Solving P-99 at the age of LLM
Examples and pedagogical considerations

Fred Mesnard[1] Thierry Marianne[1] Étienne Payet[1] Wim Vanhoof[2]

[1]LIM, Université de La Réunion, France
[2]Université de Namur, Belgium

Fourth Prolog Education Workshop
PEG 2026

This work is partly funded by the European Union under the INTERREG VI "Mise en réseau" program

Outline

Context: LLMs and logic programming – VibeVeriCoding P-99

- LLMs are increasingly capable of writing and reasoning about code
- Two emerging AI-assisted programming methodologies:

  Vibe-coding informal spec → LLM → code (+ optional tests)

  Vericoding formal spec → LLM → code + machine-checked proof
- **Our experiment**: chain both on Prolog, with **LPTP** as proof checker
  1. informal spec → LLM → code (= formal spec in LP) + tests *(vibe-coding)*
  2. formal spec (= code in LP) → LLM → (code +) machine-checked properties *(vericoding)*

P-99: Ninety-Nine Prolog Problems

- Classic set of 88 Prolog exercises by Werner Hett (25+ years old)
- We solve the **first 33 exercises** (P01–P28 + P31–P35 = 37.5%) *by prompting Claude*
- We used **Claude Code**, Opus 4.6 model

What "solving" means:

For each exercise:

1. LLM: generate Prolog code + test file; run tests
   Us: manual review and double-check
2. LLM: state and prove *types*, *groundness*, *termination*, *existence*, *uniqueness*, *functional correctness* with **LPTP**
   Us: manual review and double-check

NB: P-99 numbering skips 29–30: the list section ends at P28, arithmetic starts at P31, ...

The LP fragment

- **Pure Prolog** with:
  - equality =/2 on finite trees
  - negation as failure \+
  - if/then/else construct
  - *no builtins*
- Natural numbers in **Peano notation**: 2 as `s(s(0))`
- Unification *with occurs-check*
- Operational semantics: otherwise ISO-Prolog
- **LPTP** (Logic Program Theorem Prover, Stärk 1998) for verification:
  - specification language
  - automatic proof checker

LPTP in a nutshell

- Proof checker for logic programs (Stärk, 1998)
- Works w.r.t. a FOL extension of Clark's completion
- For each predicate R, three viewpoints in $\hat{\mathcal{L}}$:
  - $R^s(\vec{x})$ — $R(\vec{x})$ *succeeds*
  - $R^f(\vec{x})$ — $R(\vec{x})$ *fails*
  - $R^t(\vec{x})$ — $R(\vec{x})$ *terminates*
- Axioms link them: success/failure exclusive, termination $\rightarrow$ decision, plus one induction schema per recursive LP definition
- **LPTP = FOL + 9 axioms**

LPTP today

- Originally integrated in Emacs
- Now available as an **autonomous web-based IDE**:

  `https://ciao-lang.org/playground/lptp.html`
- Combines Ciao-Prolog, Ciao-PP (abstract interpreter), and LPTP
- Great tool for teaching LP from 1st year to master level
  Nothing to install

Claude setup: the CLAUDE.md file

GitHub:

Repository structure:

- `P99/CLAUDE.md` — global instructions
- `P99/P-99.html` — problems
- `P99/lptp-reference.md` — LPTP tips (maintained by Claude)
- `P99/P01/` — directory and files for P01, given as an example

Per-exercise files:

- `CLAUDE.md` — problem description
- `pxx.pl` — Prolog code
- `pxx_test.pl` — test file
- `pxx.gr` — `.pl` → LPTP format
- `pxx.pr` — properties + proofs
- `pxx-experiment-report.md`

Prompt to solve P02: *"Read CLAUDE.md, lptp-reference.md, and solve the P02 exercise"*
Iterate

Statistics

Item	Count
Exercises solved (P01–P28 + P31–P35)	33 / 88
Logic procedures	58
Prolog clauses	112
Runtime tests	508
LPTP lemmas proved	257
Proof lines	11800

- Claude used negation as failure; *no* if/then/else
- Time per exercise: 15 min (P01) to several hours (P35)
- Every file manually reviewed; proof hints needed only for functional correctness

Observations

- **Vibe-coding part** (code + tests): handled easily by Claude
  - Pure Prolog constraint respected
  - Generated code is often *different* from web solutions
- **Vericoding part** (properties + proofs): more challenging
  - Elementary properties (types, groundness, termination): Claude handles well
  - Functional correctness properties: hints needed
- **Notable:** Claude examined LPTP source code to fix problematic proofs!
- Proof checker *eliminates hallucinations* — Claude can fix its errors

At the age of LLM, the proof checker is the *killer app* of interactive theorem proving!

