Automated Theorem Proving for Prolog Verification

Fred Mesnard Thierry Marianne Étienne Payet

LIM, université de La Réunion

May 2024

Prolog verification

- Termination – which tool?
  E.g., ours for Logic Programming (LP)
  NTI+cTI ranked 1st at TermComp 2022, 2023
- Partial correctness – which tool?
  - A few theoretical frameworks
  - LPTP: Logic Program Theorem Prover
    Robert Stärk, mid 90's

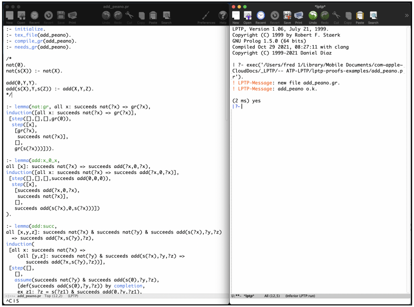
Main LPTP paper

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

LPTP: an interactive theorem prover (ITP)

- An Emacs user-interface
- A proof checker written in ISO-Prolog
- A proof manager based on TEX and HTML

Runs out of the box 30 years later!



The languages of LPTP – the object language

Pure ISO-Prolog with negation and the occurs check

- 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
- Operational semantics: ISO-Prolog with the occurs check

The languages of LPTP – the specification language

FOL

- $\hat{\mathcal{L}}$ 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 languages of LPTP – 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:

- $\mathbf{S}G$ means G *succeeds*
  Any breadth-first evaluation of G succeeds
- $\mathbf{F}G$ means G *fails*
  The ISO-Prolog evaluation stops without any answer
- $\mathbf{T}G$ means G *terminates*
  The ISO-Prolog evaluation produces a finite number of answers then stops

IND(P) – from the JLP paper

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

IND(P), comprises the following axioms:

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

1. $f(x_1,\dots,x_m) = f(y_1,\dots,y_m) \rightarrow x_i = y_i$ [if f is m-ary and $1 \leqslant i \leqslant m$]
2. $f(x_1,\dots,x_m) \neq g(y_1,\dots,y_n)$ [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. $\mathrm{gr}(x_1) \wedge \dots \wedge \mathrm{gr}(x_m) \leftrightarrow \mathrm{gr}(f(x_1,\dots,x_m))$ [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

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

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

be the formula obtained from

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

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

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

be the formula

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

Then the simultaneous induction axiom is the following formula:

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

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*] $\forall x, y\,(\mathbf{S}\,\mathtt{nat}(x) \to \exists z\ \mathbf{S}\,\mathtt{add}(x, y, z))$.

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

Lemma [*add:x_0_x*] $\forall x\,(\mathbf{S}\,\mathtt{nat}(x) \to \mathbf{S}\,\mathtt{add}(x, 0, x))$.

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

ATP for LPTP?

Observation:

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

Idea:

- Go back to FOL by *expliciting* the axioms IND(P) in FOF (*First Order Form*) and invoke *any* first-order theorem prover

Experimentation:

- Try with E and Vampire

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 $\text{IND}(P')$ compiled as FOF axioms
- Previously processed FOF conjectures are converted as FOF axioms (hence we produce as many FOF files as there are properties in the initial LPTP proof file)
- 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: 77%

lib	#	E-1s	V-1s	EV-1s	E-10s	V-10s	EV-10s	E-60s	V-60s	EV-60s
nat	91	54%	72%	78%	58%	77%	80%	61%	81%	85%
gcd	11	45%	45%	45%	45%	45%	45%	45%	45%	45%
list	84	56%	73%	83%	67%	87%	90%	68%	89%	92%
suff	31	74%	81%	93%	81%	94%	97%	81%	97%	100%
rev	25	52%	64%	64%	64%	80%	84%	68%	84%	88%
perm.	42	45%	50%	55%	52%	59%	64%	52%	64%	67%
sort	42	33%	33%	40%	43%	57%	57%	48%	59%	62%
merg.	24	50%	71%	71%	62%	79%	79%	67%	79%	79%
taut	43	0%	67%	67%	65%	70%	70%	65%	74%	74%

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

Conclusion

A compiler from Prolog/LPTP to FOF as the assembly language and an experiment with E and Vampire as FOF processors

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
- Nice slicing of the LPTP library proofs

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

- Automatic construction of the corresponding LPTP proofs