Case study: proving $\sqrt{2}$ irrational with LPTP and an LLM

Fred Mesnard Étienne Payet
LIM, université de La Réunion, France
{frederic.mesnard,etienne.payet}@univ-reunion.fr

Wim Vanhoof
Université de Namur, Belgium
wim.vanhoof@unamur.be

We present the interactions with an LLM (Large Language Model) aiming at proving that $\sqrt{2}$ is not a rational number in an LP (Logic Programming) context. We start from a few basic pure logic programming predicate definitions. We rely on the LPTP (Logic Program Theorem Prover) system written by Robert Stärk for stating and proving properties about logic programs. As the proof language of LPTP is based on natural deduction, the proofs are human readable. In our case study, we sketch in LPTP the usual proof showing the irrationality of $\sqrt{2}$. Then we describe the interactions we had with the LLM. We end up with a complete formal proof, partially generated by an LLM and fully proof-checked by LPTP.

1 Introduction

Consider the following pure Prolog program. Natural numbers are represented as terms built from the constant 0 and the function symbol $s/1$. We borrow the predicates `nat/1`, `plus/3`, `times/3`, `gcd/3`, and `gcd_leq/3` from the LPTP (Logic Program Theorem Prover, [20]) libraries *nat* and *gcd* to facilitate the future use of already proven properties.

```
even(0).                              odd(s(0)).
even(s(s(X))) :- even(X).             odd(s(s(X))) :- odd(X).

nat(0).                               leq(0,X).
nat(s(X)) :- nat(X).                  leq(s(X),s(Y)) :- leq(X,Y).

plus(0,Y,Y).                          times(0,Y,0).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).     times(s(X),Y,Z) :- times(X,Y,P), plus(P,Y,Z).

gcd(X,Y,D) :-                         gcd_leq(X,Y,D) :-
 ( leq(X,Y) -> gcd_leq(X,Y,D)          ( X = 0 -> D = Y
  ; gcd_leq(Y,X,D)).                    ; plus(X,Z,Y), gcd(X,Z,D)).
```

Question: can we find two coprime natural numbers p and q such that $\frac{p}{q} = \sqrt{2}$, *i.e.*, $p^2 = 2q^2$? (Two natural numbers p and q are *coprime* if and only if their greatest common divisor is 1.) This question corresponds to the following Prolog query, designed so that backtracking generates a fair exploration of $\mathbb{N} \times \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers:

```
?- nat(S),plus(P,Q,S),gcd(P,Q,s(0)),times(P,P,P2),times(Q,Q,Q2),plus(Q2,Q2,P2).
```

The answer to the question is *no*. This fact is known since at least the 6th century BC. But since the search space of the above query is infinite, we won't get a negative answer from *any* resolution-based logic programming (LP) engine, Prolog included. However we can prove this fact by other means, still

Preliminary Report. Final version to appear in:
ICLP 2026 Technical Communication

within an LP context. For instance, we can prove $\sqrt{2}$ is not rational using the LPTP system. We take profit of this exercise to evaluate the help we may get from an LLM (Large Language Model).

Recent research has shown that LLMs can be made increasingly effective at automated theorem proving by integrating them with formal provers and proof checkers (see, *e.g.*, [11, 17]). To the best of our knowledge, no such integration has been attempted for LPTP. Yet, LPTP is an interesting formalism and tool as it uses Prolog as representation language, and the associated prover constitutes a rather lightweight and easily manipulable process. The theoretical basis and specification language is purely first-order logic, which is well-known and largely automatable. On the other hand, less documentation and examples might be available in comparison with other provers (such as Rocq, Lean and Isabelle), which might make it more difficult for an LLM to appropriate the formalism.

The paper is organized as follows. Section 2 recalls the basics of LPTP. Section 3 presents a usual proof of the irrationality of $\sqrt{2}$ and a formalization of its skeleton. Section 4 and 5 explain how we interact with the LLM to get a complete proof. Section 6 reviews some related work, and Section 7 concludes.

2 A quick summary of LPTP

The reader already familiar with LPTP can safely skip this section. Let P be a pure logic program where negative literals may appear in the body of clauses (also called *normal program* in [10]). For sake of conciseness, we do not consider built-in predicates (see [20] for a full treatment) other than the equality `=/2`. We start with $\mathscr{L}$, the first-order language associated to P. The *goals* of $\mathscr{L}$ are:

$$G,H ::= \texttt{true} \mid \texttt{fail} \mid s = t \mid A \mid \texttt{\+}\ G \mid (G,H) \mid (G;H) \mid \texttt{some}\ x\ G$$

where s and t are two terms, x is a variable and A is an atomic goal. The goals of $\mathscr{L}$ have the operational semantics specified by ISO-Prolog [7] assuming the occurs check.

$\hat{\mathscr{L}}$ is the specification language of LPTP. For each user-defined predicate symbol R, $\hat{\mathscr{L}}$ does not include R, but instead it contains three predicate symbols R^s, R^f, R^t of the same arity as R, which respectively express success, failure and termination of R. $\hat{\mathscr{L}}$ also contains a unary constraint for groundness gr, expressing that its argument is ground. The *formulas* of $\hat{\mathscr{L}}$ are:

$$\phi, \psi ::= \top \mid \bot \mid s = t \mid R(\vec{t}) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi$$

where $\vec{t}$ is a sequence of n terms and R denotes a n-ary predicate symbol of $\hat{\mathscr{L}}$. The semantics of $\hat{\mathscr{L}}$ is the first-order predicate calculus of classical logic.

For any of the user-defined logic procedures R in a logic program P, $D_R^P(\vec{x})$ denotes its Clark's *if-and-only-if* completed definition, c.f. [2, 10].