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

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

Claude's Prolog code:

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

Properties proved (10):

- Termination, type preservation, groundness propagation, membership
- Uniqueness, existence
- Connection to append/3 (asked by us):

$$\forall x, l.\ [\texttt{my\_last}(x, l) \Leftrightarrow \exists l_1.\ \texttt{append}(l_1, [x], l)]$$

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]
```

Claude's Prolog code:

```
quot(D, D, s(0)).
quot(D, N, s(Q)) :- plus(D, M, N), quot(D, M, Q).
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).

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).
```

P35: prime factorisation (properties)

30 properties proved for `quot/3`, `smallest_factor/4`, `prime_factors/2`

Key results (assuming `prime_factors`(n, l) succeeds, n Peano integer):

1. **Product:** `product`(l, n)
2. **Ordered:** l is in ascending order
3. **All prime:** every element of l is prime
4. **Existence:** $\forall n > 0,\ \exists l$
5. **Uniqueness:** l is unique

$\Rightarrow$ an LP-based proof of the **Fundamental Theorem of Arithmetic**

Towards *certified LP*

- LP is human readable, even by non-specialists
- LP is both a specification language and an executable language
- Certified LP = LP code + machine-checked properties
- Largely Abstract-Interpretation/LLM-automatable

Potential applications:

- **Critical software components** — provable specs, usable as test oracles for imperative/OO implementations
- **Rules as Code** / **Law as Code** (OECD, EU) — official machine-consumable versions of regulations and laws

Ideas for teaching certified LP: two entry points

Bachelor level

- With LPTP: propositional logic $\rightarrow$ FOL $\rightarrow$ LP
  Use *one* IDE
- Natural deduction demystifies formal proofs
- Declarative LP first, then Prolog
- Finally LLM for simple code, tests, and first automated LPTP proofs
  where *students review all outputs*
- Emphasis on intuition and critical thinking
- Closed book exams with given cheat sheets

Master level

- Abstract interpretation for invariant generation
- LLM + ATPs for more complex LPTP proofs
- Emphasis on semantics – syntactic effort computer-assisted
- Small-group certified LP projects with oral assessment

Remarks on the pedagogy

- LPTP is both a proof checker and an IDE for propositional/FOL proofs
- **Once LLM is introduced, it is hard to go back**
  $\Rightarrow$ manual exercises are still necessary to build understanding and intuition
- A similar methodology could be applied to other paradigms (*e.g.*, Dafny)

Conclusion and perspectives

Main results:

- 2026-05: 33 exercises solved, 58 procedures, 257 lemmas, 508 tests, 11 800 proof lines
- 2026-07: 87 exercises solved, 558 procedures, 1262 lemmas, 1132 tests, 43 932 proof lines
- Vibe-coding part: easy for Claude
- Vericoding part: manageable, except for functional correctness properties (hints needed)
- *LPTP/LLM combination rejects hallucinations and helps fix LLM errors*
- *LLM as a natural language interface for LPTP*

Open questions:

- **Scalability** to larger programs?
- How to **teach** certified LP at the LLM age
  Certified LP: now largely mechanized
  Bottleneck: People?

Don't miss these TC talks at ICLP 2026:

- Monday 15:00 – Case study: proving $\sqrt{2}$ irrational with LPTP and an LLM
- Tuesday 17:30 – Case study: solving P-99 with LPTP and an LLM
- Tuesday 17:45 – Towards Relating Ciao Assertions and LPTP Theorems

Thank you!

`github.com/FredMesnard/LPTP-LLM-P99`

Please share your ideas, remarks and comments!

Some questions

- P31(**): determine whether a given integer number is prime
- Training-data contamination?
- LLM/Human effort
- Why Peano numbers & a pure fragment?
- Scalability & why stop at P35?
- Reproducibility & other models
- What do students actually learn?
- Assessment in the LLM age
- Curriculum: tested or proposed?
- Why LPTP, not Lean/Rocq/Isabelle?
- Trusted Computing Base?
- if/then/else vs negation-as-failure

P31(**): determine whether a given integer number is prime

Specification:

