $\forall \vec{x}(R(\vec{x}) \vee \overline{R}(\vec{x}))$, for each predicate symbol $R$, and the conjecture are sent to the external prover. The point is in avoiding using the full native support for negation (in both the external and the presented prover). If the external prover proves the conjecture, it returns only the above axioms that are really needed. Otherwise, if the external prover fails, all of them will be used.

Each filtering stage is optional and the proving procedure may go without them. Note that, if the filtering is used, it is not guaranteed anymore that the obtained shortest proofs are indeed the shortest over the initial set of axioms (since the external prover may have reached a proof based on some other axioms).

Apart from the above filtering based on an external prover, we have also implemented filtering based on reachability. We say that a predicate symbols is *reachable* if it occurs in the premises of the conjecture, or can be reached by applications of axioms in which all premises are over reachable predicate symbols. An axiom with a premise that is not over a reachable predicate symbols can be eliminated.

### 4.2 Implementation of the Proof Encoding

The notion of CL2 proof represented as a sequence of natural numbers is common for all the implemented constraint-based proving engines. Given some axioms and the conjecture, constraints describing a proof can be expressed in several ways. We support the following:

- the constraints can be expressed in a high-level representation language offered by our constraint system URSA;
- the constraints can be expressed in a low-level representation language of SMT-lib.

Substantially, the constraints are the same but the URSA-based approach leads to a self-contained, concise proof specification (that includes representations of the axioms and the conjecture). For instance, constraints saying that a proof step is FirstCase step are expressed in the following way (the URSA code is generated by our prover):[14]

```
 bFirstCaseStep = (nAxiomApplied[nProofStep] == nFirstCase)
                  && nProofStep>0
                  && bCases[nProofStep-1]
                  && !bCases[nProofStep]
                  && (nNesting[nProofStep-1] <= nMaxNesting)
                  && (nNesting[nProofStep] == (nNesting[nProofStep-1]<<1))
                  && (nP[nProofStep][0]==nP[nProofStep-1][0]);
for (nInd = 0; nInd < nMaxArg; nInd++)
    bFirstCaseStep &&= (nA[nProofStep][nInd]==nA[nProofStep-1][nInd]);
```

In the URSA-based approach, representations of different theorems differ only in the encoding of axioms and the conjecture itself, while the encoding of the notion of proof is common. It consists of only around 300 lines of URSA code. Therefore, development of an automated theorem prover for CL is reduced to a relatively small human effort of making this 300-lines specification, plus encoding of specific axioms and goals. The difficulty in our work was, however, the road from the initial general idea to the fully functional system requiring a number of peripheral modules, and also exploring a number of variations of encodings and looking for one that leads to efficient solving by SAT provers. We have experimented with different variations, but we think there is still room for improvements. With the experience gained and the infrastructure developed, we believe that a support for another logic, contained in another few hundreds lines, would take significantly less time.

In the SMT-based approach, the specification of the notion of proof is spread through C++ code which is later translated to SMT constraints. This specification is less readable than URSA specification, but leads to more efficient theorem proving.

[14] There are some small differences between the description of proof step FirstCase (Section 3.7) and the implementation. It can be shown that the condition `StepKind` $(s-1) =$ MP , is redundant (since the previous step has to be branching), so it was omitted in the implementation. The constraint `nNesting[nProofStep-1] <= nMaxNesting` is not given in the description of proof step FirstCase . Indeed, this constraint is optional. Limiting the maximal nesting simplifies the problem and cuts-off some parts of the search space. If there are no branching axioms, by using this constraint it can be easily stated that there are no nesting at all.

URSA constraints expressed in the above way are processed by URSA, bit-blasted into SAT instance and sent to a SAT solver. On the other hand, in SMT-based engines, the constraints are right away expressed in terms of the low-level SMT representation. The following C++ code generates constraints saying that a proof step is a FirstCase step in SMT-lib form (for the case `nProofStep` $> 0$, `sbFirstCaseStep` is trivially false otherwise):

```
sbFirstCaseStep = "(and " +
                    appeq(app("nAxiomApplied", nProofStep), eFirstCase)
                    app("bCases", nProofStep-1) +
                    "(not " + app("bCases", nProofStep) + ") " +
                    smt_less(app("nNesting",nProofStep-1),
                       mParams.max_nesting_depth+1) +
                    appeq(app("nNesting", nProofStep),
                          smt_prod(app("nNesting", nProofStep-1),2)) +
                    appeq(app("nP", nProofStep, 0),
                          app("nP", nProofStep-1, 0));
for (unsigned nInd = 0; nInd < mnMaxArity; nInd++)
    sbFirstCaseStep += appeq(app("nArg", nProofStep, nInd),
                             app("nArg", nProofStep-1, nInd));
sbFirstCaseStep += ")";
```

If the equality symbol occurs in the conjecture, the equality axioms are added to the axiom set. If inlining is to be used and if there is equality in the axioms, then, in addition to simple axioms, equality substitution axioms are also inlined.

Arithmetic constraints can be represented in linear arithmetic or in bitvector arithmetic. Constraints involving functions can be represented in terms of uninterpreted functions (such as `Nesting` ) or Ackermanization can be used. So, there are four SMT-based encodings: LIA (linear integer arithmetic), BV (bit-vector arithmetic), UFLIA (linear integer arithmetic with uninterpreted functions), UFBV (bit-vector arithmetic with uninterpreted functions).

A proof is encoded as a finite object: there is a maximal number of proof steps, a maximal number of premises of the axioms, a maximal nesting , a maximal number of new constants introduced in each step, etc. Zero-based counting (like in C/C++) of proof steps, atomic formulae within steps, etc. is used, for a more compact representation.

The maximal number of premises of the axioms and maximal arity in predicate symbols can be computed based on the axioms. Concerning the maximal number of proof steps, we preserve completeness by an iterative increase of this constant (as discussed in 3.9). For ensuring completeness, the same should be done for the maximal nesting.

The logical constants $\bot$ and $\top$ are accepted as possible atoms in CL formulae, and they need a special treatment, since their truth values are fixed. Hence, such treatment augments the description of MP steps and constraints describing the goal.

In order to make the encoding simpler, there is the same number of slots for arguments in constraints for all atomic formulae (that number is determined based on the theory signature such that it can accommodate arguments for all predicate symbols). That brings some constraints/memory overhead, but it also enables simpler encoding. In the same spirit, new constants are represented by new natural

numbers: $w \cdot s + i$, $i = 0, \ldots, w - 1$, where $s$ is the number of the proof step, and $w$ is the maximal number of existential quantifiers in the axioms.

The encoding depends on the number of binary digits in URSA, BV, UFBV-based approaches. The default value is 12.

As a kind of optimisation, the information of `StepKind` and `AxiomApplied` for one proof step are stored in one number representing that step: the possible values are firstly possible step kinds (except MP ), and then axioms (with the ordinal numbers starting from the last possible step kind). Based on the value in the model, it is then trivial to decode `StepKind` and, if relevant – `AxiomApplied` .

In all proving engines, two proof steps $s$ and $s'$, while $s$ precedes $s'$ are on *the same proof branch* if, considering the binary representation of these numbers, some prefix of the nesting of $s$ equals the nesting of $s'$: PREFIX(`Nesting` $(s)$) = `Nesting` $(s')$. This condition can be represented in all encodings by listing several possible cases (i.e., cases for all possible prefixes).

We find the URSA-based approach more flexible and better suited for debugging. However, the SMT-based approach with BV gives a better performance and most of the performance results given below are obtained using this encoding.

Generated SAT and SMT formulae are usually large, even for simple problems. For instance, for the conjecture from Example 10, the search for a proof of length 14 (via URSA) generates a SAT formula with 40817 variables and 249212 clauses. The SMT file with the corresponding problem description for the theory of bit-vectors has 558 variables and the size of 815 Kb.

### 4.3 Implementation of Proof Decoding and Export

Generated constraints are passed to underlying solvers: `clasp` [30] for URSA-based engine, and `z3` [50] for SMT-based engines (although other solvers can be used as well). If the set of constraints is satisfiable, the solver returns a model. The values of relevant variables are read and a proof outline is generated (an example is shown in Section 3.5). This outline is read and a proof in internal (C++) representation is constructed.

*Example 7* Let us again consider the proof given as a sequence of numbers in Section 3.5. The sequence

```
1.  1 13 1    4 0 6 0
```

is decoded as follows:

- the nesting equals 1;
- the axioms are numbered from 13, hence, the step kind is MP and the axiom applied is the 0th axiom (i.e., $ax1$),
- the derived formula is branching;
- in the derived formula, in the first disjunct, the dominant predicate symbol is the 4th predicate symbol (i.e. $r$), and in the second disjunct, the dominant predicate symbol is the 6th symbol (i.e. $q$). In both, the argument is the 0th constant (i.e. $a$).

The two branches in the proof have nesting 2 and 3 (10 and 11 in binary representation) and, according to the given criterion, they are not on a same proof branch, as expected.

The internal proof representation is simply exported to a needed output format. Currently, there is support for the Coq format, and for an informal, natural language proof format (an example is shown in Section 3.5)

For Coq proofs, we preserve the forward chaining and declarative style of the proof by using the standard `assert` tactic of Coq and some user defined tactics to keep the small granularity in the proofs as in the natural language output. For example, we introduce a tactic `by cases on`, allowing to perform case distinction on previously proved logical disjunctions by referring to the actual statement rather than the name of the hypothesis. Contrary to other CL provers (such as the one of Bezem [7]), the export to Coq is expressed at the tactic level rather than as a lambda term. The export as a sequence of Coq tactics has several advantages:

- It allows the use of Coq's automation to both reduce the size of the proof and ease the checking of the proof by rebuilding some pieces of information at the checking time.
- It preserves the readability and maintainability of the proof.

The tactic `conclude` solves the remaining goal if it is a previously proven statement, or a disjunction containing a previously proven statement instantiating the potential existential variables. Conjunctions are split by the tactics implicitly at each proof step. As an example, we provide the Coq proof for Example 1:

```
Require Import src.general_tactics.
Require Import Classical.

Section Sec.

Parameter MyT : Type.
Parameter p : MyT -> Prop.
Parameter r : MyT -> Prop.
Parameter q : MyT -> Prop.

Hypothesis ax1 : forall X : MyT, p X -> r X \/ q X.
Hypothesis ax2 : forall X : MyT, q X -> False.


Theorem example_with_false : forall X : MyT, p X -> r X.
Proof.
intros a.
intros.
assert (r a \/ q a) by applying (ax1 a).
by cases on ( ( r a )  \/  ( q a ) ).
- {
   conclude.
  }
- {
   assert (False) by applying (ax2 a).
   contradict.
  }
Qed.

End Sec.
```

For support of simple lemma inlining, we first prove the lemma obtained by saturation in sequence using automation. Those lemmas are put in a so-called "hint base". The lemmas are then used implicitly in the main proof. For each proof step which requires a given premise, the premise can be obtained by assumption or by using the inlined lemmas.

### 4.4 Configuring

The provers can be configured using several parameters, including the following ones: an input format, an initial (maximal) proof length, a final (maximal) proof length, a maximal nesting, a flag whether a shortest proof should be found, a flag whether proof simplification is to be used, a flag whether axiom inlining is to be used, a flag whether excluded middle axioms (for existing predicates) are to be used, a flag whether negation elimination axioms (for existing predicates) are to be used, a flag whether filtering is to be used, a flad saying if an interactive theorem prover should verify generated proofs, and a time limit.

## 5 Performances and Evaluation

For evaluation, we cannot use the standard TPTP benchmarks because the benchmarks are not in coherent logic form. Despite the fact that these benchmarks could be translated to CL by some procedure [26], we want to test our implementation on goals which are naturally in coherent form. We ran our implementation on several corpora listed below.

We compared performance of our prover to performance of several state-of-the-art, award-wining FOL provers (Vampire, E, Iprover), provers with a small implementation (leanCoP, nanoCoP), one prover based on coherent logic (Geo), a prover generating Coq proofs (Zenon), some Coq tactics (first-order and Coq Hammer) and a prover using a technique related to ours (ChewTPTP). We did not test other CL based provers because they do not accept TPTP syntax (but the prover we tested (Geo) was the most efficient prover among several CL provers in one previous study [34]). The list of provers/versions is the following:

- Vampire[15] 4.4 using CASC mode [44], a superposition-based theorem prover;
- E[16] 2.2 in auto mode, a superposition-based theorem prover;
- Iprover[17] 3.1e [43], an instantiation-based theorem prover for first-order logic;
- leanCoP[18] 2.1 [58], a prover based on the connection (tableau) calculus;
- nanoCoP[19] 1.1 [57], a prover based on the non-clausal connection calculus for classical logic;
- Geo[20] 2007f, a prover based on geometric resolution;
- ChewTPTP[21] 1.0.0 [15], which proves rigid first-order theorems by encoding the existence of a first-order connection tableau in SAT;
- Zenon[22] 0.8.4 [16], a prover based on the tableau method and capable of producing Coq proofs;
- Standard first-order tactic of Coq, a reflexive implementation of a first-order prover [21];
- Coq Hammer Tactics[23] v1.3 [22].
- Isabelle Hammer in TPTP mode without proof reconstruction[24] [59].

For converting problems from TPTP to Coq, we used tptp2coq[25]. For converting problems from TPTP/fof to the clausal form required by Geo and ChewTPTP, we used Vampire. The evaluation was performed on a PC working under Linux, with Intel(R) Core(TM) i5 CPU 750 @ 2.67GHz, 20GiB SDRAM.

[15] https://vprover.github.io/
[16] https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html
[17] http://www.cs.man.ac.uk/~korovink/iprover/
[18] http://www.leancop.de/
[19] http://leancop.de/nanocop/
[20] http://www.ii.uni.wroc.pl/~nivelle/software/geo/index.html
[21] https://github.com/erimcg/ChewTPTP
[22] http://zenon.inria.fr/
[23] https://github.com/lukaszcz/coqhammer
[24] https://isabelle.in.tum.de/
[25] https://github.com/lukaszcz/tptp2coq

### 5.1 *Elements* corpus

This corpus is the set of 234 lemmas and propositions coming from the formalisation of the first book of Euclid's Elements [5]. The proofs of these theorems, constructed by humans, have already been checked by both Coq and HOL-Light interactive provers. All lemmas and propositions are naturally expressed in CL or can be very simply modified to belong to this fragment. This is a rich, real-world corpus, with instances ranging from trivial to very complex. But we agree with Avigad that this kind of benchmarks are not fully realistic for testing effective usability of automation in an ITP context in practice [3]. For example, within 100s, Vampire can prove 25 and Larus can prove 13 of the 73 statements corresponding to the propositions from the first book of Euclid's Elements, but the reader should not interpret these results as "ATP can solve automatically around a third of Euclid's propositions." Namely, the corpus contains all lemmas and axioms that are necessary for proving Euclid's propositions, and those have been discovered in the first place only by building the proofs manually using a proof assistant.