For defining the declarative semantics of logic programs, Stärk uses three syntactic operators **S**, **F** and **T** which map goals of $\mathscr{L}$ into $\hat{\mathscr{L}}$-formulas. Intuitively, **S**G means G succeeds (any breadth-first evaluation of G succeeds), **F**G means G fails (the ISO-Prolog evaluation stops without any answer), and **T**G means G terminates (the ISO-Prolog evaluation produces a finite number of answers then stops). Moreover, termination implies a safe use of negation. The definition of the operators follows:

$\mathbf{S}R(\vec{t}) := R^s(\vec{t})$	$\mathbf{S}\ \texttt{true} := \top$	$\mathbf{S}\ \texttt{fail} := \bot$	$\mathbf{S}(s=t) := (s=t)$
$\mathbf{S}\texttt{\+}G := \mathbf{F}G$	$\mathbf{S}(G,H) := \mathbf{S}G \wedge \mathbf{S}H$	$\mathbf{S}(G;H) := \mathbf{S}G \vee \mathbf{S}H$	$\mathbf{S}(\texttt{some}\ x\ G) := \exists x \mathbf{S}G$
$\mathbf{F}R(\vec{t}) := R^f(\vec{t})$	$\mathbf{F}\ \texttt{true} := \bot$	$\mathbf{F}\ \texttt{fail} := \top$	$\mathbf{F}(s=t) := \neg(s=t)$
$\mathbf{F}\texttt{\+}G := \mathbf{S}G$	$\mathbf{F}(G,H) := \mathbf{F}G \vee \mathbf{F}H$	$\mathbf{F}(G;H) := \mathbf{F}G \wedge \mathbf{F}H$	$\mathbf{F}(\texttt{some}\ x\ G) := \forall x \mathbf{F}G$

$\mathbf{T}R(\vec{t}) := R^t(\vec{t})$ $\quad$ $\mathbf{T}\,\texttt{true} := \top$
$\mathbf{T}\,\texttt{fail} := \top$ $\quad$ $\mathbf{T}(s=t) := \top$
$\mathbf{T}\backslash{+}G := \mathbf{T}G \wedge gr(G)$ $\quad$ $\mathbf{T}(G,H) := \mathbf{T}G \wedge (\mathbf{F}G \vee \mathbf{T}H)$
$\mathbf{T}(G;H) := \mathbf{T}G \wedge \mathbf{T}H$ $\quad$ $\mathbf{T}(\texttt{some}\ x\ G) := \forall x \mathbf{T}G$

With LPTP, we prove properties of a logic program P w.r.t. its *inductive extension* IND(P) which includes Clark's completion [2] and induction along the definition of the predicates. Stärk shows that the inductive extension is always consistent and proves various correctness and completeness results w.r.t. the operational semantics of Prolog [20]. The first-order theory IND(P) (cf. [20], pp. 253–254) is defined by nine axiom schemas which we describe now. We omit the fixed point axioms for builtins. Let us point out that the specification language $\hat{\mathscr{L}}$ of LPTP can be extended by new function and predicate symbols, which can be quite handy while formalizing properties.

The first two axioms specify some properties of the trees built from the function symbols extracted from the program under consideration. The third axiom forbids infinite trees.

The axioms of Clark's equality theory

1. $f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n) \rightarrow x_i = y_i$ [if f is n-ary and $1 \leq i \leq n$]
2. $f(x_1,\ldots,x_n) \neq g(y_1,\ldots,y_m)$ [if $n \neq m$ or $f \not\equiv g$]
3. $t \neq x$ [if x occurs in t and $t \not\equiv x$]

LPTP deals with *non-ground* terms, as any ISO-Prolog processor does. LPTP offers a predefined predicate $gr/1$ that we can consider as a constraint. This relation is useful for instance for dealing with negation as failure as LPTP only allows negation by failure for *ground* goals (see the definition $\mathbf{T}\backslash{+}G$).

Axioms for gr/1

4. $\mathrm{gr}(c)$ [if c is a constant]
5. $\mathrm{gr}(x_1) \wedge \ldots \wedge \mathrm{gr}(x_m) \leftrightarrow \mathrm{gr}(f(x_1,\ldots,x_m))$ [f is m-ary]

LPTP considers each user-defined predicate through three points of view: failure, success and termination. These three viewpoints are linked with the following axioms.

Uniqueness axioms and totality axioms

6. $\neg(R^s(\vec{x}) \wedge R^f(\vec{x}))$ [if R is a user-defined predicate]
7. $R^t(\vec{x}) \rightarrow (R^s(\vec{x}) \vee R^f(\vec{x}))$ [if R is a user-defined predicate]

Axiom 6 says that for any tuple of (possibly non-ground) terms, we cannot have at the same time success and failure of R. Axiom 7 states that given termination, we have success or failure. Altogether, it means that for any tuple of terms $\vec{x}$, assuming termination, either $R(\vec{x})$ succeeds or (exclusively) $R(\vec{x})$ fails.

Fixed point axioms for user-defined predicates R

8. $R^s(\vec{x}) \leftrightarrow \mathbf{S}D_R^P(\vec{x}),\ R^f(\vec{x}) \leftrightarrow \mathbf{F}D_R^P(\vec{x}),\ R^t(\vec{x}) \leftrightarrow \mathbf{T}D_R^P(\vec{x})$

We recall that $D_R^P(\vec{x})$ denotes the definition of the completion [2] of the user-defined procedure $R(\vec{x})$ in the logic program P. In the previous section, we saw how to apply the operator **S**, **F** and **T** to formulas. So for instance, the first equivalence $R^s(\vec{x}) \leftrightarrow \mathbf{S}D_R^P(\vec{x})$ defines $R^s(\vec{x})$.

Finally, for any property of the form $\forall\vec{x}[R^s(\vec{x}) \rightarrow \phi(\vec{x})]$, where $R(\vec{x})$ is a user-defined procedure and $\phi(\vec{x})$ an $\hat{\mathscr{L}}$ -formula, we have an induction schema. The interactive prover LPTP is able to *dynamically* generate an induction axiom on demand while the user interacts with it. Let us examine a simple case. It is exactly what happens using LPTP, which slightly generalizes [20]. By *directly recursive user-defined predicate* in the box below, we forbid mutual recursive definitions. Of course, LPTP is able to handle mutually recursive properties, see [18] for some examples.

A (simplified) induction schema for a user-defined predicate R

Let R be a directly recursive user-defined predicate and let $\phi(\vec{x})$ be an $\hat{\mathscr{L}}$-formula such that the length of $\vec{x}$ is equal to the arity of R.
Let $sub(\phi(\vec{x})/R)$ be the formula to be proven $\forall\vec{x}(R^s(\vec{x}) \rightarrow \phi(\vec{x}))$.
Let $closed(\phi(\vec{x})/R)$ be the formula obtained from $\forall\vec{x}(\mathbf{S}D_R^P(\vec{x}) \rightarrow R^s(\vec{x}))$ by replacing

- $R^s(\vec{x})$ by $\phi(\vec{x})$ on the right of $\rightarrow$,
- all occurrences of $R^s(\vec{t})$ appearing on the left of $\rightarrow$ by $\phi(\vec{t}) \wedge R^s(\vec{t})$ (these are the recursive calls of R in D_R^P, turned into R^s by the operator **S**).

