Interactive Proofs for Logic Programs

Fred Mesnard

LIM, université de La Réunion, France

September 2025

Plan

Intro: which *implemented tool* for Prolog verification?

- Type system for Prolog
  Many papers, a book (Frank Pfenning), which tool today?
  A Hindley-Milner SWI add-on written by Tom Schrijvers *et al.*
  Towards Typed Prolog, ICLP 2008
- Automated program properties by abstract interpretation
  Many papers, which tool today?
  Ciao Prolog & CiaoPP
- Automated termination analysis
  Many papers, which tool today?
  E.g., ours for pure Prolog
  NTI+cTI ranked 1st at TermComp since 2022
- Partial correctness
  A few papers, which tool today?
  LPTP: Logic Program Theorem Prover
  Robert Stärk, mid-1990's

Plan

Robert Stärk:

- Swiss citizen
- Master in mathematics, ETH Zurich
- PhD in logic, Univ. Bern (92):
  The Proof Theory of Logic Programs with Negation
- Post-docs: Munich, Stanford, Pennsylvania
- Senior assistant, Univ. Fribourg (96-99)
- Assistant professor, ETH Zurich (99-05)

Plan

Summary[1]: LPTP – A Logic Program Theorem Prover

- LPTP is an interactive theorem prover for the formal verification of pure Prolog programs
- Designed and implemented (1994/1999) by Robert Stärk
- Programs may contain negation, if-then-else and built-in predicates like is/2, integer/1, call/n+1, arg/3
- Non-logical predicates and control operators like cut (!), assert /1, retract/1, var/1 are forbidden
- Hypothesis: occurs check during unification at runtime SWI-Prolog: `?- set_prolog_flag(occurs_check,true).`

[1] borrowed from Robert Stärk

Summary: LPTP – A Logic Program Theorem Prover

- Provable properties of programs:
  - universal left-termination
  - equivalence of predicates
  - existence of solutions, uniqueness of solutions
  - functional correctness, types, ...
- LPTP's notion of termination includes non-floundering:
  - negative goals are ground when called
  - built-in predicates are instantiated the right way when called
- each user-defined predicate defines its *own* induction scheme, automatically generated by LPTP
- The proof format of LPTP is natural deduction (ND)
- Proofs are written in a text editor and LPTP checks the correctness of the proofs

The distribution of LPTP[2] includes the source code, a user manual (130 pages) and 47 klop[3] including:

- the verification of various sorting algorithms
- the correctness of a tautology checker
- the verification of algorithms for AVL trees
- the correctness of alpha-beta pruning with respect to min-max
- the correctness of a fast union-find based unification algorithm
- the correctness of a deterministic parser for ISO Prolog

The parser with its specification is 635 lines long. The correctness proof of the ISO standard parser is 13 klop (3 weeks). Hence:

- Proof size $\simeq$ 20 x Prolog code size
- $\sim$ 4 klop/week for *the* expert

[2] e.g., `https://github.com/FredMesnard/lptp`

[3] klop = kilo lines of proof

So LPTP is both a research project ...

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

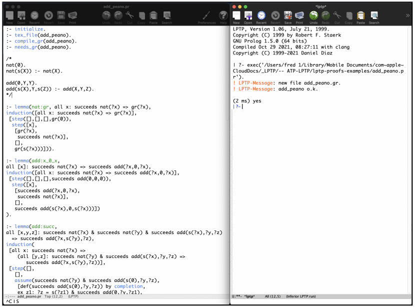
R. F. Stärk
Formal Verification of Logic Programs: Foundations and Implementation
LFCS'97, *LNCS* 1234, 354–368. Springer, 1997

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

... and an interactive theorem prover (ITP)

- An Emacs user-interface
- ND tactic-based ITP
- A *limited* `auto` tactic
- A proof checker written in ISO-Prolog
- A library for usual relations on natural numbers, lists, ...
- A proof manager based on TEX and HTML

Runs out of the box 25 years later:



Plan

A LPTP primer

- Object language
  - Pure Prolog, finite terms, negation as failure
  - Operational semantics: ISO-Prolog with the occurs check[4]
- Specification language
  - Classical first order logic
  - $gr/1$, a constraint defined as $gr(x) \leftrightarrow x$ *is ground*
  - For each user-defined atom G
    - $\mathbf{S}G$ means G *succeeds*
      The breadth-first evaluation of G succeeds
      One or more infinite branches may exist
    - $\mathbf{F}G$ means G *fails*
      The breadth-first evaluation of G fails
      One or more infinite branches may exist
    - $\mathbf{T}G$ means G *terminates*
      The ISO-Prolog evaluation produces a finite number (≥ 0) of answers then stops without floundering
      No infinite branch

[4] SWI-Prolog:

```
?- set_prolog_flag(occurs_check,true).
```

A LPTP primer: an example

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

Lemma [*nat:ground*] $\forall x\, (\mathbf{S}\, \mathtt{nat}(x) \rightarrow gr(x))$.