The formal proofs constructed manually allow us to compare them to the proofs generated automatically by our prover, but also to evaluate the difficulty of the theorems by measuring the size of the handcrafted formal proofs.

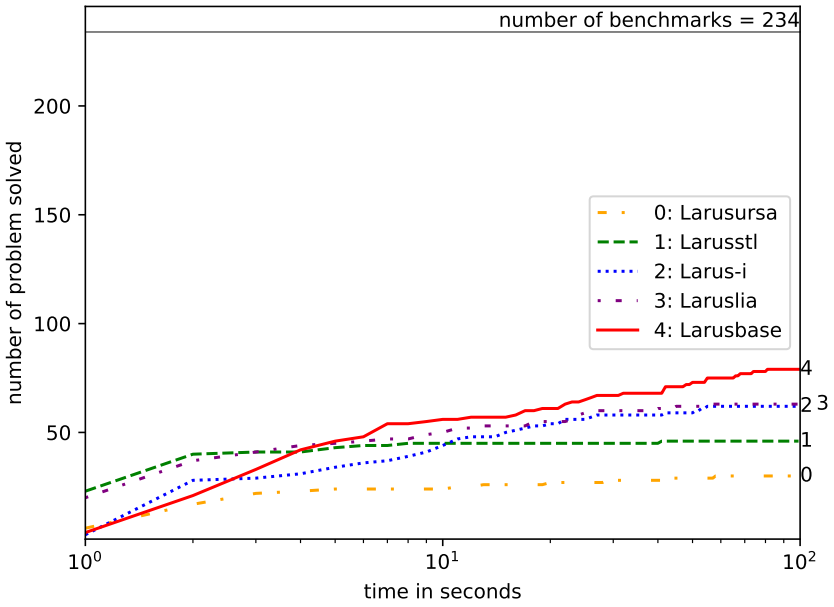



**Fig. 1** Comparing different parameters for Larus

*Comparison between different prover configurations.* On this corpus, we compared the performance of different configurations of the Larus prover. For the base configuration, we used the SMT-BV based engine and the solver `z3`, with an initial proof size of size 8, using lemma inlining, and without filtering. We also used (all without filtering):

Larusstl: based on a pure forward chaining saturation algorithm (using simple STL C++ structures);
Laruslia: based on the SMT-LIA based encoding and the solver `z3`;
Larusursa: based on the encoding of the problem into SAT, using the system URSA and the underlying SAT solver `clasp`;
Larus-i: based on the SMT-BV encoding and the solver `z3`, but without lemma inlining.

The experimental results, shown in Figure 1, show that the best configuration (among considered ones) is the base configuration. The simple forward-chaining engine showed a typical behaviour: what can be proved is what can be proved in a short time, and then providing more time does not improve that much the results. In further experiments and comparisons to other provers, we consider only the base, SMT-BV-based configuration of Larus (Larusbase), with fixed, suitable parameters for specific corpora.

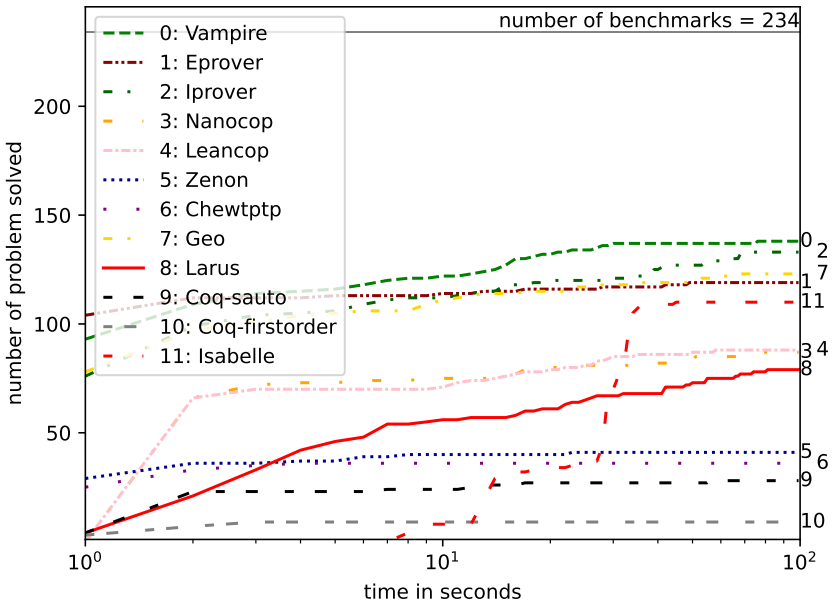



**Fig. 2** Experimental results for Euclid's Elements benchmarks

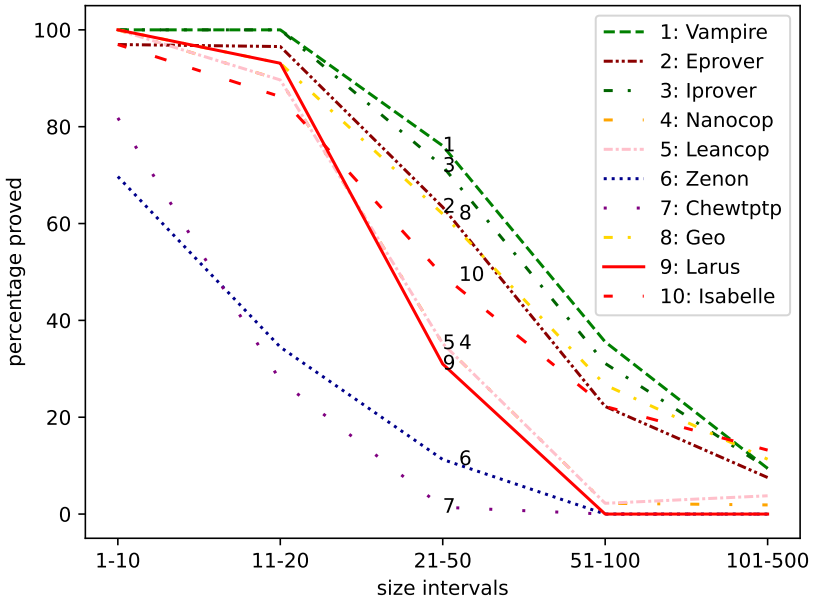



**Fig. 3** Percentage of lemmas proved within 100s compared to the length of the handcrafted formal proof.

*Comparison with other provers.* Comparing to other provers (Figure 2), Larus' performance is similar to some provers that have small implementations, but are still very efficient, such as Leancop. State-of-the-art general FOL provers give a significantly higher success rate. Overall, the best performance was by Vampire. Larus outperforms other provers producing Coq proofs such as Zenon, or Coq tactics.

We have verified experimentally that the size of the formal proof built interactively is a good measure of the difficulty of the theorem for ATP, as shown in Figure 3. For all the provers, the percentage of theorem proved against the size of the human-crafted proofs is more-or-less strictly monotonically decreasing. All the provers except ChewTPTP and Zenon can prove almost 100% of theorems with short proofs. Larus meets this patterns too for short proofs, but Larus cannot find large proofs with more than 50 steps. However, as Larus proceeds by increasing progressively the size of potential proof, along the way, Larus also confirms the absence of proofs of shorter sizes. On all our benchmarks, Larus' results are better than those of ChewTPTP, which is a similar approach based on SMT solvers, and this shows that encoding within CL gives better results. The results for Isabelle have the shape with steps because Isabelle first tries some tactics that cannot solve the goals, and later calls state-of-the-art provers such as Vampire that can solve some of these goals instantly.

### 5.2 *Coherent* corpus

This corpus is the set of 64 benchmarks used in previous research[26] about automatic theorem proving in CL [34]. These benchmarks contain both simple and difficult examples. Some originate from the formalisation of Hessenberg's theorem [8].

Some theorems, if considered in the CL setting, require proofs with no less than 100 or 200 steps, which was too difficult for our prover (for some of these, there is a small search space for the forward chaining approach, hence they can be easily proved by the engine Larusstl). On these benchmarks, the Geo prover competes well with state-of-the-art provers (Figure 4). Again on these benchmarks, Larus cannot compete with state-of-the-art FOL provers, but outperforms ChewTPTP, Zenon and the Coq tactics.

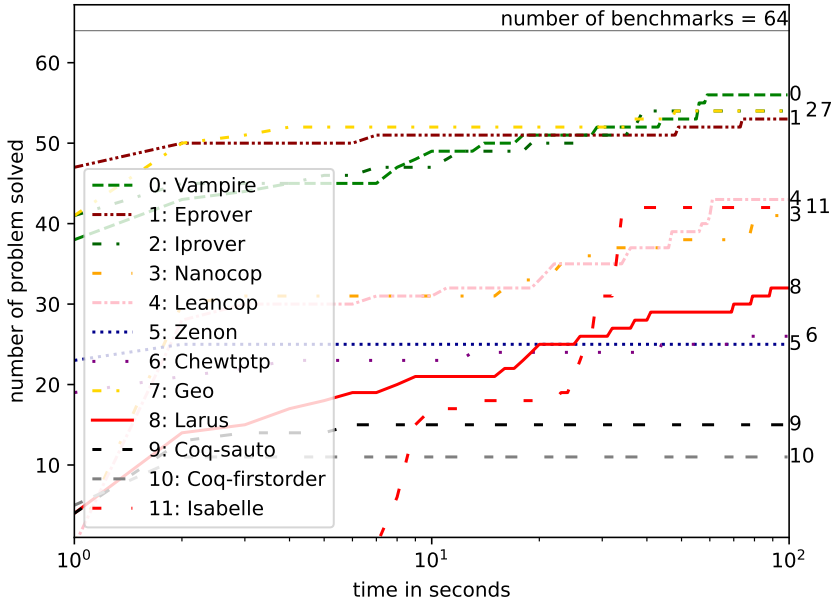



**Fig. 4** Experimental results for coherent logic benchmarks

### 5.3 *col-trans* corpus

This corpus is a set of 1361 sub-goals used in the formalization of geometry concerning some properties about pseudo transitivity of collinearity which are easy to solve using an ad-hoc procedure [18], but are more challenging for some general

[26] `https://code.google.com/archive/p/clp/source`

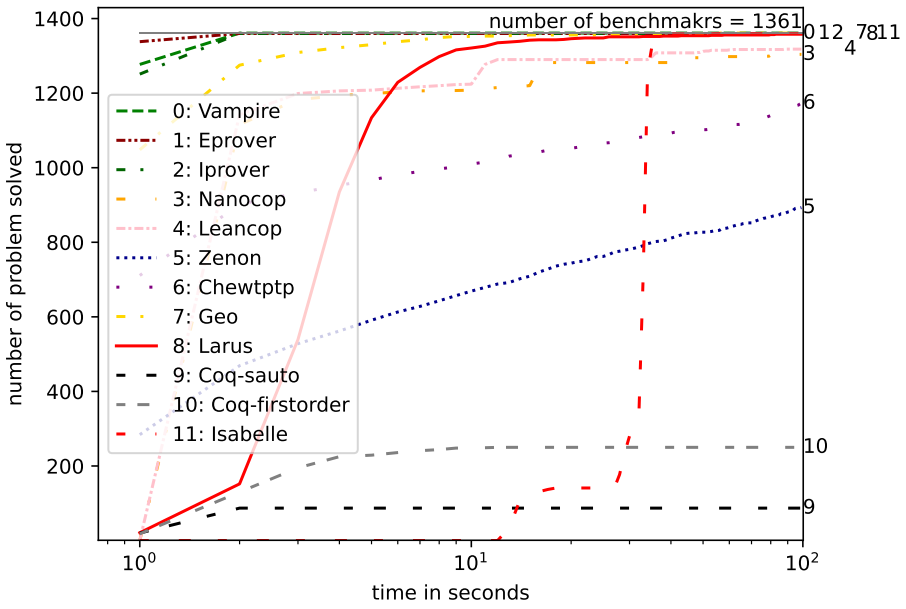



**Fig. 5** Number of problem solved in less than some given time on the col-trans corpus

purpose ATPs. This corpus originates from a concrete application: we would like to export the large Coq library of geometry proofs GeoCoq[27] to Logipedia [25]. However, in the Coq formalisation, these sub-goals are currently solved by a reflexive Coq tactic (called `ColR`), the result of which cannot currently be checked by Dedukti. We do not include the `ColR` tactic in the benchmarks because it is not a general purpose prover. This corpus is a challenge for hammers such as Coq Hammers, because the set of axioms needed is already known and is small. Hence, calling a state-of-the-art prover does not help to reduce the difficulty of the proof search within the proof assistant. The results for Isabelle do not include proof reconstruction. Isabelle hammer is running several state-of-the-art provers sequentially: if the first tactics fails, then Vampire is called and can solve the goal. For this corpus, Larus is used with options: no case splits, start by looking for a proof of at most 8 steps, and check the generated proof using Coq .[28]

## 5.4 Crafted corpus

We consider the following schema of conjectures (parametrised by $n$):

ax1: $dom(a_1) \wedge dom(a_2) \wedge \ldots \wedge dom(a_n)$

ax2: $\forall x_1\ \forall x_2\ \ldots \forall x_n (dom(x_1) \wedge dom(x_2) \wedge \ldots \wedge dom(x_n) \Rightarrow p(x_1, x_2, \ldots, x_n))$

[27] `https://geocoq.github.io/GeoCoq/`

[28] Note that in the Coq files generated from TPTP files, the statements are not curryfied. We noticed that this reduces the efficiency of the Coq tactics.

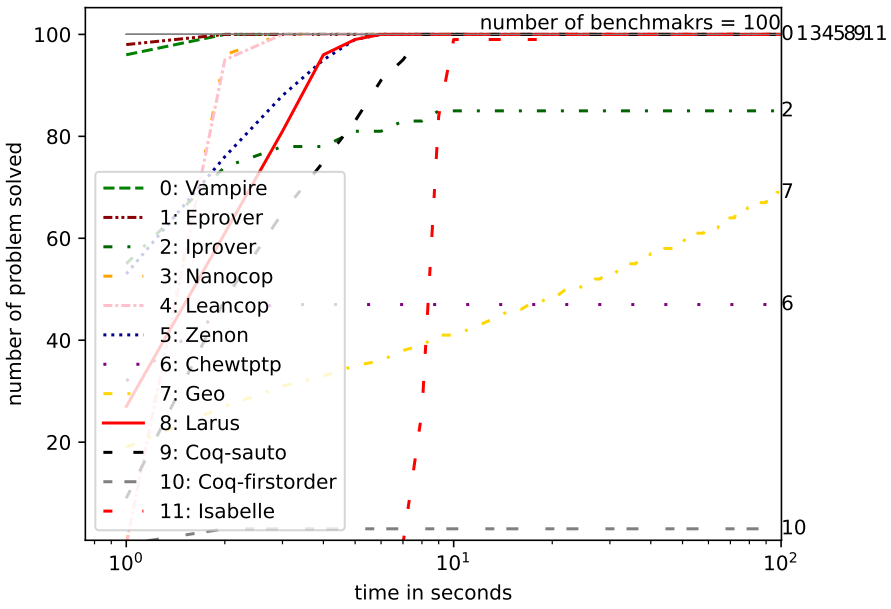



**Fig. 6** Number of problem solved in less than some given time on the crafted corpus

ax3: $p(a_n, a_{n-1}, \ldots, a_1) \Rightarrow goal$
and the following conjecture: $goal$.