Then the induction axiom is the following formula:

9. $closed(\phi(\vec{x})/R) \rightarrow sub(\phi(\vec{x})/R)$

As an illustration, consider the lemma $\forall x\ (even^s(x) \rightarrow odd^s(s(x)))$, which we will use in section 4 (lemma *even:succ:odd*). We have $\vec{x} = x$, $R \equiv even$ and $\phi(x) \equiv odd^s(s(x))$ and $sub(\phi(x)/R)$ is precisely lemma *even:succ:odd*. Now for $closed(\phi(x)/R)$, recall the clauses defining *even* and *odd*: `even(0)`, `even(s(s(X))) :- even(X)`, `odd(s(0))` and `odd(s(s(X))) :- odd(X)`. The Clark completion of *even* is $D_{even}^P(x) \equiv x = 0 \vee \exists y(x = s(s(y)) \wedge even(y))$, and applying the operator **S** gives $\mathbf{S}D_{even}^P(x) \equiv x = 0 \vee \exists y(x = s(s(y)) \wedge even^s(y))$. After the two substitutions ($even^s(x) \rightsquigarrow \phi(x)$ on the right and $even^s(y) \rightsquigarrow \phi(y) \wedge even^s(y)$ on the left), we get: $closed(\phi(x)/even) = \forall x\Big[\big(x = 0 \vee \exists y(x = s(s(y)) \wedge \phi(y) \wedge even^s(y))\big) \rightarrow \phi(x)\Big]$. Let us simplify this formula using three standard equivalences of first-order logic: (a) $\forall x[(A \vee B) \rightarrow C] \equiv \forall x(A \rightarrow C) \wedge \forall x(B \rightarrow C)$; (b) $\forall x(x = t \rightarrow \psi(x)) \equiv \psi(t)$ (when x is not free in t); (c) $\forall x[\exists y P(x,y) \rightarrow Q(x)] \equiv \forall x\forall y[P(x,y) \rightarrow Q(x)]$ (when y is not free in Q). Applying (a) splits $closed(\phi(x)/even)$ into two conjuncts. The first one, $\forall x(x = 0 \rightarrow \phi(x))$, reduces to $\phi(0)$ by (b). The second one, $\forall x[\exists y(x = s(s(y)) \wedge \phi(y) \wedge even^s(y)) \rightarrow \phi(x)]$, becomes after extraction of the existential by (c) and elimination of x by (b) (using $x = s(s(y))$): $\forall y[\phi(y) \wedge even^s(y) \rightarrow \phi(s(s(y)))]$. Renaming y to x and combining with $sub(\phi(x)/even) = \forall x(even^s(x) \rightarrow \phi(x))$, the induction axiom $closed(\phi(x)/even) \rightarrow sub(\phi(x)/even)$ instantiates to: $\big[odd^s(s(0)) \wedge \forall x\big(odd^s(s(x)) \wedge even^s(x) \rightarrow odd^s(s(s(x)))\big)\big] \rightarrow \forall x\big(even^s(x) \rightarrow odd^s(s(x))\big)$.

We refer the reader to the papers of Stärk [20, 19, 18] for a complete presentation of LPTP.

3 Our initial draft

Let us go back to our introductory example. The informal reasoning proving the irrationality of $\sqrt{2}$ that we start with is the following usual proof by contradiction.

> Assume we have two coprime natural numbers p and q such that
>
> $$p^2 = 2q^2$$
>
> So p^2 is even. Hence p is even, because the square of an odd number is odd. Thus there exists a natural number r such that $p = 2r$. By replacing p with $2r$ in the main equation, we get $4r^2 = 2q^2$, which gives $2r^2 = q^2$. Hence q is also even, which contradicts our assumption.

We formalize the above reasoning by first introducing the logical relations $divisor/2$ (which we get from the LPTP library) and $coprime/2$. We also define the logical function $square/1$ which returns the square of its argument. Before defining a function, we must show that the associated definition (here $n \mapsto p$ such that $times(n,n,p)$ holds) is actually functional: for any $nat(n)$, there exists a unique p verifying $times(n,n,p)$. So we need to prove existence and uniqueness, as done below. For proving these two properties, we rely on lemma *times:existence* and lemma *times:uniqueness*, already proven in the LPTP library *nat*.

Definition [*divisor/2*] $\forall x,y\,(divisor(x,y) \leftrightarrow \exists z\,(\mathbf{S}\,\mathtt{nat}(z) \wedge x * z = y))$.

Definition [*coprime/2*] $\forall p,q\,(coprime(p,q) \leftrightarrow \forall r\,(\mathbf{S}\,\mathtt{nat}(r) \wedge divisor(r,p) \wedge divisor(r,q) \rightarrow r = \mathtt{s}(\mathtt{0})))$.

Lemma [*square:existence*] $\forall n\,(\mathbf{S}\,\mathtt{nat}(n) \rightarrow \exists p\,\mathbf{S}\,\mathtt{times}(n,n,p))$. **Proof.**

Assumption$_0$: $\mathbf{S}\,\mathtt{nat}(n)$. $\mathbf{S}\,\mathtt{nat}(n) \wedge \mathbf{S}\,\mathtt{nat}(n)$. $\exists z\,\mathbf{S}\,\mathtt{times}(n,n,z)$ by
Lemma times:existence [*times:existence*].
Thus$_0$: $\mathbf{S}\,\mathtt{nat}(n) \rightarrow \exists p\,\mathbf{S}\,\mathtt{times}(n,n,p)$. □

Lemma [*square:uniqueness*] $\forall n,sn,sp\,(\mathbf{S}\,\mathtt{times}(n,n,sn) \wedge \mathbf{S}\,\mathtt{times}(n,n,sp) \rightarrow sn = sp)$. **Proof.**

Assumption$_0$: $\mathbf{S}\,\mathtt{times}(n,n,sn) \wedge \mathbf{S}\,\mathtt{times}(n,n,sp)$. $sn = sp$ by
Lemma times:uniqueness [*times:uniqueness*].
Thus$_0$: $\mathbf{S}\,\mathtt{times}(n,n,sn) \wedge \mathbf{S}\,\mathtt{times}(n,n,sp) \rightarrow sn = sp$. □

Definition [*square/1*] $\forall n,p\,(\mathbf{S}\,\mathtt{nat}(n) \rightarrow (square(n) = p \leftrightarrow \mathbf{S}\,\mathtt{times}(n,n,p)))$.

Now we proceed top-down, starting from the informal proof which we translate in LPTP as our main theorem. We also need auxiliary results, and we ask to the proof checker to admit them by using the LPTP tactic *by gap*. The proof checker validates the whole proof skeleton and produces a TEXfile. Here is its PDF rendering.

