Case Study: Solving P-99 with LPTP and an LLM

F. Mesnard T. Marianne É. Payet W. Vanhoof

LIM, Université de La Réunion

Université de Namur

ICLP 2026

Context & Motivation

- **P-99**: a classic set of Prolog exercises (Werner Hett, ∼2000)
- **Vibe-coding**: LLM generates code from informal specs
  - Easy and fast, but functional correctness issues
- **Vericoding**: LLM also produces machine-checkable proofs
  - Existing work: Dafny, Verus, Lean — none targeting Prolog
- **Our approach**: solve P-99 exercises with **formal guarantees**
  - combine vibe-coding with vericoding
  - use **Claude** (Anthropic, Opus 4.6) with **LPTP** (Logic Program Theorem Prover)
  - just by prompting!

What Do We Mean by "solve P-99"?

For each exercise, the LLM must:

1. **Generate** pure Prolog code (no cut, no built-ins beyond =/2, \+, if-then-else)
2. **Generate & run** a test file
3. **State & prove** in LPTP:
   - Type preservation
   - Groundness
   - Termination
   - Uniqueness & Existence
   - Functional correctness (with human hints)

Human **checks** every output
LPTP **proof-checks** every proof

LPTP in a Nutshell

- Stärk's **Logic Program Theorem Prover** (1995–1998)
- First-order theory for pure Prolog with negation and unification with occurs check
- Proof language with tactics: induction, completion, case analysis, unfolding, auto
- Library: naturals (Peano), lists, ...
- Proof-checker: $\sim$ 7kLOC of Prolog

Experimental Setup

- **LLM**: Claude Opus 4.6 in Code mode (Feb. 2026)
- **Prolog**: SWI-Prolog with `occurs_check` flag set to `true`
- **Proof checker**: LPTP v1.06
- **Peano naturals**: $0, s(0), s(s(0)), \ldots$
- **Prompt structure**: `CLAUDE.md` + `lptp-reference.md` + one example (P01)
- Typical prompt: *"Read CLAUDE.md, lptp-reference.md, and solve P02"*
- Time per exercise: 15 min (P01) to several hours (P35)

Repository Structure

Each exercise `Pxx` produces:

File	Role
`pXX.pl`	Prolog source code
`pXX.gr`	Ground representation for LPTP
`pXX.pr`	Properties & LPTP-checked proofs
`pXX_test.pl`	Runtime tests (SWI-Prolog)
`pXX-experiment-report.md`	Report
`CLAUDE.md`	Problem specification

Everything available at `github.com/FredMesnard/LPTP-LLM-P99`

Overview of Results

33 exercises solved (P01–P28 + P31–P35 = 37.5% of P-99)

Metric	**Count**
Logic procedures	58
Prolog clauses	112
Lines of Prolog code	150
Runtime tests	∼500
Lemmas proved	257
Lines of proof	∼11 800

Proof-to-code ratio ≈ 78:1

P01 — Specification & Code

P01 (*) Find the last element of a list.

```
?- my_last(X,[a,b,c,d]).
X = d
```

Prolog code (generated by Claude):

```
my_last(X, [X]).
my_last(X, [_|L]) :- my_last(X, L).
```

Solved in ∼15 minutes with runtime tests.

P01 — Properties (10 lemmas)