We generated instances of the above schema for $n$ equal 1 to 100. This crafted set serves to explore behaviour of our prover in situations where there is a very short proof, but it is hard to be found by procedures based on instantiations and model finding/evolution. Indeed, Larus performed very well on this corpus[29], and outperformed by far Iprover and Geo which had poor performance. Larus' performance was also much better then that of ChewTPTP. On the other hand, all provers based on the resolution method or some related methods had excellent results. These results suggest that automated theorem proving in CL (like in other logics) can benefit from having available different proving techniques (as discussed in Section 8.3).

## 5.5 Summary of Experimental Results

On the considered corpora, the provers Vampire and Eprover showed best performance and Larus' performance was generally worse than performance of these state-of-the-art FOL provers. On most corpora, Larus outperformed provers that generate verifiable proofs such as Zenon, and also the prover ChewTPTP, the

[29] For this corpus, Larus is used with options: no case splits, start by looking for a proof of at most 2 steps, do not use negation elimination . Proof verification using Coq is not used for this corpus (only for this corpus the time needed for proof verification is not negligible).

prover that is most closely related to Larus among the considered provers in terms of the proving approach. Concerning the main target domain, the most closely related prover is Geo, and it outperformed Larus on most of the corpora, but not on the crafted corpus. On this corpus Larus also significantly outperformed Iprover.

Overall, the experimental results suggest that Larus shows reasonable performance and that it can be practically usable in some situations, primarily when a verifiable and/or readable proof is needed.

## 6 Features and Perspectives

The presented approach provides a suitable basis for various applications and upgrades. In this section, we discuss some of them, like generation of shortest, readable, or machine verifiable proofs.

### 6.1 Short Proofs, Shortest Proofs, All Proofs

Shorter proofs are often preferred: for their beauty or for more practical reasons such as saving space in a printed article or in a computer formalisation. The quests for short, simple or simplest proofs are among the oldest quests in mathematics, as argued by Wos [75]:

> The search for elegant proofs has continuously played a key role in mathematics and in logic. Such a search can lead to the discovery of new, important relationships, and it can also lead to the formulation of significant concepts. In addition to the aesthetic aspects of elegance of concern to mathematics and logic, practical aspects also exist, easily illustrated when the focus is on proof length. Indeed, the methodology here may also serve well in the context of constructing more efficient circuits synthesizing more effective algorithms, and the like.

The challenge of finding simplest proofs was even an item (a lost one!) on the famed Hilbert's list of the most significant mathematical problems [69]:

> The twenty-fourth problem in my Paris lecture was to be: Criteria of simplicity, or proof of the greatest simplicity of certain proofs.

There are several ways to define what the simplest proof of a theorem is, but the most common one is based on proof length – the number of inference steps applied [41]. There are just a few concrete results for practical finding of short or shortest proofs automatically, although there are several possible approaches for this task, usually based on some form of exhaustive search and resolution method [41, 71, 75]. Shortest proofs could be obtained also using simple breadth-first search. There are also theoretical considerations on computability of functions that compute shortest proofs.

The approach presented in this paper provides a general method for finding shortest proofs in CL, but could be used for other underlying logics. We do not claim that it can outperform other possible approaches and a deeper study in this direction is needed (for instance, the examples from literature involve function

symbols, which would require reformulation into CL). Still, we addressed this problem to some extent and we analysed the set of proofs crafted by a human, using some automation and then formalized within Coq [5]. Our prover found several proofs shorter than the originals and with unsubstantial differences, but also some substantially different proofs (see Section 6.3).

In the context of proof length and short proofs, it is interesting to address *monotonicity* for proof lengths. We can formulate *monotonicity* as follows: for a given conjecture, if there is a proof of size $L$, then for each $L'$, such that $L' > L$, there is a proof of size $L'$. If monotonicity holds, then a simple binary search guarantees finding shortest proofs. However, monotonicity does not hold for our CL proofs. This is illustrated by the following example.

*Example 8* Consider the axiom $r(a) \vee r(b)$ and the conjecture $r(b) \vee r(a)$. It has the following proof of length 6:

It should be proved that $r(b) \vee r(a)$.

1. $r(a) \vee r(b)$ (by MP, using axiom ax1)
   2. Case $r(a)$:
      3. Proved by assumption! (by QEDas)
   4. Case $r(b)$:
      5. Proved by assumption! (by QEDas)
6. Proved by case split! (by QEDcs, by $r(a), r(b)$)

However, there is no proof of length 8. Indeed, one cannot apply anything more with respect to the given proof. One could apply excluded middle, but then more steps will be needed.

*All proofs.* In some contexts it can be interesting to have all proofs of length less than $n$ for a given conjecture. In the proposed approach, it is trivial to obtain all such proofs (or all proofs that meet some criterion) – it is just that all models of the generated constraints should be found. Having available all proofs of a theorem (within some logical framework) can be useful, for instance, in the context of education: "intelligent tutor systems" for mathematics education should be able to know all possible ways to solve a problem so they can help the students by anticipating their next steps (one such system, developed by Font et. al, based on a form of exhaustive search and implemented in Prolog, generates all proofs of geometry theorems using a custom set of axioms [28]).

In some cases, it can be also interesting just to count the number of proofs of length $\leq MaxL$ (for instance, in the educational context, the number of proofs could measure appropriateness or difficulty of an exercise). If SAT is the target problem, there are efficient techniques for counting all models [45].

## 6.2 Proof Hints

The hints within the presented approach are its unique feature: many other provers can accepts hints or axioms that *may* be used (possibly with a higher priority),

while, on the other hand, a hint for our prover means that it *must* be used. This feature reminds of GPS navigation: with GPS navigation, the user can ask for a road from a point $A$ to a point $B$, but he/she can also ask for a road that goes via certain *waypoints*. It is similar with proving: other provers can be asked for a proof for some goal from some given premises, but (only) the presented approach can be also given some *waypoints*, some concrete steps that the proof must contain. Many kinds of waypoints can be given as simple additional constraints (over the set of already introduced unknowns). These waypoints may be useful, for instance, in full reconstruction of proofs given only an outline (like proofs in textbooks). This could be a step towards a target discussed by Gowers and Hales [33]:

> One dream was to develop an automated assistant that would function at the level of a helpful graduate student. The senior mathematician would suggest the main lines of the proof, and the automated grad student would fill in the details.

A waypoint step that is provided can, of course, be totally irrelevant and superficial in the generated proof. However, if it is relevant and if, in addition, one asks for a shortest such proof, the proof is likely to heavily use the given steps/waypoints. Note that this approach is much more general than just splitting the problem into sub-problems: find a road from $A$ to $B$ and then a road from $B$ to $C$. In our context, the waypoints do not have to be ordered (one can ask for a proof using $X$ and $Y$ in no particular order), the waypoints can be vague, imposed only by partial constraints (e.g., "find a road going through some city" or "find a road going through some region", "find a road of length between $d_1$ and $d_2$" or, in our terms: "find a proof that uses some predicate symbol", or "find a proof using some lemma", without the way it is instantiated, etc). In a similar spirit, some axioms or the goal can be given imprecisely and the system could reconstruct them, along with a proof. Of course, one has to be careful and avoid trivial reconstructions (like an inconsistent set of axioms).

Our hints are different from "proof sketches" used in resolution based provers [41, 71], where "hints" are "important clauses, notable milestones toward a proof, because they already occurred in another proof of a stronger theory" and they provide patterns for giving higher priority to some sets of clauses. Using our analogy, proof hints in resolution provers are cities, such that if the algorithm happen to find a route through one of such cities, it will give a preference to this city in further search. Our hints could be a first step toward implementing a proof assistant allowing "proof sketches" in the sense of Wiedijk, i.e., partial information about the proof looking like an informal proof and whose gaps could be filled by the use of automation [74].

We have implemented support for the hints (for the URSA-based proving engine only). The implementation should be considered experimental. We slightly extended the language TPTP/fof to allow the specification of proof hints using a simple but still quite expressible semantics. Some features of this support and some kinds of hints (not all) are illustrated in the next example.

*Example 9* Consider the following simple conjecture given in the TPTP/fof format.

```
fof(ax1, axiom, (! [A,B] : (p(A,B) => r(B,A)))).
fof(ax2, axiom, (! [A,B] : (p(A,B) => q(B,A)))).
fof(ax3, axiom, (! [A,B] : (r(A,B) => r(B,A)))).
```

```
fof(ax4, axiom, (! [A,B] : (r(A,B) => p(B,A)))).
fof(ax5, axiom, (! [A,B] : (q(A,B) => q(B,A)))).
fof(ax6, axiom, (! [A,B] : (q(A,B) => p(B,A)))).
fof(ch, conjecture,(! [A,B] : (p(A,B) => p(B,A)))).
fof(hintname0, hint, r(?,?), _ , _).
```

One hint is given above as an additional, last line that brings the above specification out of the TPTP/fof format . In addition to the first two slots (in the style of the TPTP/fof format), there are three more slots: the first one provides a pattern for the atom that should appear in the proof, the second one specifies the ordinal number of the proof step constrained, and the third one provides a pattern for the (instantiated) axiom used. Each of them may be non-specified or partly non-specified. For instance, `fof(hintname0, hint, r(?,?), _, _)` imposes that a fact `r(?,?)` will be present in some step of the proof, while `fof(hintname0, hint, q(2,1), 5, _)` imposes that a fact `q(2,1)` will be present in the step 5 of the proof and the arguments will be the second and the first constant introduced. As another instance, `fof(hintname0, hint, _, 3, ax2(A,A))` imposes that the axiom `ax2` must be used in the step 3, in such a way that the first and the second universal variable are instantiated by the same constant.

The above conjecture can be proved either by using the predicate symbol `r` (and the axioms `ax1`, `ax3`, `ax4`), or the predicate symbol `q` (and the axioms `ax2`, `ax5`, `ax6`). We will illustrate how different hints can bring us to different proofs. The given hint imposes using the predicate symbol `r` while it does not impose any condition on its argument, or about the proof step where `r` is to be used. The generated shortest possible proof (without inlining) is as follows:

> Consider arbitrary $a$, $b$ such that: $p(a,b)$. It should be proved that $p(b,a)$.
>
> 1. $r(b,a)$ (by MP, from $p(a,b)$ using axiom ax1; instantiation: $A \mapsto a$, $B \mapsto b$)
> 2. $r(a,b)$ (by MP, from $r(b,a)$ using axiom ax3; instantiation: $A \mapsto b$, $B \mapsto a$)
> 3. $p(b,a)$ (by MP, from $r(a,b)$ using axiom ax4; instantiation: $A \mapsto a$, $B \mapsto b$)
> 4. Proved by assumption! (by QEDas)

If, for instance, the following hint was given instead:

```
fof(hintname0, hint, _, _, ax2(?,?)).
```

it imposes using the axiom `ax2`, and the generated shortest possible proof is as follows:

> Consider arbitrary $a$, $b$ such that: $p(a,b)$. It should be proved that $p(b,a)$.
>
> 1. $q(b,a)$ (by MP, from $p(a,b)$ using axiom ax2; instantiation: $A \mapsto a$, $B \mapsto b$)
> 2. $q(a,b)$ (by MP, from $q(b,a)$ using axiom ax5; instantiation: $A \mapsto b$, $B \mapsto a$)
> 3. $p(b,a)$ (by MP, from $q(a,b)$ using axiom ax6; instantiation: $A \mapsto a$, $B \mapsto b$)
> 4. Proved by assumption! (by QEDas)

Finally, if the following hints were given:

```
fof(hintname0, hint, r(?,?), 1, _).
fof(hintname1, hint, q(?,?), 3, _).
```

they impose using a fact `r(?,?)` in the step 1, and a fact `q(?,?)` in the step 3 of the proof (the assumption is the step 0). The generated shortest possible proof is as follows:

Consider arbitrary $a$, $b$ such that: $p(a,b)$. It should be proved that $p(b,a)$.

1. $r(b,a)$ (by MP, from $p(a,b)$ using axiom ax1; instantiation: $A \mapsto a$, $B \mapsto b$)
2. $q(b,a)$ (by MP, from $p(a,b)$ using axiom ax2; instantiation: $A \mapsto a$, $B \mapsto b$)
3. $q(a,b)$ (by MP, from $q(b,a)$ using axiom ax5; instantiation: $A \mapsto b$, $B \mapsto a$)
4. $p(b,a)$ (by MP, from $q(a,b)$ using axiom ax6; instantiation: $A \mapsto a$, $B \mapsto b$)
5. Proved by assumption! (by QEDas)

Obviously, in the latest proof, the step 1 is useless, but it is there since it was imposed by the hint. For this proof, the simplification procedure was turned off. If it was used, as normal, the first step would be eliminated as redundant.

### 6.3 Readable Proofs

There is a long history of rigorous proof systems that allow readable, natural proofs. Natural deduction, developed in 1920's, is one of the first and still most notable such systems [31]. Gentzen said: "I wanted to set up a formalism that comes as close as possible to actual reasoning". One of the key issues in readability of proofs are intuitive basic inference steps. But there is more in readability in proofs written by humans – readability is also a global property of the proof text. For example, human-style proofs or comprehensible proofs may omit many details but keep a substantial insight about why the theorem is valid. Recent approaches and challenges for automatically creating human readable proofs in geometry have been surveyed by Jiang *et. al.* [39] or by Nguyen [51].

Comprehensible proofs can be difficult to verify by a machine, but still can be understood by humans. Constructing but also verifying such proofs is an interesting research topic on its own (there is a recent overview of candidates for controlled natural languages for the communication of mathematics [33]), but it is not in the focus of this work. Still, we address a closely related task: generating rigorous proofs expressed in a strict formalism of CL with granularity similar to the granularity of natural language proofs. The CL proof system combines several natural deduction rules into one rule, thanks to which CL proofs often tend to be simpler and more natural than natural deduction proofs. Therefore, CL can serve well as a vehicle for readable proofs, and CL proofs can also suitably serve as a basis for further post-processing so they can be made even more human-style. In addition, the notion of a CL proof may be slightly modified into this direction, as discussed further.