Theorem [*sqrt2:irrational*] $\forall p,q\,(\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q) \rightarrow$
$\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q))$. **Proof.**

Assumption$_0$: $\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q)$.

Contra$_1$: $\mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. $\mathbf{S}\,\mathtt{even}(\mathit{square}(p))$ by **GAP**. $\mathbf{S}\,\mathtt{even}(p)$ by **GAP**.

$\exists r\,(\mathbf{S}\,\mathtt{nat}(r) \wedge p = \mathtt{s}(\mathtt{s}(0)) * r)$ by **GAP**.

Let$_2$ r with $\mathbf{S}\,\mathtt{nat}(r) \wedge p = \mathtt{s}(\mathtt{s}(0)) * r$. $\mathit{square}(p) = \mathit{square}(\mathtt{s}(\mathtt{s}(0))) * \mathit{square}(r)$ by **GAP**. $\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0)))) * \mathit{square}(r) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$ by **GAP**. $\mathtt{s}(\mathtt{s}(0)) * \mathit{square}(r) = \mathit{square}(q)$ by **GAP**. $\mathbf{S}\,\mathtt{even}(q)$ by **GAP**.

Thus$_2$: $\mathbf{S}\,\mathtt{even}(q)$. $\mathit{divisor}(\mathtt{s}(\mathtt{s}(0)),q)$ by **GAP**. $\mathit{divisor}(\mathtt{s}(\mathtt{s}(0)),p)$ by **GAP**. $\mathbf{S}\,\mathtt{nat}(\mathtt{s}(\mathtt{s}(0)))$. $\mathit{coprime}(p,q)$. $\forall r\,(\mathbf{S}\,\mathtt{nat}(r) \wedge \mathit{divisor}(r,p) \wedge \mathit{divisor}(r,q) \rightarrow r = \mathtt{s}(0))$ by Definition coprime/2 [*coprime/2*]. $\mathbf{S}\,\mathtt{nat}(\mathtt{s}(\mathtt{s}(0))) \wedge \mathit{divisor}(\mathtt{s}(\mathtt{s}(0)),p) \wedge \mathit{divisor}(\mathtt{s}(\mathtt{s}(0)),q) \rightarrow \mathtt{s}(\mathtt{s}(0)) = \mathtt{s}(0)$. $\mathtt{s}(\mathtt{s}(0)) = \mathtt{s}(0)$. $\bot$.

Thus$_1$: $\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. $\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$.

Thus$_0$: $\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q) \rightarrow \neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. □

Then we refine the previous attempt. We define the properties we need for proving the main theorem by *stating* these auxiliary properties. We do not prove *any* of these properties for the moment. For each proof, we just write: $\bot$ (*i.e.*, *false*) is admitted (by **GAP**). As from $\bot$ everything is true, each property holds. This pseudo proof appears for the first lemma and is omitted for the others.

Lemma [*nat:natsquare*] $\forall n\,(\mathbf{S}\,\mathtt{nat}(n) \rightarrow \mathbf{S}\,\mathtt{nat}(\mathit{square}(n)))$. **Proof.** $\bot$ by **GAP**. □

Lemma [*twotimes:even*] $\forall n,p\,(\mathbf{S}\,\mathtt{nat}(p) \wedge \mathtt{s}(\mathtt{s}(0)) * p = n \rightarrow \mathbf{S}\,\mathtt{even}(n))$.

Lemma [*evenpp:evenp*] $\forall p\,(\mathbf{S}\,\mathtt{even}(\mathit{square}(p)) \rightarrow \mathbf{S}\,\mathtt{even}(p))$.

Lemma [*even:twotimes*] $\forall n\,(\mathbf{S}\,\mathtt{even}(n) \rightarrow \exists p\,(\mathbf{S}\,\mathtt{nat}(p) \wedge n = \mathtt{s}(\mathtt{s}(0)) * p))$.

Lemma [*npq:nnppqq*] $\forall n,p,q\,(n = p * q \rightarrow \mathit{square}(n) = \mathit{square}(p) * \mathit{square}(q))$.

Lemma [*sqr2:4*] $\mathit{square}(\mathtt{s}(\mathtt{s}(0))) = \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0))))$.

Lemma [*simplify:by2*] $\forall n,p\,(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0)))) * n = \mathtt{s}(\mathtt{s}(0)) * p \rightarrow \mathtt{s}(\mathtt{s}(0)) * n = p)$.

Lemma [*even:div2*] $\forall n\,(\mathbf{S}\,\mathtt{even}(n) \rightarrow \mathit{divisor}(\mathtt{s}(\mathtt{s}(0)),n))$.

Assuming for the moment that these properties are true, here is the new version of the main theorem. It mimics the informal proof we started with, but with a bit more details so that there is no gap. Gaps now only appear in the lemmas. The file is proof-checked.

Theorem [*sqrt2:irrational*] $\forall p,q\,(\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q) \rightarrow$
$\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q))$. **Proof.**

Assumption$_0$: $\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q)$.

Contra$_1$: $\mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. $\mathbf{S}\,\mathtt{nat}(\mathit{square}(q))$ by

Lemma nat:natsquare [*nat:natsquare*]. $\mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q) = \mathit{square}(p)$. $\mathbf{S}\,\mathtt{even}(\mathit{square}(p))$ by

Lemma twotimes:even [*twotimes:even*]. $\mathbf{S}\,\mathtt{even}(p)$ by Lemma evenpp:evenp [*evenpp:evenp*].

$\exists r\,(\mathbf{S}\,\mathtt{nat}(r) \wedge p = \mathtt{s}(\mathtt{s}(0)) * r)$ by Lemma even:twotimes [*even:twotimes*].

Let$_2$ r with $\mathbf{S}\,\mathtt{nat}(r) \wedge p = \mathtt{s}(\mathtt{s}(0)) * r$. $\mathit{square}(p) = \mathit{square}(\mathtt{s}(\mathtt{s}(0))) * \mathit{square}(r)$

by Lemma npq:nnppqq [*npq:nnppqq*]. $\mathit{square}(\mathtt{s}(\mathtt{s}(0))) = \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0))))$

by Lemma sqr2:4 [*sqr2:4*]. $\mathit{square}(p) = \mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0)))) * \mathit{square}(r)$.

$\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{s}(0)))) * \mathit{square}(r) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. $\mathtt{s}(\mathtt{s}(0)) * \mathit{square}(r) = \mathit{square}(q)$ by

Lemma simplify:by2 [*simplify:by2*]. $\mathbf{S}\,\mathtt{nat}(\mathit{square}(r))$ by Lemma nat:natsquare [*nat:natsquare*].

