Automated Theorem Proving for Prolog Verification

Fred Mesnard   Thierry Marianne   Etienne Payet

LIM, université de La Réunion, France

ICLP 2025 – TC

Prolog verification

Total correctness = termination + partial correctness

- Termination – *many* papers, which tool?
  E.g., ours for pure Prolog
  NTI+cTI ranked 1st at TermComp since 2022

- Partial correctness – a few papers, which tool?
  LPTP: Logic Program Theorem Prover
  Robert Stärk, mid-1990's

Some LPTP examples

```
nat(0).                     add(0,Y,Y).
nat(s(X)) :- nat(X).        add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

Lemma [*add:existence*] $\forall x, y\, (\mathbf{S}\, \texttt{nat}(x) \to \exists z\ \mathbf{S}\, \texttt{add}(x, y, z))$.

Lemma [*add:uniqueness*]
$\forall x, y, z_1, z_2\, (\mathbf{S}\, \texttt{add}(x, y, z_1) \wedge \mathbf{S}\, \texttt{add}(x, y, z_2) \to z_1 = z_2)$.

Lemma [*add:term:1*] $\forall x, y, z\, (\mathbf{S}\, \texttt{nat}(x) \to \mathbf{T}\, \texttt{add}(x, y, z))$.

Lemma [*add:term:3*] $\forall x, y, z\, (\mathbf{S}\, \texttt{nat}(z) \to \mathbf{T}\, \texttt{add}(x, y, z))$.

Theorem [*add:commutative*]
$\forall x, y, z\, (\mathbf{S}\, \texttt{nat}(x) \wedge \mathbf{S}\, \texttt{nat}(y) \wedge \mathbf{S}\, \texttt{add}(x, y, z) \to \mathbf{S}\, \texttt{add}(y, x, z))$.

An LPTP proof: source and PDF