Proofs expressed within CL can still be hard to read because the reader may be overwhelmed by uninteresting details. For example, in a natural language proof in a research level publication, no mathematician would state explicitly that:
a) "$ABCD$ is a rectangle because $ABCD$ is square" or

b) "$ABCD$ is a parallelogram because $BCDA$ is a parallelogram",
because such details would hide the important steps of the proofs. Keeping some inference steps implicit is hence crucial for the readability of the proofs. What should be implicit or not is difficult to say in general and depends on the context. For instance, at the primary school level, pupils may be encouraged or asked to state explicitly claims like a), but not claims like b). Having implicit proof steps but also mechanical proof checking of such proof steps, is useful. In geometry for example, there is a long history of incorrect proofs of Euclid's fifth postulate, that relied on an implicit assumption equivalent to the fifth postulate. For example, if one defines a parallelogram as a non-crossed quadrilateral such that two sides are parallel and of the same length, then the property b) stated above is equivalent to Euclid's fifth postulate (see [17] for more details about these issues).

As an extension of our basic approach, we use a variant of CL proofs, where proof steps allow implicit application of "simple" lemmas as described in Section 3.11. We have chosen to inline lemmas with at most one premise, but in natural language proofs sometimes proof steps involving several premises are also implicit, for example in geometry the transitivity of parallelism will usually not be stated explicitly. For Proposition 5 of Book I of Euclid's Elements, inlining leads to a quite readable proof, very similar to Pappus' proof, given in the following example (see appendix A for a full comparison of the proof of this proposition as generated by several provers).

*Example 10* Euclid Book I, Proposition 5: In isosceles triangles the angles at the base equal one another. Or, in formal terms:

$$\forall\ A, B, C\ (isosceles(A, B, C) \Rightarrow congA(A, B, C, A, C, B)\ )$$

Pappus' proof is as follows:[30]

> The two triangles $BAC$ and $CAB$ have two sides equal to two sides, namely side $BA$ of the first triangle equals side $CA$ of the second triangle, and side $AC$ of the first triangle equal to side $AB$ of the second, and the contained angles are equal, namely angle $BAC$ of the first triangle equals angle $CAB$ of the second, therefore, by proposition I.4, the corresponding parts of the two triangles are equal, in particular, the angle $B$ in the first triangle equals the angle $C$ of the second.

Proposition I.4, mentioned in the above proof, corresponds to the Side-Angle-Side theorem stating: if two triangles have two sides equal to two sides respectively, and have the angles contained by the equal sides equal, then they also have the base equal to the base, the triangle equals the triangle, and the remaining angles equal the remaining angles respectively.

The above, Pappus' proof assumes that the triangle is not degenerated, but the following (shortest) proof generated by Larus distinguishes the case of the degenerated triangle , showing that it is impossible (using implicitly a simple lemma that vertices of isosceles triangle are not colinear and the axiom `nnncolNegElim`: $\forall X, Y, Z\ \ col(X, Y, Z) \wedge \neg col(X, Y, Z) \Rightarrow \bot$). Larus' proof does not give details

[30] `https://mathcs.clarku.edu/~djoyce/java/elements/bookI/propI5.html`

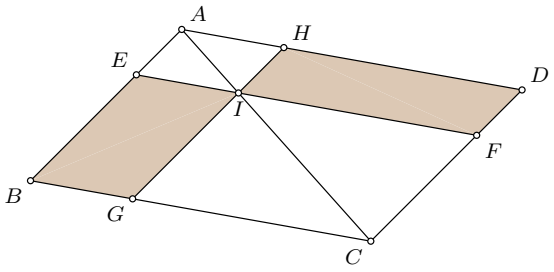


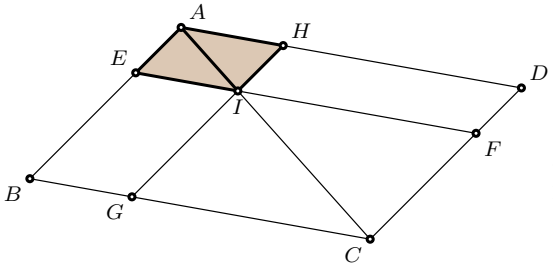

**Fig. 7** Euclid's Proposition 43: the areas of the two shaded parallelograms are equal.

about the fact that angle $BAC$ is congruent to angle $CAB$, because it is a lemma that is considered simple enough to be inlined. The premise stating that $abc$ is isosceles appears twice at step 6 because it corresponds to the two triangles ($BAC$ and $CAB$) that appear in proposition I.4 (Side-Angle-Side theorem, named here `proposition_04`).

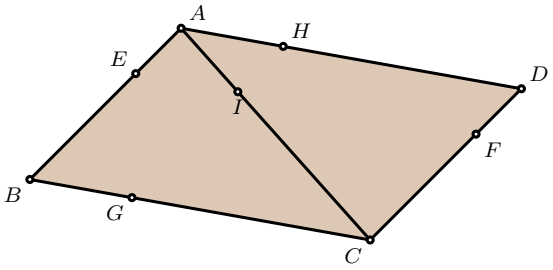
Consider arbitrary $a$, $b$, $c$ such that: $isosceles(a, b, c)$. It should be proved that $congA(a, b, c, a, c, b)$.

1. $col(c, a, b) \vee \neg col(c, a, b)$ (by MP, using axiom cn_col1b; instantiation: $A \mapsto c$, $B \mapsto a$, $C \mapsto b$)
   2. Case $col(c, a, b)$:
      3. $\perp$ (by MP, from $col(c, a, b)$, $isosceles(a, b, c)$ using axiom nnncolNegElim; instantiation: $A \mapsto a$, $B \mapsto b$, $C \mapsto c$)
      4. Contradiction! (by QEDefq)
   5. Case $\neg col(c, a, b)$:
      6. $congA(a, b, c, a, c, b)$ (by MP, from $isosceles(a, b, c)$, $isosceles(a, b, c)$, $\neg col(c, a, b)$ using axiom proposition_04; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto b$, $Xa \mapsto a$, $Xb \mapsto b$, $Xc \mapsto c$)
      7. Proved by assumption! (by QEDas)
8. Proved by case split! (by QEDcs, by $col(c, a, b), \neg col(c, a, b)$)

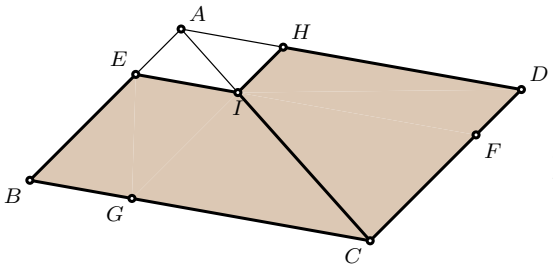
Note that the quantity of information which should be kept implicit is difficult to control, as illustrated on Euclid's Proposition 43 in the next example.



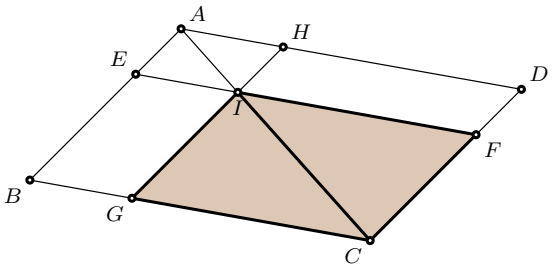

The triangles $AEI$ and $AHI$ have equal areas because they are congruent, because $AHIE$ is a parallelogram (implicit step).



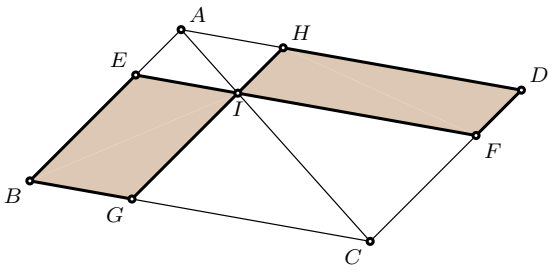

The triangles $ABC$ and $ADC$ have equal areas because they are congruent, because $ADCB$ is a parallelogram (implicit step).




The quadrilateral $HDCI$ and $EICB$ have equal areas which can be deduced from the two previous implicit steps using the axiom `cutoff1`.




The triangles $IGC$ and $IFC$ have equal areas because they are congruent, because $GIFC$ is a parallelogram (implicit step).




The conclusion can be deduced from the previous steps using the axiom `cutoff2`.

**Fig. 8** Euclid's Proposition 43: details of the proofs steps, three proof steps are implicit in the proof generated by Larus.

*Example 11* Euclid Book I, Proposition 43: In any parallelogram the complements of the parallelograms about the diameter equal one another (Fig. 7). Formally:[31]

$$\forall\, A, B, C, D, E, F, G, H, I$$
$$(pG(A,B,C,D) \wedge betS(A,H,D) \wedge betS(A,E,B) \wedge$$
$$betS(D,F,C) \wedge betS(B,G,C) \wedge betS(A,I,C) \wedge$$
$$pG(E,A,H,I) \wedge pG(G,I,F,C) \Rightarrow eF(I,G,B,E,D,F,I,H)\ )$$

The ratio between the size of a formal proof and size of the natural language proof, called the De Bruijn factor, is rarely below 1. With Larus, for this theorem, we get a surprising result: the (shortest) formal proof generated using inlined lemmas (if we omit the proofs of the inline lemmas) is shorter than both the original natural language proof and the original formal proof (30 steps).[32] The idea of the proof is depicted on Figure 8. The formal proof generated by Larus is just a two line proof, given here in the natural-language:

> Consider arbitrary $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$, $i$ such that: $pG(a,b,c,d)$, $betS(a,h,d)$, $betS(a,e,b)$, $betS(d,f,c)$, $betS(b,g,c)$, $betS(a,i,c)$, $pG(e,a,h,i)$, $pG(g,i,f,c)$. It should be proved that $eF(i,g,b,e,d,f,i,h)$.
>
> 1. $eF(c,i,h,d,b,e,i,c)$ (by MP, from $betS(a,i,c)$, $betS(a,e,b)$, $betS(a,h,d)$, $betS(a,i,c)$, $pG(e,a,h,i)$, $pG(a,b,c,d)$ using axiom axiom_cutoff1; instantiation: $A \mapsto c$, $B \mapsto i$, $C \mapsto a$, $D \mapsto h$, $E \mapsto d$, $Ca \mapsto b$, $Cb \mapsto e$, $Cc \mapsto a$, $Cd \mapsto i$, $Ce \mapsto c$)
> 2. $eF(h,d,f,i,e,b,g,i)$ (by MP, from $betS(d,f,c)$, $betS(b,g,c)$, $pG(g,i,f,c)$, $eF(c,i,h,d,b,e,i,c)$ using axiom axiom_cutoff2; instantiation: $A \mapsto h$, $B \mapsto d$, $C \mapsto f$, $D \mapsto c$, $E \mapsto i$, $Ca \mapsto e$, $Cb \mapsto b$, $Cc \mapsto g$, $Cd \mapsto c$, $Ce \mapsto i$)
> 3. Proved by assumption! (by QEDas)

The axiom `cutoff1` says that if we cut triangles of equal areas, with one end of the cut at a vertex and the other two on the adjacent sides, from triangles with equal areas, the resulting quadrilaterals have equal areas. The axiom `cutoff2` says that if we cut triangles with equal areas off of quadrilaterals with equal areas, with one end of the cut at a vertex and the other on a non-adjacent side, the results are quadrilaterals with equal areas.

In this case, the mechanism that generates inline lemmas by saturation of simple axioms, has gone quite far, maybe too far, because it generated about 400 lemmas, and then most of the proof is implicit! It means that the proof of Proposition 43 is mainly a composition of lemmas with a single premise.

In the first step of the proof, the fact that the triangles $AEI$ and $AHI$ have equal areas is deduced implicitly from the fact that $EAHI$ is a parallelogram using Euclid's Proposition 34 (which states that the diagonal of parallelogram divides it into two congruent triangles) and a lemma stating that congruent triangles have equal areas and many technical lemmas about preservation of equality of area by

[31] $pG(A,B,C,D)$ stands for "$ABCD$ is a parallelogram", $betS(A,B,C)$ stands for "$B$ is between $A$ and $C$", and $eF(I,G,B,E,D,F,I,H)$ stands for "$IGBE$ and $DFIH$ are figures with equal areas".

[32] `https://github.com/GeoCoq/GeoCoq/blob/master/Elements/OriginalProofs/proposition_43.v`

permutation of the points. Similarly it is implicitly proved that the triangles $ABC$ and $ADC$ have equal areas.

We can further modify the look of our computer generated proofs so they are more like human-created proofs. For example, we can replace case splits by *reductio ad absurdum* in some cases, we can replace predicate symbols and logical connectives by some natural language wording, generate nicer looking names, etc. We leave this for future work, as in this paper we focus on the granularity of the proofs steps and the structure of the proof.

### 6.4 Reconstruction of Proofs within ITP Systems

Integration of automatic theorem provers within interactive provers and exchange of proofs is crucial for development of interactive theorem proving but it poses several difficulties. The main ones come from differences between the underlying logical frameworks of the ITP and the ATPs systems (higher-order logics vs first-order logic) and the absence of standards for representing proofs in both worlds. There are several lines of research trying to address these problems. The TSTP format allows to represent proofs as a list of steps, and the language does not specify what are the acceptable proof steps [67]. The Logipedia project provides automatic transformations between different logical frameworks [25]. Tools called "hammers" have been developed for Isabelle/HOL [10, 11, 13, 14], HOL4 [70], Mizar [40] and Coq [22, 23]. In many hammer systems, the ATP is used only to find relevant lemmas/axioms and the actual proof argument by the ATP system is rarely used. Instead, an automatic theorem prover integrated in the ITP system is ran, using lemmas/axioms filtered out, for constructing an acceptable formal proof. Therefore, for examples where the list of relevant lemmas/axioms is already known, the hammer approach using state-of-the-art FOL provers will not provide any useful information and will not be more efficient than solely automatic tactics integrated inside ITPs. In some hammer systems, external solvers are used to produce certificates based on which the proof objects are constructed within the ITP.

If one wants to obtain a readable proof within an ITP system, then the difficulty of reconstructing a proof from an ATP argument is even higher, because state-of-the-art ATPs usually rely on clausification, Skolemization and proof by refutation. Blanchette has proposed an algorithm to reconstruct a forward chaining Isar proof from the resolution proof [12]. The approach presented here has the advantage of allowing the trivial generation of machine verifiable proofs which are fairly readable: they do not rely on clausification nor proof by refutation, they are in the forward reasoning style and the proof steps are higher-level than natural deduction proofs. With the support of proof hints as described in Section 6.2, by extracting automatically hints from formal or informal proofs, this approach could also serve as a base for translators from one proof language to another.

### 6.5 Classical vs. Intuitionistic Logic

While, for instance, in the resolution method, classical logic and reasoning are deeply built-in, in the presented approach and when using the prover, one can

easily choose whether to use excluded middle or not (and, hence, choose between classical and intuitionistic setting), it is just a matter of adding axioms. Also, as said in Section 2, the additional ¬Intro rule could be supported. On the other hand, this feature (of flexible control of fundamental axioms) increases costs and influences efficiency of the presented approach.