$\mathbf{S}\,\mathtt{even}(\mathit{square}(q))$ by Lemma twotimes:even [*twotimes:even*].

$\mathbf{S}\,\mathtt{even}(q)$ by Lemma evenpp:evenp [*evenpp:evenp*].

Thus$_2$: $\mathbf{S}\,\mathtt{even}(q)$. $\mathit{divisor}(\mathtt{s}(\mathtt{s}(0)), q)$ by Lemma even:div2 [*even:div2*].

$\mathit{divisor}(\mathtt{s}(\mathtt{s}(0)), p)$ by Lemma even:div2 [*even:div2*]. $\mathbf{S}\,\mathtt{nat}(\mathtt{s}(\mathtt{s}(0)))$. $\mathit{coprime}(p,q)$.

$\forall r\,(\mathbf{S}\,\mathtt{nat}(r) \wedge \mathit{divisor}(r,p) \wedge \mathit{divisor}(r,q) \rightarrow r = \mathtt{s}(0))$ by Definition coprime/2 [*coprime/2*].

$\mathbf{S}\,\mathtt{nat}(\mathtt{s}(\mathtt{s}(0))) \wedge \mathit{divisor}(\mathtt{s}(\mathtt{s}(0)), p) \wedge \mathit{divisor}(\mathtt{s}(\mathtt{s}(0)), q) \rightarrow \mathtt{s}(\mathtt{s}(0)) = \mathtt{s}(0)$. $\mathtt{s}(\mathtt{s}(0)) = \mathtt{s}(0)$. $\bot$.

Thus$_1$: $\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. $\neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$.

Thus$_0$: $\mathbf{S}\,\mathtt{nat}(p) \wedge \mathbf{S}\,\mathtt{nat}(q) \wedge \mathit{coprime}(p,q) \rightarrow \neg \mathit{square}(p) = \mathtt{s}(\mathtt{s}(0)) * \mathit{square}(q)$. □

For each lemma, an LPTP warning is reported: `there is a gap in the proof.` But our initial draft is syntactically and logically coherent. Moreover, the level of granularity seems reasonable, at least for us. We moved from the informal usual high-level proof of the beginning of this section to the LPTP equivalent of the low-level Peano axioms. Note that even the most elementary pieces (like $2^2 = 4$) of the above reasoning have to be proven (see Lemma *sqr2:4*). Let us ask for help for proving these lemmas by invoking an LLM.

4 The proof completed

We choose Claude from Anthropic and do the experiment with Opus 4.5, the most powerful large language model released by Anthropic in late 2025. As this model is well known for its efficiency in programming tasks, we expect it to be able to construct proofs validated with the LPTP proof checker. We did not try any cheaper model (but we also run resolution-based automated theorem provers). All interactions used Anthropic's default decoding settings (no temperature or sampling tuning). We perform *in-context learning*, *i.e.*, we feed the LLM with the LPTP user manual in PDF format. We also give Claude two basic LPTP libraries (Prolog code and LPTP proofs in text format), one for Peano numbers and the other one about lists. The code and library of properties for Peano numbers will be used in the proofs generated by Claude. On the other hand, the code and properties for lists will not be directly used but give many other proof examples to Claude.

Our interaction follows a simple feedback loop: we submit a lemma (most often in LPTP syntax, sometimes in natural language) to Claude, feed its proof to the LPTP proof checker, and on failure forward to Claude the first `incorrect derivation step` reported by LPTP. We iterate until either LPTP accepts the proof (DP or BF in the tables below) or we estimate that Claude is not converging, in which case we provide a natural-language hint (PH) or the proof itself (PG). The human developer is

responsible for choosing the order in which lemmas are attempted and for deciding when to abandon a back-and-forth. The ability of LPTP to localize the first incorrect step is what makes this loop tractable in practice.

So we start by asking the proofs of the easiest (from our point of view) lemmas, one lemma after the other. We state the lemmas formally using the LPTP syntax. The proofs of lemmas *nat:natsquare* (informally: $nat(n) \to nat(n^2)$), *sqr2:4* ($2^2 = 4$), *twotimes:even* ($n = 2p \to even(n)$), *even:twotimes* ($even(n) \to n = 2p$)), and *even:div2* ($even(n) \to 2|n$) are proposed by Claude and accepted by the proof checker of LPTP. For instance, our prompt for lemma *sqr2:4* reads:

```
Donne-moi une preuve LPTP de :
:- lemma(sqr2:4, square(s(s(0))) = s(s(s(s(0)))), ff by gap).
```

mixing a brief French instruction with the formal LPTP statement, the `ff by gap` clause being the placeholder to be filled in. With each of its proofs, Claude summarizes in natural language the idea of the proof. For lemma *sqr2:4*, Claude explains that it first shows that 0, 1 and 2 are natural numbers. Then it computes $times(2,2,4)$ step by step: we have $times(0,2,0)$, $plus(2,0,2)$ hence $times(1,2,2)$. We also have $plus(2,2,4)$ hence $times(2,2,4)$. Finally, we get $square(2) = 4$.

The proofs of *twotimes:even* ($2p = n \to even(n)$) and its reciprocal *even:twotimes* rely heavily on results from the *nat* library given to Claude initially, which it uses adequately.

For the lemma *simplify:by2* ($4n = 2p \to 2n = p$), we have to handle a few back and forths between Claude and LPTP. The proof checker detects some *incorrect derivation steps* at some places in the generated proof. Each time, Claude analyzes the error and proposes a new proof. Claude proposed and proved the auxiliary lemma *times:injective:second*. Finally, Claude was able to correct all the errors.

We observe a similar behaviour for the lemma *npq:nnppqq* ($n = pq \to n^2 = p^2q^2$).

Table 1 gives our obtained results and Figure 1 shows the dependency graph of the proof. We also add a column ATP for recording the answers we get from running two automated theorem provers on the same lemmas within a 20 seconds timeout. This approach is fully described in [12] and summarized in the next section.