#	Statement
1	$\forall x, l.\ \mathit{list}(l) \Rightarrow \mathit{terminates}\ \mathit{my\_last}(x, l)$
2	$\forall x, l.\ \mathit{my\_last}(x, l) \Rightarrow \mathit{list}(l)$
3	$\forall x, l.\ \mathit{my\_last}(x, l) \wedge \mathit{gr}(l) \Rightarrow \mathit{gr}(x)$
4	$\forall x, l.\ \mathit{my\_last}(x, l) \Rightarrow \mathit{member}(x, l)$
5	$\forall x, y, l.\ \mathit{my\_last}(x, l) \wedge \mathit{my\_last}(y, l) \Rightarrow x = y$
6	$\forall z, l.\ \mathit{list}(l) \Rightarrow \exists x.\ \mathit{my\_last}(x, [z \mid l])$
7	$\forall l.\ \mathit{list}(l) \wedge l \neq [\,] \Rightarrow \exists x.\ \mathit{my\_last}(x, l)$
8	$\forall x, l.\ \mathit{my\_last}(x, l) \Rightarrow \exists l_1.\ \mathit{append}(l_1, [x], l)$
9	$\forall x, l_1, l.\ \mathit{list}(l_1) \wedge \mathit{append}(l_1, [x], l) \Rightarrow \mathit{my\_last}(x, l)$
10	$\forall x, l_1, l.\ \mathit{append}(l_1, [x], l) \Rightarrow \mathit{my\_last}(x, l)$

Properties 8 + 10 give a **characterization** of `my_last/2` w.r.t. `append/3`.
Human hint: *"What is the connection with `append/3`?"*

P31 — Specification & Code

P31 (**) Determine whether a given integer number is prime.

?- is_prime(7).

Yes

Three predicates generated by Claude:

```
divides(D, D).
divides(D, N) :- plus(D, M, N), divides(D, M).

no_factor(0, _, _).
no_factor(s(K), D, N) :- \+ divides(D, N), no_factor(K, s(D), N).

is_prime(s(s(X))) :- no_factor(X, s(s(0)), s(s(X))).
```

P31 — Properties

Claude proved properties for `divides/2`, `no_factor/3` and `is_prime/1` (termination, types, groundness, uniqueness, existence).

Key results — soundness and completeness of `is_prime/1`:

Soundness:

$$\forall n.\ [nat(n) \wedge is\_prime(n)] \ \Rightarrow\ [\exists x.\ n{=}s(s(x)) \ \wedge\ (\forall d.\ 1{<}d{<}s(x) \Rightarrow fails\ divides(s(d), n))]$$

Completeness:

$$\forall n, x.\ [nat(n) \wedge n{=}s(s(x)) \wedge (\forall d.\ 1{<}d{<}s(x) \Rightarrow fails\ divides(s(d), n))] \ \Rightarrow\ is\_prime(n)$$

Human hint: we asked for such properties and gave the idea of their formalization in natural language.

P35 — Specification & Code (1/2)

P35 (**) Determine the prime factors of a given positive integer.
Construct a flat list containing the prime factors in ascending order.

?- prime_factors(315, L).
L = [3,3,5,7]

```
% quot(D, N, Q) - quotient Q = N/D
quot(D, D, s(0)).
quot(D, N, s(Q)):- plus(D, M, N), quot(D, M, Q).

% smallest_factor(N, D, K, F)
smallest_factor(N, D, K, D) :- divides(D, N).
smallest_factor(N, D, s(K), F):- \+ divides(D, N), smallest_factor(N, s(D), K, F).
```

P35 — Code (2/2)

```
prime_factors(s(0), []).
prime_factors(s(s(X)), [F|L]) :-
    smallest_factor(s(s(X)), s(s(0)), X, F),
    quot(F, s(s(X)), Q),
    prime_factors(Q, L).
```

Auxiliary predicates for functional correctness:

```
ordered([]).
ordered([_]).
ordered([X,Y|L]) :- X @=< Y, ordered([Y|L]).

product([], s(0)).
product([X|L], P) :- product(L, P1), times(X, P1, P).
```

Most complex exercise: several hours of interaction.