### 6.6 Proof complexity

The central problems in proof complexity are: what is the size of the smallest proof of a theorem $F$ in some logical system and how difficult is it to construct such a smallest proof [2, 49]. These problems are relevant for the work presented in this paper, but not of a concrete, practical value for implementing the prover. Namely, answers to the given questions are typically given in terms of bounds too loose to be useful for limiting and guiding our proof-searching process.

## 7 Related Work

In the previous sections, we mentioned some related research, for instance, in the context of short or readable or machine verifiable proofs. In this section we discuss research and systems most closely related to the one presented, especially in terms of proof encoding and of using SAT/SMT in automated theorem proving.

*Coherent Logic provers.* There are several automated theorem provers for CL [6, 7, 8, 37, 66]. Some of them use advanced methods for efficient rule-matching [34] or back-jumping and lemma-learning inspired by CDCL SAT solving [52, 55, 56]. They all use some form of forward chaining, in contrast to the prover presented here. Only some of them produce machine verifiable proofs [7] or both machine verifiable and human readable proofs [52, 66].

*Automated provers using SAT/SMT solvers.* There are approaches in which SAT and SMT are used in first-order theorem proving. For instance, in showing inconsistency of clause sets, Schultz uses saturation and SAT solving — ground facts are represented by propositional variables and there is a periodic check of propositional consistency [63]. In Vampire, the AVATAR architecture also tightly integrates SAT/SMT solvers [73]. These approaches are completely different from ours as SAT formulae are used for modelling the object level (formulae of underlying theory), while in ours, SAT/SMT formulae model the meta level, the level of proofs.

SAT solving is used also in higher-order theorem proving. For instance, there are complete theorem proving procedures, implemented in a prover Satallax, for higher-order logic based on a complete, cut-free, ground refutation calculus that delegates different tasks to a SAT solver [19]. Like in the case of first-order logic, SAT formulae do not model proofs, but again formulae, in contrast to our approach.

*Proof encoding.* The idea of encoding a proof by numbers dates back at least to Gödel's enumeration of proofs by which (in the context of his famous theorems) he demonstrated that deduction operations can be described in terms of numbers and arithmetical operations [64]. However, apart from such theoretical usage, there are hardly any practical applications of proof encoding in automated or interactive theorem proving . We are aware only of the following ones and they are the most closely related work.

There is a work on encoding propositional resolution proofs in propositional logic [62]. This is done trying to show that local search methods (suitable for solving satisfiable SAT instances) can be applied to unsatisfiable SAT instances. Our approach is also an alternative way of transforming unsatisfiability checking (using refutation) into satisfiability checking (i.e., checking existence of a proof).

Rigid tableau proofs (a rigid tableau is a tableau in which multiple instances of a clause appearing in the tableau are identical copies of the clause appearing in the given formula) can be encoded as a SAT problem, as in a system ChewTPTP-SAT [24]. There are some problem instances that can be proved using this approach, and not by other provers, but the conclusion is that this approach is still not competitive to leading FOL provers. That work extends to encoding of the existence of a rigid first-order connection tableau to SMT (datatypes and arithmetic) [15, 48]. It was shown that SMT-based encoding is more efficient than SAT-based encoding, but there is no experimental comparison to other kinds of provers. Machine verifiable or readable proofs are not considered. The prover ChewTPTP-SMT based on this approach was used for our experimental comparison given in Section 5.

Alrabbaa et.al. addressed the problem of finding small proofs in description logic (DL) [1]. They encode DL proofs as labelled, directed hypergraphs, where each hyper-edge corresponds to a single derivation step, and they study the complexity of the problem of finding the smallest DAG-shaped and tree shaped DL proofs.

## 8 Future Work

We are planning to implement export of proofs to Isabelle and other interactive theorem provers.

The first implementation of the presented approach shows proving performance below the state-of-the-art provers such as Vampire. We think there is a room for improvements of efficiency in the following directions:

- improving the encoding used (e.g., by using some form of incremental encoding);
- improving the solving process (e.g., by using some other SAT/SMT solvers, or by instructing SAT/SMT solvers to take into account some specifics of input instances);
- improving the communication between the prover and the external solvers.

Apart from the above (still rather vague directions), we consider several ways for improving the prover, discussed in the rest of this section.

### 8.1 Normalization of CL Proofs and Application to Automated Theorem Proving

Normalisation of proofs has been present for decades as one of central topics in proof theory. For instance, permutability of inference rules has been studied from a theoretical point of view since a long time, by Kleene in the 50's [42] and more recently by Dyckhoff *et. al.* [27], Guenot [32], Lutovac and Harland [46], etc. However, to our knowledge, it has been hardly used in automated theorem provers during the search process. Namely, the provers do not look for a proof, but for a formula and they usually introduce many superficial formulae along the way. During that process, it is not known which of them are really needed and how to use some normalisation criterion over them on the fly. In our approach, however, we can impose restrictions on a sought proof directly (no superficial facts) and we can also impose some normalisation criterion as an additional constraint. Actually, in the current version of the system, we already use some normalisation criteria: a case split can occur only at the end of a proof, the first case step occurs immediately after the disjunction has been derived, etc. By imposing restrictions on the form of a proof, some parts of the search space can be eliminated early which could facilitate the process. But, on the other hand, the number of possible models (i.e., proofs) will be smaller which could make search harder. We are planning to explore these ideas further and we hope that our approach could lead to new practical applications of normalisation results coming from proof theory.

### 8.2 Paralellisation

Let us assume one wants to prove a theorem for which a shortest proof has $k$ steps, the number not known in advance. When searching for a proof of the theorem, one must provide a concrete, maximal proof length. If the chosen length is small, the chances are good for a quick response, but if it is lower than $k$, the proof will not be found (also, proving that there is no proof of some given length sometimes takes more time than finding a proof of a larger length). On the other hand, if the chosen length is big, it could be likely that there is a proof of smaller length, but the proving process can be slow and maybe over the given time limit. Therefore, it can be beneficial if different maximal proof lengths are considered in parallel. Some parts of the encoding can be shared, but the proving process can be completely independent and can be simply performed in parallel. We will work on this simple paralellisation and look for other potentially beneficial forms of paralellisation.

### 8.3 Using Other Kinds of Provers

We observed there are theorems that the presented approach cannot prove in some reasonably short time, while they can be proved by our simple forward-chaining based prover in a negligible time. The explanation is, we believe, that these theorems admit proofs in which in many steps one can proceed in just a very few ways – many steps are almost uniquely determined. For such theorems there are relatively long proofs (e.g., more than 200 proof steps), but substantially very easy for the forward-chaining approach. This observation leads to the following possible directions for future work:

- One can combine the presented approach with other kinds of provers, as they can be best performing on different sets of theorems. Combining different provers can be done in a well-developed portfolio approach [53, 76]. An additional challenge (probably well-suited to machine learning) would be to detect which provers are better suited for which theorems. Actually, within a portfolio, one can use not only different provers, but also different configurations of Larus (given by its parameters).
- One can use the forward-chaining approach along the proving process for shaping additional constraints that could help the prover based on constraint solving.
- One can try to help the SAT/SMT solvers in solving the constraint corresponding to a proof. Choosing decision variables is one of critical operations in SAT/SMT solving. In some situations, the solver may choose a decision variable between one corresponding to the axiom applied in some proof step and one corresponding to the instantiation. We expect that often it would be beneficial to choose the former. Such guiding information could be possibly also gathered by a controlled application of forward-chaining. This would require modifying both the communication between the CL prover and SAT/SMT solvers, and the SAT/SMT solvers themselves.

### 8.4 Applications in other domains

We are planning to use the presented approach to some other logics (not only CL), especially those without good available decision or semi-decision procedures. Also, we will try to apply the presented approach to complex symbolic computations that can be presented in a similar manner to proofs, such as computing Gröbner bases or solving the Rubik's cube.

## 9 Conclusions

We presented an approach for automated theorem proving and for automated construction of proof objects based on constraint solving (i.e., a new paradigm that we can call "theorem proving as constraint solving"). That idea is not surprising or completely unexpected (especially given many other kinds of problems tackled by constraint solving), but still it has almost no mentions in the literature. One may only speculate why it is so. Maybe it was expected that this approach would be unfeasible and without practical usability.

Our work shows that, for an interesting and expressible fragment of FOL, the approach is feasible and, moreover, that it has some unique merits and features such as: producing all proofs; producing shortest proofs; producing natural-looking proofs expressed in a forward reasoning manner – both in a natural language form and in a form verifiable by interactive provers and readable at the same time; producing even more compact proofs using inlining, using hints about a proof to be found, etc.

We find that the proposed approach and the presented prover can be useful especially in the context of formalisation of mathematical theories and as a support in interactive theorem proving. Our prototype implementation gives reasonable

performance especially when compared to provers producing proofs verifiable by a proof assistant.

In the area of automated theorem proving without construction of verifiable proofs, the presented approach faces a strong competition in FOL provers. They have been evolving for more than half a century, they often combine several underlying strategies or provers using portfolio approach and it is hard to beat their performances . However, beating performance of the leading FOL provers is not one of the targets of our approach – it is rather complementary with the FOL provers and has different main merits. In addition, there is a number of other logical frameworks where solvers and reasoners are not so well developed and the approach presented here could easily lead to practically usable automated theorem provers – one needs only a precise notion of proof or a solution (which could take only a few hundreds lines, as shown for CL) and encoding to a sequence of natural numbers. An ultimate target would be a generic framework that automatically prove theorems, given only a precise notion of a proof in the underlying logic and theory. The approach could be applied to virtually any underlying logic and theory with a precise syntactical notion of proof. However, although it would be possible, it would be challenging to encode proof steps involving non-trivial features such as unification, or Fourier/Motzkin elimination [35], or elimination steps used in the area method [38]. And not only the encoding, but also the efficiency of such proving procedure would be also questionable. On the other hand, the approach is very suitable for syntactically simple theories and proofs, such are propositional calculi and coherent logic. Indeed, the current version of the presented approach, applied here to coherent logic, enjoys a suitable simplicity: it does not support function symbols. The presented encoding has to be significantly extended to make function symbols usable. Alternatively, problems involving function symbols could be encoded using predicate symbols only, but that would make generated proofs longer and less readable.

In summary, we believe that "theorem proving as constraint solving" work presented in this paper is a first encouraging attempt, leaving room for further improvements and a range of applications.

**Acknowledgements** We are grateful to the anonymous reviewers who gave us a very useful feedback and number of suggestions that significant improved the earlier version of this paper.

## References

1. Christian Alrabbaa, Franz Baader, Stefan Borgwardt, Patrick Koopmann, and Alisa Kovtunova. Finding Small Proofs for Description Logic Entailments: Theory and Practice. In *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 32–5, 2020.
2. Albert Atserias and Moritz Müller. Automating Resolution is NP-Hard. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 498–509. IEEE Computer Society, 2019.
3. Jeremy Avigad. Automated reasoning for the working mathematician. In *Frontiers of Combining Systems (FroCoS)*, London, 2019. Invited Talk.

4. Jeremy Avigad, Edward Dean, and John Mumma. A Formal System for Euclid's Elements. *The Review of Symbolic Logic*, 2:700–768, 2009.
5. Michael Beeson, Julien Narboux, and Freek Wiedijk. Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, 85(2-4):213–257, 2019. Publisher: Springer.
6. Marc Bezem and Thierry Coquand. Newman's Lemma – a Case Study in Proof Automation and Geometric Logic. *Current trends in Theoretical Computer Science*, 2:267–282, 2004.
7. Marc Bezem and Thierry Coquand. Automating Coherent Logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2005*, volume 3835 of *Lecture Notes in Computer Science*, pages 246–260. Springer-Verlag, 2005.
8. Marc Bezem and Dimitri Hendriks. On the Mechanization of the Proof of Hessenberg's Theorem in Coherent Logic. *Journal of Automated Reasoning*, 40(1):61–85, 2008.
9. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
10. Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, Vol 9:101–148 Pages, January 2016. Artwork Size: 101-148 Pages Publisher: Alma Mater Studiorum - University of Bologna.
11. Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. Automatic Proof and Disproof in Isabelle/HOL. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *Frontiers of Combining Systems, 8th International Symposium, Proceedings*, volume 6989 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2011.
12. Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. Semi-intelligible Isar Proofs from Machine-Generated Proofs. *Journal of Automated Reasoning*, 56(2):155–200, February 2016.
13. Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT Solvers. *Journal of Automated Reasoning*, 51(1):109–128, 2013.
14. Jasmin Christian Blanchette, Andrei Popescu, Daniel Wand, and Christoph Weidenbach. More SPASS with Isabelle. In *International Conference on Interactive Theorem Proving*, pages 345–360. Springer, 2012.
15. Jeremy Bongio, Cyrus Katrak, Hai Lin, Christopher Lynch, and Ralph Eric McGregor. Encoding First Order Proofs in SMT. *Electron. Notes Theor. Comput. Sci.*, 198(2):71–84, 2008.
16. Richard Bonichon, David Delahaye, and Damien Doligez. Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 4790, pages 151–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.
17. Pierre Boutry, Charly Gries, Julien Narboux, and Pascal Schreck. Parallel postulates and continuity axioms: a mechanized study in intuitionistic logic

using Coq. *Journal of Automated Reasoning*, page 68, 2017.
18. Pierre Boutry, Julien Narboux, and Pascal Schreck. A reflexive tactic for automated generation of proofs of incidence to an affine variety. October 2015.
19. Chad E. Brown. Reducing Higher-Order Theorem Proving to a Sequence of SAT Problems. *Journal of Automated Reasoning*, 51(1):57–77, June 2013.
20. Sascha Böhme and Tjark Weber. Fast LCF-Style Proof Reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2010.
21. Evelyne Contejean and Pierre Corbineau. Reflecting Proofs in First-Order Logic with Equality. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and Robert Nieuwenhuis, editors, *Automated Deduction – CADE-20*, volume 3632, pages 7–22. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. Series Title: Lecture Notes in Computer Science.
22. Łukasz Czajka. Practical Proof Search for Coq by Type Inhabitation. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020, Automated Reasoning*, volume 12167, pages 28–57. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.
23. Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for Dependent Type Theory. *Journal of Automated Reasoning*, 61(1-4):423–453, June 2018.
24. Todd Deshane, Wenjin Hu, Patty Jablonski, Hai Lin, Christopher Lynch, and Ralph Eric McGregor. Encoding First Order Proofs in SAT. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2007.
25. Gilles Dowek and François Thiré. Logipedia: a multi-system encyclopedia of formal proofs. In *Workshop on Large Mathematical Libraries*, Prague, 2019.
26. Roy Dyckhoff and Sara Negri. Geometrization of first-order logic. *The Bulletin of Symbolic Logic*, 21:123–163, 2015.
27. Roy Dyckhoff and Luís Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212(1-2):141–155, February 1999.
28. Ludovic Font, Sébastien Cyr, Philippe R. Richard, and Michel Gagnon. Automating the Generation of High School Geometry Proofs using Prolog in an Educational Context. In *Proceedings 8th International Workshop on Theorem Proving Components for Educational Software, ThEdu@CADE*, Natal, Brazil, 2019.
29. M. Ganesalingam and W. T. Gowers. A fully automatic problem solver with human-style output. *CoRR*, abs/1309.4501, 2013.
30. Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp : A Conflict-Driven Answer Set Solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265.