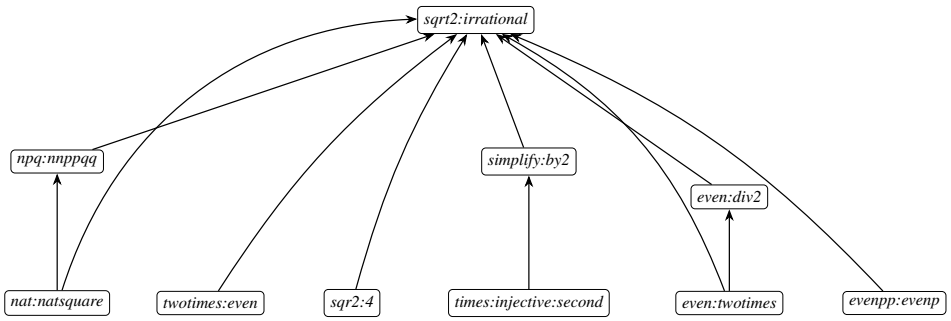

Figure 1: Dependency graph of the lemmas of Table 1. An arrow $A \to B$ means A is used in the proof of B.

The last remaining property, *evenpp:evenp* ($even(p^2) \to even(p)$), proved much trickier: Claude failed to find a proof or even go in the right direction. So we derived the main steps by hand and then asked Claude to prove all auxiliary lemmas stated in natural language.

Finally we ask Claude to prove the lemma *evenpp:evenp* ($nat(p) \wedge even(p^2) \to even(p)$) using the following hint.

Lemma or theorem	Shorthand	LLM	ATP
nat:natsquare	$nat(n) \to nat(square(n))$	DP	yes
twotimes:even	$nat(p) \wedge 2 \times p = n \to even(n)$	DP	no
evenpp:evenp	$even(square(p)) \to even(p)$	NP	no
even:twotimes	$even(n) \to \exists p : nat(p) \wedge n = 2 \times p$	DP	no
npq:nnppqq	$n = p \times q \to$ $square(n) = square(p) \times square(q)$	BF	no
sqr2:4	$square(2) = 4$	DP	yes
times:injective:second	$(n+1) \times p = (n+1) \times q \to p = q$	DP	yes
simplify:by2	$4 \times n = 2 \times p \to 2 \times n = p$	BF	no
even:div2	$even(n) \to divisor(2,n)$	DP	yes
sqrt2:irrational	$nat(p) \wedge nat(q) \wedge coprime(p,q) \to$ $\neg square(p) = 2 \times square(q)$	PG	yes

Table 1: Summary of the obtained proof results. Shorthands for the column LLM: DP: directly proven by Claude, NP: not proven by Claude, BF: proven by Claude with back and forths, and PG: proof was given to Claude. A *yes* in the ATP column means a proof was found by one of our two provers.

Lemma or theorem	Shorthand	LLM	ATP
even:succ:odd	$even(p) \to odd(p+1)$	DP	yes
odd:succ:even	$odd(p) \to even(p+1)$	DP	yes
nat:even:disj:odd	$nat(p) \to even(p) \vee odd(p)$	DP	yes
odd:negation:even	$odd(n) \to even(n)$ *fails*	DP	yes
even:types	$even(n) \to nat(n)$	DP	yes
odd:types	$odd(n) \to nat(n)$	DP	yes
odd:plus:odd	$odd(m) \wedge odd(n) \to even(m+n)$	DP	no
even:plus:odd	$even(m) \wedge odd(n) \to odd(m+n)$	DP	yes
odd:times:odd	$odd(m) \wedge odd(n) \to odd(m \times n)$	DP	no
odd:square:odd	$nat(p) \wedge odd(p) \to odd(p^2)$	DP	yes
evenpp:evenp	$nat(p) \wedge even(square(p)) \to even(p)$	PH	yes
sqrt2:irrational	$nat(p) \wedge nat(q) \wedge coprime(p,q) \to$ $\neg square(p) = 2 \times square(q)$	PG	no

Table 2: Summary of the obtained proof results for auxiliary lemmas: DP: directly proven by Claude, PH: proven by Claude with given hints, and PG: proof was given to Claude

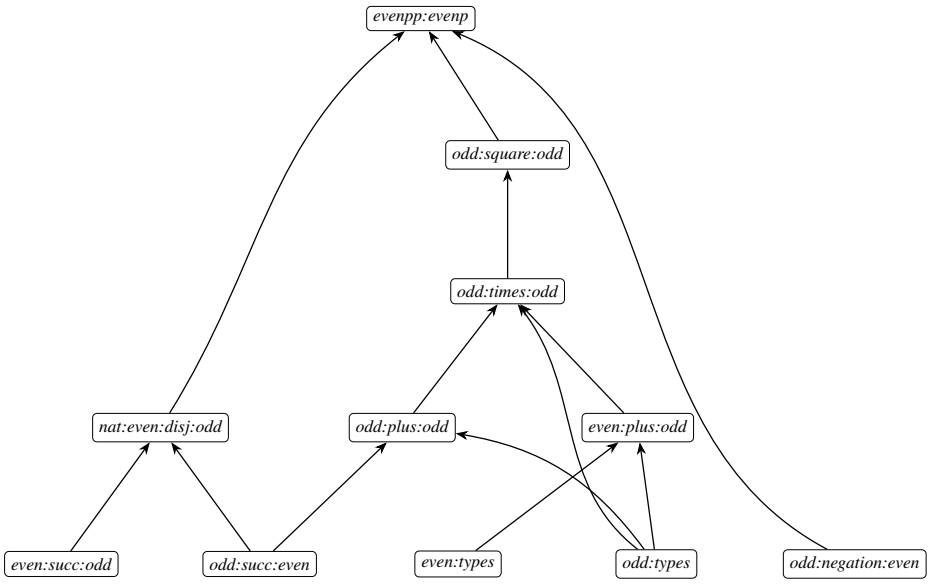

Figure 2: Dependency graph of the auxiliary lemmas of Table 2. An arrow $A \to B$ means A is used in the proof of B.

> Either p is even or odd. If p is even, we are done. Otherwise p is odd so p^2 is odd, hence $even(p^2)$ is false. Thus $nat(p) \wedge even(p^2)$ is false and the implication is true. So in all cases, the implication is true.

Claude generates the proof following our suggestion. Finally LPTP validates the whole proof file. Note that the ATPs were not able to prove the main theorem, contrary to the previous experiment. It is likely that they need more time to deal with a larger search space due to the increased number of lemmas. Table 2 presents the results and Figure 2 shows the dependency graph of the proof.