```
:- lemma(add:exist,
all [x,y]: succeeds nat(?x) => (ex z: succeeds add(?x,?y,?z)),
induction(
 [all x: succeeds nat(?x) => (all y: ex z: succeeds add(?x,?y,?z))],
 [step([],
   [],
   [succeeds add(0,?y,?y),
    ex z: succeeds add(0,?y,?z)],
   all y: ex z: succeeds add(0,?y,?z)),
  step([x],
   [all y: ex z: succeeds add(?x,?y,?z),
    succeeds nat(?x)],
   [ex z: succeeds add(?x,?y,?z),
    exist(z0, succeeds add(?x,?y,?z0),
     [succeeds add(s(?x),?y,s(?z0)) by sld],
     ex z1: succeeds add(s(?x),?y,?z1))],
   all y: ex z: succeeds add(s(?x),?y,?z))])).
```

Lemma 1 [*add:exist*] $\forall x, y\, (\mathsf{S}\, \mathsf{nat}(x) \to \exists z\ \mathsf{S}\, \mathsf{add}(x, y, z))$.

Proof.

Induction$_0$: $\forall x\, (\mathsf{S}\, \mathsf{nat}(x) \to \forall y\, \exists z\ \mathsf{S}\, \mathsf{add}(x, y, z))$.

Hypothesis$_1$: none. $\mathsf{S}\, \mathsf{add}(0, y, y)$. $\exists z\ \mathsf{S}\, \mathsf{add}(0, y, z)$.

Conclusion$_1$: $\forall y\, \exists z\ \mathsf{S}\, \mathsf{add}(0, y, z)$.

Hypothesis$_1$: $\forall y\, \exists z\ \mathsf{S}\, \mathsf{add}(x, y, z)$ and $\mathsf{S}\, \mathsf{nat}(x)$. $\exists z\ \mathsf{S}\, \mathsf{add}(x, y, z)$.

Let$_2$ z_0 with $\mathsf{S}\, \mathsf{add}(x, y, z_0)$. $\mathsf{S}\, \mathsf{add}(\mathsf{s}(x), y, \mathsf{s}(z_0))$ by sld.

Thus$_2$: $\exists z_1\ \mathsf{S}\, \mathsf{add}(\mathsf{s}(x), y, z_1)$.

Conclusion$_1$: $\forall y\, \exists z\ \mathsf{S}\, \mathsf{add}(\mathsf{s}(x), y, z)$. $\square$

Main LPTP papers

R. F. Stärk
First-order theories for pure Prolog programs with negation
Arch. Math. Log., 34(2):113–144, 1995

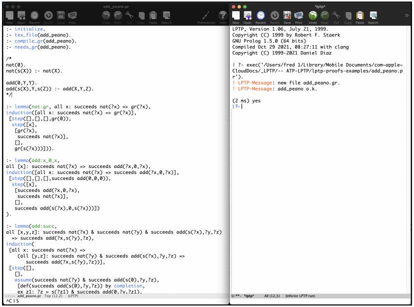
R. F. Stärk
Total correctness of logic programs: A formal approach
ELP'96, *LNCS* 1050, 237–254. Springer, 1996

R. F. Stärk
The theoretical foundations of LPTP
(a logic program theorem prover)
Journal of Logic Programming (JLP), 36(3):241–269, 1998

LPTP is also an interactive theorem prover (ITP)

- An Emacs user-interface
- Natural-deduction tactic-based ITP
- A *very limited* `auto` tactic
- A proof checker written in ISO-Prolog
- A proof manager based on TEX and HTML

Runs out of the box 25 years later:



The two languages of LPTP – ① the object language

$\boxed{\text{Pure Prolog (finite terms) with negation as failure}}$

- Let P be a pure logic program with negation and $\mathcal{L}$ the first-order language associated to P
- The *goals* of $\mathcal{L}$ are:

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

  s and t are terms, x is a variable and A is an atomic goal
- $\boxed{\text{Operational semantics: ISO-Prolog with the occurs check}}$

The two languages of LPTP – ② the specification language

Classical first order logic

- $\hat{\mathcal{L}}$ is the specification language of LPTP
- For each user-defined predicate symbol R, $\hat{\mathcal{L}}$ 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
- The *formulas* of $\hat{\mathcal{L}}$ are:

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

  where $\vec{t}$ is a sequence of n terms and R denotes a n-ary predicate symbol of $\hat{\mathcal{L}}$
- The semantics of $\hat{\mathcal{L}}$ is classical first order logic (FOL)
- For any of the user-defined logic procedure $R(\vec{x})$ in P, $D_R^P(\vec{x})$ denotes its Clark's *if-and-only-if* completed definition

The two languages of LPTP – ② the specification language

For defining the declarative semantics of LP, three syntactic operators **S**, **F** and **T** which map goals of $\mathcal{L}$ into $\hat{\mathcal{L}}$-formulas

Intuitively:

- **S**G means G *succeeds*
  The breadth-first left-to-right evaluation of G succeeds
  One or more infinite branches may exist
- **F**G means G *fails*
  The ISO-Prolog evaluation stops without any answer
  No infinite branch
- **T**G means G *terminates*
  The ISO-Prolog evaluation produces a finite number of answers then stops
  No infinite branch

IND(P) – from the JLP paper

The declarative semantics of P is IND(P), an always consistent theory which includes Clark's equality theory ...

IND(P), comprises the following axioms:

I. The axioms of Clark's equality theory CET:

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

II. Axioms for gr:

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

III. Uniqueness axioms (UNI):