Springer, 2007.
31. Gerhard Gentzen. Untersuchungen über das logische Schliessen, I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in "The Collected Papers of Gerhard Gentzen", North-Holland Publ.Co, 1969.
32. Nicolas Guenot. Concurrency and Permutability in the Sequent Calculus. In M. Parigot and L. Straßburger, editors, *Structures and Deduction (ESSLLI'09 workshop)*, pages 39–52, 2009.
33. Thomas Hales. An argument for controlled natural languages in mathematics, 2019.
34. Bjarne Holen, D. Hovland, and M. Giese. Efficient Rule-Matching for Automated Coherent Logic. In *NIK-2013 proceedings*, 2012.
35. Jean-Louis Imbert. Fourier's Elimination: Which to Choose? In *Principles and Practice of Constraint Programming*, pages 117–129, 1993.
36. Predrag Janičić. URSA: A System for Uniform Reduction to SAT. *Logical Methods in Computer Science*, 8(3):30, September 2012.
37. Predrag Janičić and Stevan Kordić. EUCLID — the Geometry Theorem Prover. *FILOMAT*, 9(3):723–732, 1995.
38. Predrag Janičić, Julien Narboux, and Pedro Quaresma. The Area Method : a Recapitulation. *Journal of Automated Reasoning*, 48(4):489–532, 2012.
39. Jianguo Jiang and Jingzhong Zhang. A review and prospect of readable machine proofs for geometry theorems. *Journal of Systems Science and Complexity*, 25(4):802–820, 2012.
40. Cezary Kaliszyk and Josef Urban. HOL (y) Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015. Publisher: Springer.
41. Michael Kinyon. Proof simplification and automated theorem proving. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 377(2140):20180034, March 2019.
42. Stephen Cole Kleene. Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10:1–26, 1952.
43. Konstantin Korovin. iProver – An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195, pages 292–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.
44. Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.
45. Jean-Marie Lagniez and Pierre Marquis. On Preprocessing Techniques and Their Impact on Propositional Model Counting. *Journal of Automated Reasoning*, 58(4):413–481, April 2017.
46. Tatjana Lutovac and James Harland. A contribution to automated-oriented reasoning about permutability of sequent calculi rules. *Computer Science and Information Systems*, 10(3):1185–1210, 2013.
47. Saunders MacLane and Ieke Moerdijk. *Sheaves in geometry and logic: a first introduction to topos theory*. Springer-Verlag, 1992.

48. Ralph Eric McGregor. *Automated Theorem Proving Using SAT*. PhD Thesis, Clarkson University, 2011. Publication Title: Electron. Notes Theor. Comput. Sci.
49. Ian Mertz, Toniann Pitassi, and Yuanhao Wei. Short Proofs Are Hard to Find. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 84:1–84:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
50. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.
51. H. D. Nguyen, Diem Nguyen, and V. T. Pham. Design an intelligent problem solver in solid geometry based on knowledge model about relations. In *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*, pages 150–155, 2016.
52. Mladen Nikolić and Predrag Janičić. CDCL-Based Abstract State Transition System for Coherent Logic. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*, volume 7362 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2012.
53. Mladen Nikolić, Filip Marić, and Predrag Janičić. Simple algorithm portfolio for SAT. *Artificial Intelligence Review*, 2012. to appear.
54. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle HOL: a Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
55. Hans de Nivelle. Subsumption Algorithms for Three-Valued Geometric Resolution. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2016.
56. Hans de Nivelle and Jia Meng. Geometric Resolution: A Proof Procedure Based on Finite Model Search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2006.
57. Jens Otten. nanoCoP: Natural Non-clausal Theorem Proving. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4924–4928, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization.
58. Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *Journal of Symbolic Computation*, 36(1-2):139–161, July 2003.

59. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
60. Lawrence C. Paulson. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. In Renate A. Schmidt, Stephan Schulz, and Boris Konev, editors, *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*, volume 9 of *EPiC Series in Computing*, pages 1–10. EasyChair, 2010.
61. Andrew Polonsky. *Proofs, Types and Lambda Calculus*. PhD thesis, University of Bergen, 2011.
62. Steven David Prestwich and Inês Lynce. Local Search for Unsatisfiability. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2006.
63. Stephan Schulz. Light-weight integration of SAT solving into first-order reasoners–first experiments. *Vampire 2017 - Proceedings of the 4th Vampire Workshop*, 53:9–19, 2017.
64. Peter Smith. *An Introduction to Gödel's Theorems*. Cambridge University Press, 2013.
65. Sana Stojanović, Julien Narboux, Marc Bezem, and Predrag Janičić. A Vernacular for Coherent Logic. In StephenM. Watt, JamesH. Davenport, AlanP. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics*, volume 8543 of *Lecture Notes in Computer Science*, pages 388–403. Springer International Publishing, 2014.
66. Sana Stojanović, Vesna Pavlović, and Predrag Janičić. A Coherent Logic Based Geometry Theorem Prover Capable of Producing Formal and Readable Proofs. In *Automated Deduction in Geometry*, volume 6877 of *Lecture Notes in Computer Science*, pages 201–220. Springer, 2011.
67. Geoff Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning – LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2010.
68. The Coq development team. The Coq proof assistant. 2020.
69. Ruediger Thiele and Larry Wos. Hilbert's Twenty-Fourth Problem. *Journal of Automated Reasoning*, 29(1):67–89, 2002.
70. Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *Journal of Automated Reasoning*, 50(2):229–241, 2013. Publisher: Springer.
71. Robert Veroff. Finding Shortest Proofs: An Application of Linked Inference Rules. *Journal of Automated Reasoning*, 27(2):123–139, 2001.
72. Steven Vickers. Geometric Logic in Computer Science. In *Theory and Formal Methods*, Workshops in Computing, pages 37–54. Springer, 1993.
73. Andrei Voronkov. AVATAR: The Architecture for First-Order Theorem Provers. In *Computer Aided Verification*, pages 696–710. Springer, Cham, July 2014.
74. Freek Wiedijk. Formal proof sketches. In *International Workshop on Types for Proofs and Programs*, pages 378–393. Springer, 2003.

75. Larry Wos. Automating the Search for Elegant Proofs. *Journal of Automated Reasoning*, 21(2):135–175, 1998.
76. Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

## A Example proofs of Proposition 5

In this section, we list the proofs of Euclid Book 1, Proposition 5, generated by different theorem provers.

### A.1 Leancop

```
_____________________________________________________
Explanations for the proof presented below:
- to solve unsatisfiable problems they are negated
- equality axioms are added if required
- terms and variables are represented by Prolog terms
  and Prolog variables, negation is represented by -
- I^[t1,..,tn] represents the atom P_I(t1,..,tn)
  or the Skolem term f_I(t1,t2,..,tn) introduced
  during the clausal form translation
- the substitution [[X1,..,Xn],[t1,..,tn]] represents
  the assignments X1:=t1, .., Xn:=tn

Proof:
------

Translation into (disjunctive) clausal form:
 (1)  [-(isosceles(1 ^ [], 2 ^ [], 3 ^ []))]
 (2)  [congA(1 ^ [], 2 ^ [], 3 ^ [], 1 ^ [], 3 ^ [], 2 ^ [])]
 (3)  [isosceles(_4916, _4976, _5035), -(triangle(_4916, _4976, _5035))]
 (4)  [isosceles(_4916, _4976, _5035), -(cong(_4916, _4976, _4916, _5035))]
 (5)  [-(isosceles(_5481, _5541, _5600)), triangle(_5481, _5541, _5600), cong
      (_5481, _5541, _5481, _5600)]
 (6)  [cong(_6105, _6163, _6046, _6220), -(cong(_6046, _6220, _6105, _6163))]
 (7)  [triangle(_6597, _6651, _6704), col(_6597, _6651, _6704)]
 (8)  [-(col(_7006, _7060, _7113)), -(triangle(_7006, _7060, _7113))]
 (9)  [col(_7415, _7495, _7574), -(col(_7495, _7415, _7574))]
 (10)  [col(_7415, _7495, _7574), -(col(_7495, _7574, _7415))]
 (11)  [col(_7415, _7495, _7574), -(col(_7574, _7415, _7495))]
 (12)  [col(_7415, _7495, _7574), -(col(_7415, _7574, _7495))]
 (13)  [col(_7415, _7495, _7574), -(col(_7574, _7495, _7415))]
 (14)  [-(col(_8342, _8399, _8455)), -(congA(_8342, _8399, _8455, _8455,
      _8399, _8342))]
 (15)  [-(cong(_8987, _9093, _9302, _9405)), cong(_8880, _8987, _9198, _9302)
      , cong(_8880, _9093, _9198, _9405), congA(_8987, _8880, _9093, _9302,
      _9198, _9405)]
 (16)  [-(congA(_8880, _8987, _9093, _9198, _9302, _9405)), cong(_8880, _8987
      , _9198, _9302), cong(_8880, _9093, _9198, _9405), congA(_8987, _8880,
      _9093, _9302, _9198, _9405)]
 (17)  [-(congA(_8880, _9093, _8987, _9198, _9405, _9302)), cong(_8880, _8987
      , _9198, _9302), cong(_8880, _9093, _9198, _9405), congA(_8987, _8880,
      _9093, _9302, _9198, _9405)]

We prove that the given clauses are valid, i.e. for
a given substitution they evaluate to true for all
interpretations. The proof is by contradiction:
Assume there is an interpretation so that the given
clauses evaluate to false. Then in each clause there
has to be at least one literal that is false.

 Then clause (7) under the substitution [[_6704, _6651, _6597], [3 ^ [], 2 ^
      [], 1 ^ []]]
 is false if at least one of the following is false:
 [triangle(1 ^ [], 2 ^ [], 3 ^ []), col(1 ^ [], 2 ^ [], 3 ^ [])]
1 Assume triangle(1 ^ [], 2 ^ [], 3 ^ []) is false.
  Then clause (3) under the substitution [[_5035, _4976, _4916], [3 ^ [], 2 ^
       [], 1 ^ []]]
  is false if at least one of the following is false:
  [isosceles(1 ^ [], 2 ^ [], 3 ^ [])]
1.1 Assume isosceles(1 ^ [], 2 ^ [], 3 ^ []) is false.
    Then clause (1) is true.
2 Assume col(1 ^ [], 2 ^ [], 3 ^ []) is false.
  Then clause (9) under the substitution [[_7574, _7415, _7495], [3 ^ [], 2 ^
       [], 1 ^ []]]
  is false if at least one of the following is false:
  [col(2 ^ [], 1 ^ [], 3 ^ [])]
2.1 Assume col(2 ^ [], 1 ^ [], 3 ^ []) is false.
    Then clause (14) under the substitution [[_8455, _8399, _8342], [3 ^ [],
         1 ^ [], 2 ^ []]]
    is false if at least one of the following is false:
    [-(congA(2 ^ [], 1 ^ [], 3 ^ [], 3 ^ [], 1 ^ [], 2 ^ []))]
2.1.1 Assume -(congA(2 ^ [], 1 ^ [], 3 ^ [], 3 ^ [], 1 ^ [], 2 ^ [])) is
     false.
      Then clause (16) under the substitution [[_9405, _9198, _9302, _9093,
           _8880, _8987], [2 ^ [], 1 ^ [], 3 ^ [], 3 ^ [], 1 ^ [], 2 ^ []]]
      is false if at least one of the following is false:
```

```
      [-(congA(1 ^ [], 2 ^ [], 3 ^ [], 1 ^ [], 3 ^ [], 2 ^ [])), cong(1 ^ [],
            2 ^ [], 1 ^ [], 3 ^ []), cong(1 ^ [], 3 ^ [], 1 ^ [], 2 ^ [])]
2.1.1.1 Assume -(congA(1 ^ [], 2 ^ [], 3 ^ [], 1 ^ [], 3 ^ [], 2 ^ [])) is
     false.
        Then clause (2) is true.
2.1.1.2 Assume cong(1 ^ [], 2 ^ [], 1 ^ [], 3 ^ []) is false.
        Then clause (4) under the substitution [[_5035, _4976, _4916], [3 ^
             [], 2 ^ [], 1 ^ []]]
        is false if at least one of the following is false:
        [isosceles(1 ^ [], 2 ^ [], 3 ^ [])]
2.1.1.2.1 Assume isosceles(1 ^ [], 2 ^ [], 3 ^ []) is false.
          Then clause (1) is true.
2.1.1.3 Assume cong(1 ^ [], 3 ^ [], 1 ^ [], 2 ^ []) is false.
        Then clause (6) under the substitution [[_6163, _6105, _6220, _6046],
             [2 ^ [], 1 ^ [], 3 ^ [], 1 ^ []]]
        is false if at least one of the following is false:
        [cong(1 ^ [], 2 ^ [], 1 ^ [], 3 ^ [])]
2.1.1.3.1 Assume cong(1 ^ [], 2 ^ [], 1 ^ [], 3 ^ []) is false.
          This assumption has been refuted in 2.1.1.2.

Therefore there is no interpretation that makes all
given clauses simultaneously false. Hence the given
clauses are valid.
                                          q.e.d.
```