5 Back to Prolog

We now have an LPTP proof that $\sqrt{2}$ is irrational. So on a conceptual level, we know that solving the Prolog query of Section 1 is hopeless. Within the LPTP framework, let's take a closer look by first introducing the Prolog rule:

```
sqrt2_is_rational :-
    nat(S),plus(P,Q,S),gcd(P,Q,s(0)),times(P,P,P2),times(Q,Q,Q2),plus(Q2,Q2,P2).
```

In the meantime, Opus 4.6 is out (released early February 2026), and we switch to this new version in *Code mode*. We give Claude access to the *full Prolog code* of LPTP. Contrary to the *Chat mode*, Claude is able to call the proof checker itself, and we allow this capability. We want to prove an operational property of our Prolog code, so we create a `CLAUDE.md` file containing the following *Problem specification*:

> The predicate 'sqrt2_is_rational/0' is defined in 'sqrt2.pl'. This file also uses predicates from the files 'nat.pl' and 'gcd.pl' present in the directory. The file 'sqrt2_v2_no_gap.pr' contains a valid LPTP proof that the square root of 2 is not a rational. This proof should help to prove with LPTP that the query '?- sqrt2_is_rational.' does not succeed.

Furthermore, we point out that a bridge is needed between the logical definition of `coprime(p,q)` used in the hypothesis of Theorem [*sqrt2:irrational*] and the condition `gcd(P,Q,s(0))` appearing in the clause above (actually, the conditions are equivalent). Claude presents a first plan. Initially it wants to prove that *fails sqrt*2*_is_rational*. We explain that, as termination of `sqrt2_is_rational` can not be established, we do not have the usual dichotomy between *success* and *failure* (Axiom 7 of Section 2). The best we can hope is a proof of $\neg$ *succeeds sqrt*2*_is_rational* (Axiom 6). As Claude has access to the Prolog code of LPTP, it does use this access to have a better understanding of the LPTP syntax and of the LPTP proof checking process. Claude is able to generate three bridging lemmas and to prove that `sqrt2_is_rational` can not succeed by contradiction.

6 Related Work

There are a few Prolog verification frameworks, see, *e.g.*, [1, 3, 5, 15] and more recently [4]. Most of them aim at *paper-and-pencil* proofs and do not provide any implemented proof checker. Of course, as the most recent paper was published ten years ago, none of these papers reports any form of interaction with an LLM.

A comparison of seventeen proof assistants used in formal mathematics is provided in [24] through their formalization of a proof of the irrationality of $\sqrt{2}$. A proof in each system is presented, highlighting differences in logical foundations, syntax, automation, and usability. The main goal of the book is to illustrate the diversity of formal reasoning systems and to evaluate the state of proof assistant technology in the early 2000s. The included systems satisfy two criteria: they are designed, or used, for the formalization of mathematics and they are better, in at least one dimension, than all other systems in the collection. We note that LPTP is not considered. This work, published 20 years ago, does not of course report any interaction with an LLM.

Recently, we showed in [13] how one can easily plug any first order logic (FOL) automated theorem prover (ATP) into LPTP. We observe that inside the LPTP interactive development environment, namely Emacs, the FOL axioms are hardwired within tactics. Going back to the FOL formalization of the operational semantics of Prolog described in [20], and translating the axioms in FOF, a human-readable syntax for FOL [21], we can apply any *off-the-shelf* FOF-compatible ATP. We applied this strategy to the whole LPTP library, and obtained a success rate of about 80% with a 20 seconds timeout, running two of the most successful ATPs. The main advantage of this approach is that we can run the freely available ATPs locally on our machines, without relying on a non-free external provider. Moreover, the experiments are easily reproducible. On the other hand, we get resolution proofs (more precisely superposition calculus proofs) which cannot be directly rewritten as natural deduction proofs as needed for LPTP. So we cannot proof-check these proofs with LPTP. We experimented the approach on our $\sqrt{2}$ example. The results appear in the ATP column of Tables 1 and 2 and are summarized in the conclusion.

Back to our case study, we have illustrated how a formal proof checker, LPTP, can be combined with an LLM (in our experiment Claude) in order to arrive at a successful (partial) automatization of a theorem proving process. In particular, we use the LLM to generate proof steps whose correctness is subsequently checked by the proof checker. This is known as proof-step generation (*e.g.*, [8, 16]) and contrasts somewhat with other approaches that try to generate the proof as a whole (*e.g.*, [6, 9]).

Our observations are in line with the recent literature where similar combinations of language models and proof checkers are explored using a variety of approaches. Giving a complete overview of existing and ongoing work is outside the scope of the current paper, but we can cite the following. For example, in [25] a language model (DeepSeek-Prover-V1.5) is introduced that is specifically designed and trained

for doing theorem proving in Lean 4. Being a model specifically trained on formal languages, it is in sharp contrast with other approaches using an off-the-shelf language model, including our own. In [23], the authors introduce COPRA a system that uses an off-the-shelf LLM (GPT-4) and in-context learning to let the model generate proof steps directly written in the formal language of Rocq or Lean. COPRA uses a backtracking search methodology to construct the proof in a step-by-step manner, using feedback from the proof checker to construct the prompt for the next step. Likewise, [22] also uses a general-purpose model (GPT-4) but proposes to use natural language as an intermediate language. An interactive environment (called *Pétanque*) provides two proof tactics that allow to convert the internal reasoning to Rocq code. In a somewhat similar way, the Hermes framework [14] couples informal reasoning by an LLM with a translator module that allows to translate this reasoning into Lean code and a prover module (such as Goëdel-Prover-V2) for verification.

7 Conclusion

We report in this paper one of the very first interactions – to the best of our knowledge – between LPTP (Logic Program Theorem Prover) and an LLM. We have chosen Claude from Anthropic and did the experiment with Opus 4.5. We fed the LLM with the LPTP user manual in PDF format and two basic LPTP libraries (Prolog code and LPTP proofs in text format), one for Peano numbers and the other one about lists. We have selected a small classical problem among the proof assistant community, namely the irrationality of $\sqrt{2}$. This problem has been the running example of a whole book [24]. We did not find any LPTP proof of the irrationality of $\sqrt{2}$ on the Internet.

While we had to explicitly give the proof structure of the two main results – namely *sqrt2:irrational* and *evenpp:evenp* –, the LLM was able to prove some simple properties, sometimes proposing and proving new auxiliary lemmas. We stated the properties either in the LPTP language or in natural language (French in our case). Sometimes after a few back and forths between the LLM and LPTP, but sometimes directly, the lemmas generated by Claude were approved by the proof checker.

Beyond the LLM, we also leveraged off-the-shelf first-order ATPs such as Vampire and E by translating the LPTP axiomatization into FOF [21, 13]. They proved 5 out of 10 lemmas in the intermediate version (Section 3) and 9 out of 12 in the final, finer-grained version (Section 4). ATPs and the LLM play complementary roles: ATPs are free, local, fast and reproducible, but their resolution-based proofs cannot be easily rewritten as natural deduction and thus cannot directly enrich the LPTP library.