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

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

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

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

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

A LPTP primer: a 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*] $\forall x, y\,(\mathbf{S}\,\mathsf{nat}(x) \to \exists z\ \mathbf{S}\,\mathsf{add}(x, y, z))$.

Proof.

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

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

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

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

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

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

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

A LPTP primer: let's play!

- the Ciao Prolog Playground for LPTP
  `https://ciao-lang.org/playground/lptp.html`
- Work in progress!
- Best user experience with Google Chrome

Plan

The object language

$\boxed{\text{Pure Prolog, finite terms, negation as failure}}$

- 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
- $\boxed{\text{Operational semantics: ISO-Prolog with the occurs check}}$

SWI-Prolog: `?- set_prolog_flag(occurs_check,true).`

Plan

The specification language

$\boxed{\text{Classical first order logic}}$

- $\hat{\mathcal{L}}$ 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)
- LPTP reasons with the Clark's *if-and-only-if* completed definition of R^s, R^f, R^t for each user-defined predicate R

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*
  The breadth-first evaluation of G succeeds
  One or more infinite branches may exist
- $\mathbf{F}G$ means G *fails*
  The breadth-first evaluation of G fails
  One or more infinite branches may exist[5]
- $\mathbf{T}G$ means G *terminates*
  The ISO-Prolog evaluation produces a finite number of answers then stops without floundering
  No infinite branch

[5] As $\mathbf{F}\backslash{+}G := \mathbf{S}G$, see next slide.

The specification language

Formally:

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

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

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

$$gr(\texttt{true}) := \top \qquad gr((G,H)) := gr(G) \wedge gr(H)$$
$$gr(\texttt{fail}) := \top \qquad gr((G;H)) := gr(G) \wedge gr(H)$$
$$gr(s=t) := gr(s) \wedge gr(t) \qquad gr(\backslash{+}G) := gr(G)$$
$$gr(R(t_1,\ldots,t_n)) := gr(t_1) \wedge \ldots \wedge gr(t_n) \qquad gr(\texttt{some}\ x\ G) := \exists x\ gr(G)$$

IND(P)

Given a logic program P, IND(P) is the following set of nine first order axioms that models the operational semantics of P.

The axioms of Clark's equality theory

1. $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \to x_i = y_i$ [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 $t \not\equiv x$]

The first two axioms specify the usual properties of the trees built from the function symbols extracted from P.
The third axiom forbids infinite trees. It is an axiom schema, i.e., an infinite set of first order axioms.

The predefined constraint $gr/1$

The specification language of LPTP includes a predefined constraint $gr/1$, similar to `ground/1`.

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]

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

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]

Let $D_R^P(\vec{x})$ denote the definition of the completion of the user-defined procedure $R(\vec{x})$ in the logic program P. We know 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 $R^s(\vec{x})$.

Fixed point axioms for user-defined predicates R

8. [for any user-defined predicate R]
   $R^s(\vec{x}) \leftrightarrow \mathbf{S}D_R^P(\vec{x})$
   $R^f(\vec{x}) \leftrightarrow \mathbf{F}D_R^P(\vec{x})$
   $R^t(\vec{x}) \leftrightarrow \mathbf{T}D_R^P(\vec{x})$

Finally, for any property of the form $\forall\vec{x}[R^s(\vec{x}) \to \phi(\vec{x})]$, where $R(\vec{x})$ is a user-defined procedure and $\phi(\vec{x})$ an $\hat{\mathcal{L}}$ -formula, we have a specific induction schema. We examine the simple case of *directly recursive user-defined predicate*.

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 $\hat{\mathcal{L}}$-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 $\forall\vec{x}(R^s(\vec{x}) \to \phi(\vec{x}))$. Let $closed(\phi(\vec{x})/R)$ be the formula obtained from $\forall\vec{x}(\mathbf{S}D_R^P(\vec{x}) \to R^s(\vec{x}))$ by replacing

- $R^s(\vec{x})$ by $\phi(\vec{x})$ on the right of $\to$,
- all occurrences of $R(\vec{t})$ appearing on the left of $\to$ by $\phi(\vec{t}) \wedge R(\vec{t})$.