## A.2 Nanocop

```
tptp-problems/euclid-native-eq/015_proposition_05.p is a Theorem
Start of proof for tptp-problems/euclid-native-eq/015_proposition_05.p
[(26 ^ 0) ^ [83 ^ [], 82 ^ [], 81 ^ []] : [triangle(81 ^ [], 82 ^ [], 83 ^
     []),
col(81 ^ [], 82 ^ [], 83 ^ [])], [(2 ^ 1) ^ [83 ^ [], 82 ^ [], 81 ^ []] :
[-(triangle(81 ^ [], 82 ^ [], 83 ^ [])), 5 ^ 1 : [(6 ^ 1) ^ [] : []],
isosceles(81 ^ [], 82 ^ [], 83 ^ [])], [(85 ^ 2) ^ [] : [-(isosceles(81 ^ [],
      82 ^ [], 83 ^ []))]]],
[(38 ^ 1) ^ [83 ^ [], 81 ^ [], 82 ^ []] : [-(col(81 ^ [], 82 ^ [], 83 ^ [])),
     41 ^ 1 :
[(42 ^ 1) ^ [] : []], col(82 ^ [], 81 ^ [], 83 ^ [])], [(52 ^ 2) ^ [83 ^ [],
     81 ^ [],
82 ^ []] : [-(col(82 ^ [], 81 ^ [], 83 ^ [])), -(congA(82 ^ [], 81 ^ [], 83 ^
      [], 83 ^
[], 81 ^ [], 82 ^ []))], [(58 ^ 3) ^ [82 ^ [], 83 ^ [], 81 ^ [], 83 ^ [], 82
     ^ [],
81 ^ []] : [congA(82 ^ [], 81 ^ [], 83 ^ [], 83 ^ [], 81 ^ [], 82 ^ []), 69 ^
      3 :
[(70 ^ 3) ^ [] : [-(cong(82 ^ [], 83 ^ [], 83 ^ [], 82 ^ []))], (72 ^ 3) ^ []
     :
[-(congA(81 ^ [], 82 ^ [], 83 ^ [], 81 ^ [], 83 ^ [], 82 ^ []))], (74 ^ 3) ^
[] : [-(congA(81 ^ [], 83 ^ [], 82 ^ [], 81 ^ [], 82 ^ [], 83 ^ []))]], cong
     (81 ^ [],
82 ^ [], 81 ^ [], 83 ^ []), cong(81 ^ [], 83 ^ [], 81 ^ [], 82 ^ [])], [(87 ^
      6) ^ []
: [congA(81 ^ [], 82 ^ [], 83 ^ [], 81 ^ [], 83 ^ [], 82 ^ [])]], [(2 ^ 4) ^
     [83 ^ [],
82 ^ [], 81 ^ []] : [-(cong(81 ^ [], 82 ^ [], 81 ^ [], 83 ^ [])), 5 ^ 4 : [(8
     ^ 4) ^ []
: []], isosceles(81 ^ [], 82 ^ [], 83 ^ [])], [(85 ^ 5) ^ [] : [-(isosceles
     (81 ^ [],
82 ^ [], 83 ^ []))]]], [(20 ^ 4) ^ [83 ^ [], 82 ^ [], 81 ^ [], 81 ^ []] :
[-(cong(81 ^ [], 83 ^ [], 81 ^ [], 82 ^ [])), cong(81 ^ [], 82 ^ [], 81 ^ [],
      83 ^ [])],
[_16585 ^ _16586 : [-(cong(81 ^ [], 82 ^ [], 81 ^ [], 83 ^ []))]]]]]]]
End of proof for tptp-problems/euclid-native-eq/015_proposition_05.p
```

## A.3 Vampire

[149, 27 ⟶ 151, subsumption resolution]

$$\frac{congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0)) \quad \neg congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0))}{\square}$$

[25 ⟶ 27, cnf transformation]

$$\frac{isosceles(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0)) \wedge \neg congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0))}{\neg congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0))}$$

[148, 26 ⟶ 149, resolution]

$$\frac{\neg isosceles(x_0, x_1, x_2) \vee congA(x_0, x_1, x_2, x_0, x_2, x_1) \quad isosceles(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0))}{congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0))}$$

[25 ⟶ 26, cnf transformation]

$$\frac{isosceles(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0)) \wedge \neg congA(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0), \sigma_0(a0), \sigma_2(a0), \sigma_1(a0))}{isosceles(\sigma_0(a0), \sigma_1(a0), \sigma_2(a0))}$$

[145, 50 ⟶ 148, subsumption resolution]

$$\frac{congA(x_0, x_1, x_2, x_0, x_2, x_1) \vee col(x_1, x_0, x_2) \vee \neg isosceles(x_0, x_1, x_2) \quad \neg isosceles(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2)}{\neg isosceles(x_0, x_1, x_2) \vee congA(x_0, x_1, x_2, x_0, x_2, x_1)}$$

[45, 31 ⟶ 50, resolution]

$$\frac{\neg triangle(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2) \quad triangle(x_0, x_1, x_2) \vee \neg isosceles(x_0, x_1, x_2)}{\neg isosceles(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2)}$$

[16 ⟶ 31, cnf transformation]

$$\frac{(\forall x_0 \forall x_1 \forall x_2)(\neg isosceles(x_0, x_1, x_2) \vee (triangle(x_0, x_1, x_2) \wedge cong(x_0, x_1, x_0, x_2)))}{triangle(x_0, x_1, x_2) \vee \neg isosceles(x_0, x_1, x_2)}$$

[34, 33 ⟶ 45, resolution]

$$\frac{col(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2) \quad \neg col(x_0, x_1, x_2) \vee \neg triangle(x_0, x_1, x_2)}{\neg triangle(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2)}$$

[17 ⟶ 33, cnf transformation]

$$\frac{(\forall x_0 \forall x_1 \forall x_2)(\neg triangle(x_0, x_1, x_2) \vee \neg col(x_0, x_1, x_2))}{\neg col(x_0, x_1, x_2) \vee \neg triangle(x_0, x_1, x_2)}$$

[18 ⟶ 34, cnf transformation]

$$\frac{(\forall x_0 \forall x_1 \forall x_2)(\neg col(x_0, x_1, x_2) \vee (col(x_1, x_0, x_2) \wedge col(x_1, x_2, x_0) \wedge col(x_2, x_0, x_1) \wedge col(x_0, x_2, x_1) \wedge col(x_2, x_1, x_0)))}{col(x_1, x_0, x_2) \vee \neg col(x_0, x_1, x_2)}$$

[144, 32 ⟶ 145, resolution]

$$\frac{\neg cong(x_0, x_1, x_0, x_2) \vee congA(x_0, x_1, x_2, x_0, x_2, x_1) \vee col(x_1, x_0, x_2) \quad cong(x_0, x_1, x_0, x_2) \vee \neg isosceles(x_0, x_1, x_2)}{congA(x_0, x_1, x_2, x_0, x_2, x_1) \vee col(x_1, x_0, x_2) \vee \neg isosceles(x_0, x_1, x_2)}$$

[16 ⟶ 32, cnf transformation]

$$\frac{(\forall x_0 \forall x_1 \forall x_2)(\neg isosceles(x_0, x_1, x_2) \vee (triangle(x_0, x_1, x_2) \wedge cong(x_0, x_1, x_0, x_2)))}{cong(x_0, x_1, x_0, x_2) \vee \neg isosceles(x_0, x_1, x_2)}$$

[143, 40 ⟶ 144, subsumption resolution]

$$\frac{congA(x_0, x_1, x_2, x_0, x_2, x_1) \vee \neg cong(x_0, x_2, x_0, x_1) \vee \neg cong(x_0, x_1, x_0, x_2) \vee col(x_1, x_0, x_2) \quad cong(x_0, x_3, x_1, x_2) \vee \neg cong(x_1, x_2, x_0, x_3)}{\neg cong(x_0, x_1, x_0, x_2) \vee congA(x_0, x_1, x_2, x_0, x_2, x_1) \vee col(x_1, x_0, x_2)}$$

[21 ⟶ 40, cnf transformation]

$$\frac{(\forall x_0 \forall x_1 \forall x_2 \forall x_3)(\neg cong(x_1, x_2, x_0, x_3) \vee cong(x_0, x_3, x_1, x_2))}{cong(x_0, x_3, x_1, x_2) \vee \neg cong(x_1, x_2, x_0, x_3)}$$

[42, 30 ⟶ 143, resolution]
[15 ⟶ 30, cnf transformation]
[23 ⟶ 42, cnf transformation]
[22 ⟶ 23, flattening]
[12 ⟶ 22, cnf transformation]
[8 ⟶ 12, rectify]
[8, input]
[7 ⟶ 15, cnf transformation]
[7, input]
[3 ⟶ 21, cnf transformation]
[3, input]
[6 ⟶ 18, cnf transformation]
[6, input]
[4 ⟶ 17, cnf transformation]
[4, input]
[1 ⟶ 16, cnf transformation]
[1, input]
[13, 24 ⟶ 25, skolemisation]
[24, choice axiom]

$$\frac{\forall x_0 \forall x_1 \forall x_2 (isosceles(x_0, x_1, x_2) \rightarrow congA(x_0, x_1, x_2, x_0, x_2, x_1))}{\forall x_0 \forall x_1 \forall x_2 (isosceles(x_0, x_1, x_2) \vee \neg congA(x_0, x_1, x_2, x_0, x_2, x_1))}$$

[11 ⟶ 13, emf transformation]

$$\frac{\forall x_0 \forall x_1 \forall x_2 (isosceles(x_0, x_1, x_2) \vee \neg congA(x_0, x_1, x_2, x_0, x_2, x_1))}{\neg \exists x_0 \exists x_1 \exists x_2 (isosceles(x_0, x_1, x_2) \wedge congA(x_0, x_1, x_2, x_0, x_2, x_1))}$$

[10 ⟶ 11, negated conjecture]

$$\frac{\forall x_0 \forall x_1 \forall x_2 (isosceles(x_0, x_1, x_2) \rightarrow congA(x_0, x_1, x_2, x_0, x_2, x_1))}{\neg \exists x_0 \exists x_1 \exists x_2 (isosceles(x_0, x_1, x_2) \wedge congA(x_0, x_1, x_2, x_0, x_2, x_1))}$$

[10, input]

$$(\forall x_0 \forall x_1 \forall x_2 (isosceles(x_0, x_1, x_2) \rightarrow congA(x_0, x_1, x_2, x_0, x_2, x_1))$$

## A.4 iProver

```
% SZS output start CNFRefutation for 015_proposition_05.p

cnf(c_1,plain,
    ( ~ congA(X0,X1,X2,X3,X4,X5)
    | congA(X1,X2,X0,X4,X5,X3)
    | ~ cong(X1,X2,X4,X5)
    | ~ cong(X1,X0,X4,X3) ),
    file('clausifier', c_0_40) ).

cnf(c_7,plain,
    ( cong(X0,X1,X0,X2) | ~ isosceles(X0,X1,X2) ),
    file('clausifier', c_0_46) ).

cnf(c_16,negated_conjecture,
    ( isosceles(esk1_0,esk2_0,esk3_0) ),
    file('clausifier', c_0_55) ).

cnf(c_206,plain,
    ( cong(esk1_0,esk2_0,esk1_0,esk3_0) ),
    inference(resolution,[status(thm)],[c_7,c_16]) ).

cnf(c_398,plain,
    ( ~ congA(X0,esk1_0,esk2_0,X1,esk1_0,esk3_0)
    | congA(esk1_0,esk2_0,X0,esk1_0,esk3_0,X1)
    | ~ cong(esk1_0,X0,esk1_0,X1) ),
    inference(resolution,[status(thm)],[c_1,c_206]) ).

cnf(c_5,plain,
    ( ~ cong(X0,X1,X2,X3) | cong(X2,X3,X0,X1) ),
    file('clausifier', c_0_44) ).

cnf(c_394,plain,
    ( cong(esk1_0,esk3_0,esk1_0,esk2_0) ),
    inference(resolution,[status(thm)],[c_5,c_206]) ).

cnf(c_461,plain,
    ( ~ congA(esk3_0,esk1_0,esk2_0,esk2_0,esk1_0,esk3_0)
    | congA(esk1_0,esk2_0,esk3_0,esk1_0,esk3_0,esk2_0) ),
    inference(resolution,[status(thm)],[c_398,c_394]) ).

cnf(c_11,plain,
    ( ~ col(X0,X1,X2) | col(X1,X2,X0) ),
    file('clausifier', c_0_50) ).

cnf(c_138,plain,
    ( ~ triangle(X0,X1,X2) | triangle(X2,X0,X1) ),
    inference(def_merge,[status(esa)],[c_11]) ).

cnf(c_14,plain,
    ( triangle(X0,X1,X2) | ~ isosceles(X0,X1,X2) ),
    file('clausifier', c_0_53) ).

cnf(c_200,plain,
    ( triangle(esk1_0,esk2_0,esk3_0) ),
    inference(resolution,[status(thm)],[c_14,c_16]) ).

cnf(c_412,plain,
    ( triangle(esk3_0,esk1_0,esk2_0) ),
    inference(resolution,[status(thm)],[c_138,c_200]) ).

cnf(c_4,plain,
    ( congA(X0,X1,X2,X2,X1,X0) | col(X0,X1,X2) ),
    file('clausifier', c_0_43) ).

cnf(c_134,plain,
    ( congA(X0,X1,X2,X2,X1,X0) | ~ triangle(X0,X1,X2) ),
    inference(def_merge,[status(esa)],[c_4]) ).

cnf(c_435,plain,
    ( congA(esk3_0,esk1_0,esk2_0,esk2_0,esk1_0,esk3_0) ),
    inference(resolution,[status(thm)],[c_412,c_134]) ).

cnf(c_3,negated_conjecture,
    ( ~ congA(esk1_0,esk2_0,esk3_0,esk1_0,esk3_0,esk2_0) ),
    file('clausifier', c_0_42) ).

cnf(contradiction,plain,
    ( $false ),
    inference(minisat,[status(thm)],[c_461,c_435,c_3]) ).

% SZS output end CNFRefutation for 015_proposition_05.p
```