```
?- is_prime(7).
Yes
```

Claude's code: `divides/2`, `no_factor/3`, `is_prime/1`

```
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))).
```

Key properties:

- Soundness: `is_prime`$(n) \Rightarrow n \geq 2$ and no factor in $[2, n-1]$
- Completeness: converse

Training-data contamination?

Objection: P-99 is 25+ years old, solutions are all over the web.

- Generated *code* is often **different** from web solutions:
  pure Prolog, Peano, no cuts ... unlike most of them
- Crucially, the **LPTP specs and proofs do not exist online** — that is our contribution
- Validity is **independent of provenance**: the checker decides, not the source

LLM/Human effort

- LLM: per exercise: 15 min (P01) to several hours (P35)
- Human: where the time goes:
  - design decisions and **reviewing** (code, test, statements)
  - hints for **functional-correctness** proofs

Key point: *reviewing* a machine-checked proof $\ll$ *producing* it:
does the statement make sense (at an intuitive level)?

Why Peano numbers & a pure fragment?

- LPTP semantics: Clark's completion, occurs-check, finite trees
- Peano notation $\Rightarrow$ structural **induction** available, clean arithmetic semantics
- No builtins / no cut $\Rightarrow$ every predicate has a **logical reading**
- Trade-off: *clarity & provability* vs. native efficiency

Native integers: **WIP**

Scalability & why stop at P35?

- P35 = end of the arithmetic section reached — a **boundary, not a wall**
- Cost grows: P35 took several hours; functional correctness needs hints
- Stated as an **open question** in the conclusion
- July 2026: we've solved all P99 exercises

Reproducibility & other models

- Model used: **Claude Opus 4.6** via Claude Code
- We offer an MCP (Model Context Protocol) connector
- Other model tried: Gemini
- LLMs are non-deterministic — but **validity does not depend on it** only LPTP-checked proofs count
- A repo is available as a starting point on GitHub (with only P01)

What do students actually learn?

Concern: if the LLM writes code, tests *and* proofs — deskilling?

- LLM does **not** replace understanding: students *review all outputs*
- Skills trained: reading specs, evaluating correctness, intuition about software verification
- **Manual exercises stay mandatory** — "hard to go back, so build the base first"
- Risk acknowledged; mitigated by **assessment design**

Assessment in the LLM age

- **Bachelor:** closed-book exams with *given* cheat sheets
- **Master:** small-group certified-LP projects with **oral** defence
- Oral assessment probes *genuine* understanding — hard to outsource

Curriculum: tested or proposed?

- The certified-LP **experiment** (all P99 exercises): **done**
- The **teaching curriculum**: a proposal

Why LPTP, not Lean/Rocq/Isabelle?

- Our goal is **LP program verification**: types, groundness, termination, existence, uniqueness, functional correctness ⇒ *certified LP*
- **LPTP is the native tool**: its object language *is* Prolog, its axioms *are* Prolog's operational semantics — math provers would have to re-implement that first, for no gain
- **Maths is a by-product**: P35 yields the Fundamental Theorem of Arithmetic *while verifying* `prime_factors/2`
- **Lightweight, agent-friendly checker**: in Code mode Claude reads the *full* LPTP source and calls the checker itself — impractical with the Lean/Rocq kernel

For *mathematics:* Lean/Rocq/Isabelle win (mathlib, hammers, more LLM training)
The choice depends on the goal — ours is **LP program verification**: LPTP wins

Trusted Computing Base?

Wikipedia: The *trusted computing base* (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system.

- What we trust:
  - the theory behind LPTP (FOL + 9 axioms, JLP Stärk 1998)
  - the LPTP checker (pure Prolog, written by Stärk)
  - the Prolog engine
- What we don't trust
  - the LLM

The proof-checker is small (7k Prolog loc) and fixed — independent of the LLM
Ultimately, LPTP's ND proofs are FOL, human-readable texts

if/then/else vs negation-as-failure

- The fragment **allows** if/then/else
- Yet Claude frequently chose **negation-as-failure** (\+)
- Plausible reasons: simpler logical reading for LPTP; matches its training distribution
- **No correctness impact** — both stay within the pure fragment
- NB: *the LPTP's if/then/else allows a cut (!) at the implementation level without compromising the proofs*