P35 — Properties for `quot/3` and `smallest_factor/4`

#	Predicate	Property
1	`quot/3`	Termination
2		Type preservation ($nat(q)$)
3		Strict bound: $q < n$
4		Correctness: $d \times q = n$
5		Positive: $\exists q_0.\ q = s(q_0)$
6		Success (if d divides n)
7		Uniqueness
8	`sm._factor/4`	Termination
9		Type preservation
10		Lower bound: $f \geq 1$
11		Lower bound 2: $f \geq 2$
12		f divides n
13		Completeness
14		Minimality: $f \leq g$ for any factor $g \geq d$
15		Uniqueness
16		$is\_prime(f)$

P35 — Properties for `divides/2` and `prime_factors/2`

#	Predicate	Property
17	`divides/2`	Self: $\mathit{divides}(d, d)$
18		Sum: $d \mid n \wedge d \mid b \Rightarrow d \mid (n + b)$
19		Times factor: $d \mid q \Rightarrow d \mid (f \times q)$
20		Transitivity
21	`prime_factors/2`	Termination
22		Types: $\mathit{list}(l)$
23–24		All members are *nat*
25		Head info: head divides n, is ≥ 2
26		**Product**: $\mathit{product}(l, n)$
27		**Ordered**: $\mathit{ordered}(l)$
28		**All prime**: $\forall z \in l.\ \mathit{is\_prime}(z)$
29		**Existence**: $\forall n.\ \mathit{nat}(n) \Rightarrow \exists l.\ \mathit{prime\_factors}(s(n), l)$
30		**Uniqueness**

Properties 26–30 together constitute an **LP-based proof of the Fundamental Theorem of Arithmetic**.

Human–LLM Interaction

- **Vibe-coding part** (code + tests): handled *easily* by the LLM
- **Verification part**: more challenging
  - Types, groundness, termination: largely autonomous
  - Functional correctness: **human hints needed**
    - e.g. "What is the connection with `append/3`?"
    - Sometimes full property statements in natural language
- Claude discovered proof techniques on its own (e.g. strengthen-then-weaken for induction)
- Claude even inspected LPTP's Prolog source code to debug proofs
- All outputs manually checked; proofs certified by LPTP

MCP Server for LPTP

Model Context Protocol (Anthropic, Nov. 2024) — open standard for tool integration.

`atp-lptp-mcp` server (TypeScript SDK, available on npm):

- Exposes LPTP tactics as tools: `tactic_ind`, `tactic_comp`, `tactic_case`, `apply_tactic`, ...
- `verify_lemma` / `check_proof`: certify proof steps
- `get_lptp_grammar`: embeds LPTP syntax

Benchmark (P14–P24): Claude Code (Sonnet 4.6) and Gemini 3.1 Pro

- Gemini: more diverse property generation
- Claude: only one producing valid proofs for P14–P24
- All proofs certified within 40–60 min per batch

Related Work

Vibe-coding	Sarkar & Drosos 2025; Karpathy 2025 — widespread but correctness issues
Vericoding	Bursuc, Tegmark et al. 2025 — Dafny/Verus/Lean benchmarks; no Prolog
Agentic coding	Hassan et al. 2025 — autonomous agents; $\sim$50% PR rejection rate
Formal LP verification	Cousot & Cousot; Hermenegildo et al. — abstract interpretation tools

Our positioning: first work combining an LLM with a Prolog-specific theorem prover (LPTP) to produce certified proofs of Prolog programs.

Lessons Learned

- The LLM grasped LPTP's formalism **remarkably fast**
- Combining LLM + proof-checker **eliminates hallucinations**: either proofs are certified, or they have gaps
- Functional correctness remains **the hardest part** — requires human guidance
- The `lptp-reference.md` file (lessons learned) significantly **speeds up** subsequent exercises
- **Scalability** is an open question

Conclusion & Future Work

Summary:

- 33 P-99 exercises solved with formal proofs via Claude + LPTP
- 58 logic procedures — 508 tests
- 257 lemmas, ∼11 800 proof lines — all machine-checked
- Includes an LP-based proof of the Fundamental Theorem of Arithmetic
- MCP server enables multi-LLM comparison

Future work:

- Finish P-99!
- Algorithmic generation of LPTP proofs for standard properties (*e.g.*, groundness)
- Integrate automated theorem provers (E, Vampire) as backends
- Scalability

Thank you!

github.com/FredMesnard/LPTP-LLM-P99