## A.5 Zenon

```
Theorem proposition_05 : (forall A : zenon_U, (forall B : zenon_U, (forall C
     : zenon_U,
((isosceles A B C)->(congA A B C A C B))))).
Proof.
assert (zenon_L1_ : forall (zenon_TC_n : zenon_U) (zenon_TB_o : zenon_U) (
     zenon_TA_p : zenon_U),
(forall B : zenon_U, (forall C : zenon_U, ((isosceles zenon_TA_p B C)->
((triangle zenon_TA_p B C)/\(cong zenon_TA_p B zenon_TA_p C))))) ->
(~(cong zenon_TA_p zenon_TB_o zenon_TA_p zenon_TC_n)) -> (isosceles
     zenon_TA_p zenon_TB_o zenon_TC_n) -> False).
do 3 intro. intros zenon_Ha zenon_Hb zenon_Hc.
generalize (zenon_Ha zenon_TB_o). zenon_intro zenon_H10.
generalize (zenon_H10 zenon_TC_n). zenon_intro zenon_H11.
apply (zenon_imply_s _ _ zenon_H11); [ zenon_intro zenon_H13 | zenon_intro
     zenon_H12 ].
exact (zenon_H13 zenon_Hc).
apply (zenon_and_s _ _ zenon_H12). zenon_intro zenon_H15. zenon_intro
     zenon_H14.
exact (zenon_Hb zenon_H14).
(* end of lemma zenon_L1_ *)
assert (zenon_L2_ : forall (zenon_TC_n : zenon_U) (zenon_TB_o : zenon_U) (
     zenon_TA_p : zenon_U),
(isosceles zenon_TA_p zenon_TB_o zenon_TC_n) -> (~(cong zenon_TA_p zenon_TB_o
      zenon_TA_p zenon_TC_n)) -> False).
do 3 intro. intros zenon_Hc zenon_Hb.
generalize (defisosceles zenon_TA_p). zenon_intro zenon_Ha.
apply (zenon_L1_ zenon_TC_n zenon_TB_o zenon_TA_p); trivial.
(* end of lemma zenon_L2_ *)
assert (zenon_L3_ : forall (zenon_TC_n : zenon_U) (zenon_TB_o : zenon_U) (
     zenon_TA_p : zenon_U),
(isosceles zenon_TA_p zenon_TB_o zenon_TC_n) -> (~(cong zenon_TA_p zenon_TC_n
      zenon_TA_p zenon_TB_o)) -> False).
do 3 intro. intros zenon_Hc zenon_H16.
generalize (lemma_congruencesymmetric zenon_TA_p). zenon_intro zenon_H17.
generalize (defisosceles zenon_TA_p). zenon_intro zenon_Ha.
generalize (zenon_H17 zenon_TA_p). zenon_intro zenon_H18.
generalize (zenon_H18 zenon_TB_o). zenon_intro zenon_H19.
generalize (zenon_H19 zenon_TC_n). zenon_intro zenon_H1a.
apply (zenon_imply_s _ _ zenon_H1a); [ zenon_intro zenon_Hb | zenon_intro
     zenon_H1b ].
apply (zenon_L1_ zenon_TC_n zenon_TB_o zenon_TA_p); trivial.
exact (zenon_H16 zenon_H1b).
(* end of lemma zenon_L3_ *)
apply NNPP. intro zenon_G.
apply (zenon_notallex_s (fun A : zenon_U => (forall B : zenon_U, (forall C :
     zenon_U,
((isosceles A B C)->(congA A B C A C B))))) zenon_G); [ zenon_intro zenon_H1c
     ; idtac ].
elim zenon_H1c. zenon_intro zenon_TA_p. zenon_intro zenon_H1d.
apply (zenon_notallex_s (fun B : zenon_U => (forall C : zenon_U, ((isosceles
     zenon_TA_p B C)->
(congA zenon_TA_p B C zenon_TA_p C B)))) zenon_H1d); [ zenon_intro zenon_H1e;
     idtac ].
elim zenon_H1e. zenon_intro zenon_TB_o. zenon_intro zenon_H1f.
apply (zenon_notallex_s (fun C : zenon_U => ((isosceles zenon_TA_p zenon_TB_o
      C)->
(congA zenon_TA_p zenon_TB_o C zenon_TA_p C zenon_TB_o))) zenon_H1f); [
     zenon_intro zenon_H20; idtac ].
elim zenon_H20. zenon_intro zenon_TC_n. zenon_intro zenon_H21.
apply (zenon_notimply_s _ _ zenon_H21). zenon_intro zenon_Hc. zenon_intro
     zenon_H22.
generalize (lemma_ABCequalsCBA zenon_TB_o). zenon_intro zenon_H23.
generalize (proposition_04 zenon_TA_p). zenon_intro zenon_H24.
generalize (zenon_H24 zenon_TB_o). zenon_intro zenon_H25.
generalize (zenon_H25 zenon_TC_n). zenon_intro zenon_H26.
generalize (zenon_H26 zenon_TA_p). zenon_intro zenon_H27.
generalize (zenon_H27 zenon_TC_n). zenon_intro zenon_H28.
generalize (zenon_H28 zenon_TB_o). zenon_intro zenon_H29.
apply (zenon_imply_s _ _ zenon_H29); [ zenon_intro zenon_H2b | zenon_intro
     zenon_H2a ].
apply (zenon_notand_s _ _ zenon_H2b); [ zenon_intro zenon_Hb | zenon_intro
     zenon_H2c ].
apply (zenon_L2_ zenon_TC_n zenon_TB_o zenon_TA_p); trivial.
apply (zenon_notand_s _ _ zenon_H2c); [ zenon_intro zenon_H16 | zenon_intro
     zenon_H2d ].
apply (zenon_L3_ zenon_TC_n zenon_TB_o zenon_TA_p); trivial.
generalize (zenon_H23 zenon_TA_p). zenon_intro zenon_H2e.
generalize (zenon_H2e zenon_TC_n). zenon_intro zenon_H2f.
apply (zenon_imply_s _ _ zenon_H2f); [ zenon_intro zenon_H31 | zenon_intro
     zenon_H30 ].
apply zenon_H31. zenon_intro zenon_H32.
generalize (lemma_collinearorder zenon_TC_n). zenon_intro zenon_H33.
generalize (zenon_H33 zenon_TB_o). zenon_intro zenon_H34.
generalize (lemma_collinearorder zenon_TB_o). zenon_intro zenon_H35.
generalize (lemma_collinearorder zenon_TA_p). zenon_intro zenon_H36.
generalize (zenon_H36 zenon_TC_n). zenon_intro zenon_H37.
```

```
generalize (zenon_H34 zenon_TA_p). zenon_intro zenon_H38.
apply (zenon_imply_s _ _ zenon_H38); [ zenon_intro zenon_H3a | zenon_intro
    zenon_H39 ].
generalize (zenon_H35 zenon_TA_p). zenon_intro zenon_H3b.
generalize (zenon_H37 zenon_TB_o). zenon_intro zenon_H3c.
apply (zenon_imply_s _ _ zenon_H3c); [ zenon_intro zenon_H3e | zenon_intro
    zenon_H3d ].
generalize (zenon_H3b zenon_TC_n). zenon_intro zenon_H3f.
apply (zenon_imply_s _ _ zenon_H3f); [ zenon_intro zenon_H41 | zenon_intro
    zenon_H40 ].
exact (zenon_H41 zenon_H32).
apply (zenon_and_s _ _ zenon_H40). zenon_intro zenon_H43. zenon_intro
    zenon_H42.
apply (zenon_and_s _ _ zenon_H42). zenon_intro zenon_H45. zenon_intro
    zenon_H44.
exact (zenon_H3e zenon_H45).
apply (zenon_and_s _ _ zenon_H3d). zenon_intro zenon_H47. zenon_intro
    zenon_H46.
apply (zenon_and_s _ _ zenon_H46). zenon_intro zenon_H49. zenon_intro
    zenon_H48.
exact (zenon_H3a zenon_H49).
apply (zenon_and_s _ _ zenon_H39). zenon_intro zenon_H4b. zenon_intro
    zenon_H4a.
apply (zenon_and_s _ _ zenon_H4a). zenon_intro zenon_H32. zenon_intro
    zenon_H4c.
apply (zenon_and_s _ _ zenon_H4c). zenon_intro zenon_H45. zenon_intro
    zenon_H4d.
apply (zenon_and_s _ _ zenon_H4d). zenon_intro zenon_H47. zenon_intro
    zenon_H43.
generalize (deftriangle zenon_TA_p). zenon_intro zenon_H4e.
generalize (zenon_H4e zenon_TB_o). zenon_intro zenon_H4f.
generalize (zenon_H4f zenon_TC_n). zenon_intro zenon_H50.
apply (zenon_imply_s _ _ zenon_H50); [ zenon_intro zenon_H52 | zenon_intro
    zenon_H51 ].
generalize (defisosceles zenon_TA_p). zenon_intro zenon_Ha.
generalize (zenon_Ha zenon_TB_o). zenon_intro zenon_H10.
generalize (zenon_H10 zenon_TC_n). zenon_intro zenon_H11.
apply (zenon_imply_s _ _ zenon_H11); [ zenon_intro zenon_H13 | zenon_intro
    zenon_H12 ].
exact (zenon_H13 zenon_Hc).
apply (zenon_and_s _ _ zenon_H12). zenon_intro zenon_H15. zenon_intro
    zenon_H14.
exact (zenon_H52 zenon_H15).
exact (zenon_H51 zenon_H43).
exact (zenon_H2d zenon_H30).
apply (zenon_and_s _ _ zenon_H2a). zenon_intro zenon_H54. zenon_intro
    zenon_H53.
apply (zenon_and_s _ _ zenon_H53). zenon_intro zenon_H56. zenon_intro
    zenon_H55.
exact (zenon_H22 zenon_H56).
Qed.
```

## A.6 Larus

Not minimal proof:

```
Theorem proposition_05 : forall A B C : MyT, isosceles A B C -> congA A B C A
     C B.
Proof.
intros ca cb cc.
intros.
assert (col ca cb cc \/ ~ col ca cb cc) by applying (cn_col1b ca cb cc).
by cases on ( ( col ca cb cc )  \/  ( ~ col ca cb cc ) ).
- {
   assert (triangle ca cb cc) by applying (defisosceles ca cb cc).
   assert (~ col ca cb cc) by applying (deftriangle ca cb cc).
   assert (col ca ca ca \/ ~ col ca ca ca) by applying (cn_col1b ca ca ca).
   by cases on ( ( col ca ca ca )  \/  ( ~ col ca ca ca ) ).
   - {
      assert (False) by contradiction_on (col ca cb cc).
      contradict.
     }
   - {
      assert (~ col ca cb cc) by applying (deftriangle ca cb cc).
      assert (False) by contradiction_on (col ca cb cc).
      contradict.
     }
  }
- {
   assert (col cc ca cb \/ ~ col cc ca cb) by applying (cn_col1b cc ca cb).
   by cases on ( ( col cc ca cb )  \/  ( ~ col cc ca cb ) ).
```

```
    - {
       assert (col ca cc cb) by applying (lemma_collinearorder cc ca cb).
       assert (col ca cb cc) by applying (lemma_collinearorder ca cc cb).
       assert (False) by contradiction_on (col ca cb cc).
       contradict.
      }
    - {
       assert (cong ca cb ca cc) by applying (defisosceles ca cb cc).
       assert (triangle ca cb cc) by applying (defisosceles ca cb cc).
       assert (cong ca cc ca cb) by applying (lemma_congruencesymmetric ca ca
            cb cc).
       assert (isosceles ca cb cc) by applying (defisosceles2 ca cb cc).
       assert (congA cc ca cb cb ca cc) by applying (lemma_ABCequalsCBA cc ca
            cb).
       assert (congA ca cb cc ca cc cb) by applying (proposition_04 ca cc cb
            ca cb cc).
       conclude.
      }
  }
Qed.
```

Minimal CL proof found:

```
Theorem proposition_05 : forall A B C : MyT, isosceles A B C -> congA A B C A
     C B.
Proof.
intros a b c.
intros.
assert (cong a b a c) by applying (defisosceles a b c).
assert (triangle a b c) by applying (defisosceles a b c).
assert (col b a c \/ ~ col b a c) by applying (cn_col1b b a c).
by cases on ( ( col b a c )  \/  ( ~ col b a c ) ).
- {
   assert (~ col a b c) by applying (deftriangle a b c).
   assert (col a b c) by applying (lemma_collinearorder b a c).
   assert (False) by contradiction_on (col a b c).
   contradict.
  }
- {
   assert (cong a c a b) by applying (lemma_congruencesymmetric a a b c).
   assert (congA b a c c a b) by applying (lemma_ABCequalsCBA b a c).
   assert (congA a b c a c b) by applying (proposition_04 a b c a c b).
   conclude.
  }
Qed.
```

Proof with inline lemmas:

```
Lemma defisoscelesAuxConjConcl2sat0 : forall  A  B  C,  isosceles A B C ->
     cong A C A B.
Proof.
inline_lemma_solver.
Qed.

Hint Resolve defisoscelesAuxConjConcl2sat0 : Sym.

Lemma defisoscelesAuxConjConcl0sat1 : forall  A  B  C,  isosceles A B C -> ~
     col A B C.
Proof.
inline_lemma_solver.
Qed.

Hint Resolve defisoscelesAuxConjConcl0sat1 : Sym.

Lemma deftrianglesat2 : forall  A  B  C,  triangle A B C -> congA A B C C B A
    .
Proof.
inline_lemma_solver.
Qed.

Hint Resolve deftrianglesat2 : Sym.

Lemma defisoscelesAuxConjConcl0sat1sat3 : forall  A  B  C,  isosceles A B C
     -> congA A B C C B A.
Proof.
inline_lemma_solver.
Qed.

Hint Resolve defisoscelesAuxConjConcl0sat1sat3 : Sym.
```

```
Theorem proposition_05 : forall A B C : MyT, isosceles A B C -> congA A B C A
     C B.
Proof.
intros a b c.
intros.
assert (isosceles a b c) by applying (defisosceles2 a b c).
assert (col b a c \/ ~ col b a c) by (destruct (classic (col b a c));auto).
by cases on ( ( col b a c )  \/  ( ~ col b a c ) ).
- {
   assert (False) by contradiction_on (col a b c).
   contradict.
  }
- {
   assert (congA a b c a c b) by applying (proposition_04 a b c a c b).
   conclude.
  }
Qed.
```

## A.7 Isabelle

Isabelle 2016 generated several proof using Sledgehammer. The first proof is the most informative but we can still feel that the proof has been converted from resolution proof. Moreover, the generated proof is not accepted by Isabelle. The second proof is accepted by not very informative. The third proof is a one-liner calling the meeson tactic.

```
lemma proposition_05 :
  "∀ A B C. isosceles A B C → congA A B C A C B"
by (meson defisosceles deftriangle lemma_ABCequalsCBA
  lemma_collinearorder lemma_congruencesymmetric proposition_04)
```

```
lemma proposition_05 :
  "∀ A B C. isosceles A B C → congA A B C A C B"
proof -
  have "col v2_0 v0_2 v1_1 ∨ congA v2_0 v0_2 v1_1 v1_1 v0_2 v2_0"
    using lemma_ABCequalsCBA by blast
  then have f1: "¬ isosceles v0_2 v1_1 v2_0 ∨ congA v0_2 v1_1 v2_0 v0_2 v2_0
       v1_1"
    using defisosceles deftriangle lemma_collinearorder
          lemma_congruencesymmetric proposition_04 by blast
  obtain mm :: MyT and mma :: MyT and mmb :: MyT where
    "(∃v0 v1 v2. isosceles v0 v1 v2 ∧ ¬ congA v0 v1 v2 v0 v2 v1) =
  (isosceles mm mma mmb ∧ ¬ congA mm mma mmb mm mmb mma)"
    by blast
  then show ?thesis
    using f1 by blast
 qed

 lemma proposition_05 :
  "∀ A B C. isosceles A B C → congA A B C A C B"
proof -
  { fix mm :: MyT and mma :: MyT and mmb :: MyT
    have "¬ isosceles mm mma mmb ∨ congA mm mma mmb mm mmb mma"
      by (metis (no_types) defisosceles deftriangle lemma_ABCequalsCBA
      lemma_collinearorder lemma_congruencesymmetric proposition_04) }
  then show ?thesis
    by blast
qed
```

## A.8 Manual formal proof in Coq

For comparison, we provide the original Coq proof:

```
Lemma proposition_05 :
   forall A B C,
   isosceles A B C ->
   CongA A B C A C B.
Proof.
intros.
assert ((Triangle A B C /\ Cong A B A C)) by (conclude_def isosceles ).
assert (Cong A C A B) by (conclude lemma_congruencesymmetric).
assert (nCol A B C) by (conclude_def Triangle ).
assert (~ Col C A B).
 {
 intro.
 assert (Col A B C) by (forward_using lemma_collinearorder).
 contradict.
 }
assert (CongA C A B B A C) by (conclude lemma_ABCequalsCBA).
assert ((Cong C B B C /\ CongA A C B A B C /\ CongA A B C A C B))
                                                  by (conclude proposition_04).
close.
Qed.
```