As already implied by the literature (*e.g.*, [11, 23]), the proof checker is the key element to eradicate LLM's hallucinations. It allows the automatic and efficient classification of the LLM's answers: either *correct* – the proof checker validates the proposed proof – or *wrong* – the proof checker does not validate the proposed proof and reports the first *incorrect derivation step* within the proposed proof –. The ability to point out the first step in the proof that is not correct is a great help for the LLM. Note also that once a proof is checked, the corresponding result can be safely added to the LPTP library, without compromising its logical consistency. This last point is crucial as the introduction of a false result in the library would allow the proof of *any* statement.

The interactions between LPTP, an LLM, and an LP developer leading to a semi-LLM-generated LPTP proof of the irrationality of $\sqrt{2}$ is the main contribution of this work[1]. It paves the way to an AI-enhanced approach for proving LP/Prolog properties, as the relevance of LLMs will continue to grow in the future. We plan to evaluate the combination LPTP/LLM on a set of Prolog verification problems.

[1]The data from the experiment are available at `https://github.com/FredMesnard/LPTP-LLM.git`.

References

[1] K. R. Apt & E. Marchiori (1994): *Reasoning about Prolog programs: from modes through types to assertions*. *Formal Aspects of Computing* 6(6), pp. 743–765.

[2] K. L. Clark (1978): *Negation as Failure*. In H. Gallaire & J. Minker, editors: *Logic and Databases*, Plenum Press, New York, pp. 293–322.

[3] Pierre Deransart (1993): *Proof methods of declarative properties of definite programs*. *Theor. Comput. Sci.* 118(2), p. 99–166, doi:10.1016/0304-3975(93)90107-5.

[4] W. Drabent (2016): *Correctness and Completeness of Logic Programs*. *ACM Trans. Comput. Log.* 17(3), p. 18, doi:10.1145/2898434.

[5] G. Ferrand & P. Deransart (1993): *Proof Method of Partial Correctness and Weak Completeness for Normal Logic Programs*. *J. Log. Program.* 17(2/3&4), pp. 265–278, doi:10.1016/0743-1066(93)90033-D.

[6] Emily First, Markus N. Rabe, Talia Ringer & Yuriy Brun (2023): *Baldur: Whole-Proof Generation and Repair with Large Language Models*. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, Association for Computing Machinery, New York, NY, USA, p. 1229–1241, doi:10.1145/3611643.3616243.

[7] ISO/IEC 13211-1 (1995): *Information Technology – Programming Languages – Prolog – Part 1: General Core*.

[8] Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu & Mateja Jamnik (2022): *Thor: wielding hammers to integrate language models and automated theorem provers*. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Curran Associates Inc., Red Hook, NY, USA.

[9] Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu & Guillaume Lample (2023): *Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs*. arXiv:2210.12283.

[10] J. W. Lloyd (1987): *Foundations of Logic Programming*. Springer-Verlag.

[11] Lachlan McGinness & Peter Baumgartner (2024): *Automated Theorem Provers Help Improve Large Language Model Reasoning*. In Nikolaj Bjørner, Marijn Heule & Andrei Voronkov, editors: *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*, *EPiC Series in Computing* 100, EasyChair, pp. 51–69, doi:10.29007/2n9m. Available at `/publications/paper/vzpW`.

[12] Fred Mesnard, Thierry Marianne & Étienne Payet (2024): *Automated Theorem Proving for Prolog Verification*. In Nikolaj S. Bjørner, Marijn Heule & Andrei Voronkov, editors: *LPAR 2024 Complementary Volume*, *Kalpa Publications in Computing* 18, EasyChair, pp. 137–151, doi:10.29007/C25R.

[13] Fred Mesnard, Thierry Marianne & Étienne Payet (2026): *Automated Theorem Proving for Prolog Verification*. *Electronic Proceedings in Theoretical Computer Science* 439, p. 469–481, doi:10.4204/eptcs.439.32.

[14] Azim Ospanov, Zijin Feng, Jiacheng Sun, Haoli Bai, Xin Shen & Farzan Farnia (2025): *HERMES: Towards Efficient and Verifiable Mathematical Reasoning in LLMs*, doi:10.48550/arXiv.2511.18760. ArXiv:2511.18760 [cs].

[15] D. Pedreschi & S. Ruggieri (1999): *Verification of Logic Programs*. *J. Log. Program.* 39(1-3), pp. 125–176, doi:10.1016/S0743-1066(98)10035-3.

[16] Stanislas Polu & Ilya Sutskever (2020): *Generative Language Modeling for Automated Theorem Proving*. arXiv:2009.03393.

[17] Balaji Rao, William Eiers & Carlo Lipizzi (2025): *Neural Theorem Proving: Generating and Structuring Proofs for Formal Verification*. arXiv:2504.17017.

[18] R. F. Stärk (1995): *First-order theories for pure Prolog programs with negation*. *Arch. Math. Log.* 34(2), pp. 113–144, doi:10.1007/BF01270391.

[19] R. F. Stärk (1996): *Total Correctness of Logic Programs: A Formal Approach*. In R. Dyckhoff, H. Herre & P. Schroeder-Heister, editors: *ELP'96*, *LNCS* 1050, Springer, pp. 237–254, doi:10.1007/3-540-60983-0_17.

[20] Robert F. Stärk (1998): *The theoretical foundations of LPTP (a logic program theorem prover)*. *The Journal of Logic Programming* 36(3), pp. 241–269, doi:10.1016/S0743-1066(97)10013-9.

[21] G. Sutcliffe (2023): *The logic languages of the TPTP world*. *Log. J. IGPL* 31(6), pp. 1153–1169, doi:10.1093/JIGPAL/JZAC068.

[22] Laetitia Teodorescu, Guillaume Baudart, Emilio Jesús Gallego Arias & Marc Lelarge (2024): *NLIR: Natural Language Intermediate Representation for Mechanized Theorem Proving*. In: *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*. Available at `https://openreview.net/forum?id=QzOc0tpdef`.

[23] Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin & Swarat Chaudhuri (2024): *An In-Context Learning Agent for Formal Theorem-Proving*. In: *First Conference on Language Modeling*. Available at `https://openreview.net/forum?id=V7HRrxXUhN`.

[24] F. Wiedijk, editor (2006): *The Seventeen Provers of the World, Foreword by Dana S. Scott*. *Lecture Notes in Computer Science* 3600, Springer, doi:10.1007/11542384.

[25] Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo & Chong Ruan (2025): *DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search*. In: *The Thirteenth International Conference on Learning Representations*. Available at `https://openreview.net/forum?id=I4YAIwrsXa`.