6. $\neg(R^{\mathrm{s}}(\vec{x}) \wedge R^{\mathrm{f}}(\vec{x}))$

IV. Totality axioms (TOT):

7. $R^{\mathrm{t}}(\vec{x}) \rightarrow R^{\mathrm{s}}(\vec{x}) \vee R^{\mathrm{f}}(\vec{x})$

V. Fixed point axioms for user-defined predicates R:

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

IND(P) – from the JLP paper

... and induction

VIII. The simultaneous induction scheme for user-defined predicates: Let $R_1, \dots, R_n$ be user-defined predicates and let $\varphi_1(\vec{x}_1), \dots, \varphi_n(\vec{x}_n)$ be $\hat{\mathscr{L}}$-formulas such that the length of $\vec{x}_i$ is equal to the arity of R_i for $i = 1, \dots, n$. Let

$$closed(\varphi_1(\vec{x}_1)/R_1, \dots, \varphi_n(\vec{x}_n)/R_n)$$

be the formula obtained from

$$\forall \vec{x}_1(\mathbf{S}D^P_{R_1}[\vec{x}_1] \to R^s_1(\vec{x}_1)) \wedge \cdots \wedge \forall \vec{x}_n(\mathbf{S}D^P_{R_n}[\vec{x}_n] \to R^s_n(\vec{x}_n))$$

by replacing simultaneously all occurrences of $R_i(\vec{t})$ by $\varphi_i(\vec{t})$ for $i = 1, \dots, n$ and renaming the bound variables when necessary. Let

$$sub(\varphi_1(\vec{x}_1)/R_1, \dots, \varphi_n(\vec{x}_n)/R_n)$$

be the formula

$$\forall \vec{x}_1(R^s_1(\vec{x}_1) \to \varphi_1(\vec{x}_1)) \wedge \cdots \wedge \forall \vec{x}_n(R^s_n(\vec{x}_n) \to \varphi_n(\vec{x}_n)).$$

Then the simultaneous induction axiom is the following formula:

10. $closed(\varphi_1(\vec{x}_1)/R_1, \dots, \varphi_n(\vec{x}_n)/R_n) \to sub(\varphi_1(\vec{x}_1)/R_1, \dots, \varphi_n(\vec{x}_n)/R_n)$.

Correct and partially complete w.r.t. the Prolog semantics

ATP for LPTP?

Observation:

- Within LPTP, we prove properties of a Prolog program P using a natural-deduction tactic-based ITP where the axioms IND(P) of the theoretical framework are *hardwired*

Idea:

- Go back to FOL by translating the axioms IND(P) in TPTP FOF (*First Order Form*) and invoke *any* FOF-compatible theorem prover

Experimentation:

- Try with E and Vampire on the LPTP lib

Workflow of our experiment

- Requirements: the logic program P and the associated proof file
  We do not use the proofs, only the statements!
- If P depends on other logic programs, we include them
- If the associated proof file uses other proof files, we include them
- We build a target logic program P' and a target LPTP proof file
- Each property is compiled as a FOF conjecture possibly with its induction axiom and stored in a single file which also contains the logic theory IND(P') compiled as FOF axioms
- Previously processed FOF conjectures are converted as FOF axioms
  So we produce as many FOF files as there are properties in P'
- Both E and Vampire are applied to each FOF file with predefined time limits

Benchmarks

On a Mac Book Air M2, 400 properties from the LPTP library

Average success rate: 83% for a 1 min timeout

lib	#	E-1s	V-1s	EV-1s	E-10s	V-10s	EV-10s	E-60s	V-60s	EV-60s
nat	91	70%	88%	88%	76%	95%	95%	78%	97%	97%
gcd	11	45%	45%	45%	45%	45%	45%	45%	45%	45%
ack	3	33%	33%	33%	33%	33%	33%	33%	33%	33%
int	67	76%	82%	87%	79%	88%	90%	79%	91%	91%
list	84	75%	94%	94%	80%	96%	96%	81%	99%	99%
suffix	31	94%	100%	100%	94%	100%	100%	97%	100%	100%
reverse	25	72%	88%	88%	84%	100%	100%	84%	100%	100%
permut.	42	48%	71%	71%	60%	79%	81%	62%	86%	86%
sort	42	45%	62%	62%	50%	74%	74%	55%	76%	76%
merges.	24	79%	88%	88%	79%	92%	92%	79%	100%	100%
taut	43	65%	81%	81%	70%	84%	84%	74%	84%	84%

Table: Experimental Evaluation

- `https://github.com/FredMesnard/lptp`
- `https://github.com/atp-lptp/automated-theorem-proving-for-prolog-verification`

Conclusion

We have presented a compiler

- from LPTP: FOL for Prolog verification
- to FOF: the assembly language
- executable: on any FOF processor
- + an experiment with E and Vampire

Why does it work?

- Concepts and tools closely related to FOL:
  - pure Prolog + sound negation
  - LPTP specification language + IND(P)
  - FOF, the *Esperanto* of FOL syntax for ATP
  - Availability of efficient FOL theorem provers
- Huge computing power of our modern laptops
- *Smart* slicing of the LPTP library proofs by Stärk

What's next? A *hammer* for LPTP?

Conclusion

What's next? A *hammer* for LPTP?

- From an ATP resolution proof
  - build the corresponding LPTP natural deduction proof
  - check it with the LPTP proof checker