Then the induction axiom is the following formula:

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

Main theoretical results from the JLP paper

Adequacy of IND(.)

The inductive extension IND(.) is always consistent, and is a sound and complete axiomatization of the operational semantics of pure Prolog.

Let Q be a query $(G_1, \ldots, G_{n+1})$:

- If $\mathrm{IND}(P) \vdash \mathbf{T}Q$ then Q terminates
- If Q terminates then $\mathrm{IND}(P) \vdash \mathbf{T}Q$
- If $\mathrm{IND}(P) \vdash \mathbf{T}Q \wedge \mathbf{S}Q\sigma$ then Q terminates and one of its answers includes σ
- If Q breadth-first succeeds with answer σ then $\mathrm{IND}(P) \vdash \mathbf{S}Q\sigma$
- If $\mathrm{IND}(P) \vdash \mathbf{T}Q \wedge \mathbf{F}Q$ then Q finitely fails
- If Q finitely fails then $\mathrm{IND}(P) \vdash \mathbf{T}Q \wedge \mathbf{F}Q$

NB: Termination is *ubiquitous*

Plan

A LPTP-derivation is a finite list of derivation steps:

```
derivation_step → formula
        | formula by tag
        | assume(formula,derivation,formula)
        | cases(formula,derivation,formula,derivation,formula)
        | cases([case(formula,derivation),...],formula)
        | exist(name,formula,derivation,formula)
        | exist([name,...],formula,derivation,formula)
        | induction([formula,...],
           [step([name,...],[formula,...],derivation,formula),...])
        | contra(formula,derivation)
        | indirect(~formula,derivation)
```

The LPTP-proof format is based on natural deduction (1)

The basic rules of natural deduction:

	introduction	*elimination*
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}\wedge\mathrm{i}$	$\dfrac{\phi \wedge \psi}{\phi}\wedge\mathrm{e}_1 \qquad \dfrac{\phi \wedge \psi}{\psi}\wedge\mathrm{e}_2$
$\vee$	$\dfrac{\phi}{\phi \vee \psi}\vee\mathrm{i}_1 \qquad \dfrac{\psi}{\phi \vee \psi}\vee\mathrm{i}_2$	$\dfrac{\phi \vee \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\vee\mathrm{e}$
$\rightarrow$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \rightarrow \psi}\rightarrow\mathrm{i}$	$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\rightarrow\mathrm{e}$
$\neg$	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\neg\mathrm{i}$	$\dfrac{\phi \quad \neg\phi}{\bot}\neg\mathrm{e}$

```
:- lemma(dnprop:conj:intro, p & q => (p & q), []).
```

```
:- lemma(dnprop:conj:elim:1, (p & q) => p, []).
```

```
:- lemma(dnprop:disj:intro:1, p => (p \/ q), []).
```

```
:- lemma(dnprop:disj:elim, ((p \/ q) & (p => r) & (q => r)) =>
assume((p \/ q) & (p => r) & (q => r),
 [p \/ q, cases(p,r,q,r,r)], r)).
```

```
:- lemma(dnprop:impl:intro,
p => p,
assume(p, [p], p)).
```

```
:- lemma(dnprop:impl:elim, (p & (p => q)) => q, []).
```

```
:- lemma(dnprop:nega:intro,
(p => ff) => ~ p,
assume(p => ff,contra(p,ff),~ p)).
```

```
:- lemma(dnprop:nega:elim, (p & ~ p) => ff, []).
```

The LPTP-proof format is based on natural deduction (2)

$\neg\neg$

Some useful derived rules:

$$\frac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi}\ \mathrm{MT}$$

```
:- lemma(dnprop:modus_tollens,
((p => q) &  ~ q) => ~ p,
assume((p => q) & ~ q,
 contra(p,[p => q, q, ~q, ff]),~ p)).
```

$$\frac{\boxed{\begin{array}{c}\neg\phi\\ \vdots\\ \bot\end{array}}}{\phi}\ \mathrm{PBC}$$

```
:- lemma(dnprop:proof_by_contradiction, (~ p => ff) => p,
assume(~ p => ff,indirect(~p,ff),p)).
```

$$\frac{\neg\neg\phi}{\phi}\ \neg\neg\mathrm{e}$$

```
:- lemma(dnprop:neganega:elim, (~ ~ p) => p,
assume(~ ~ p,indirect(~ p, ff),p)).
```

$$\frac{\phi}{\neg\neg\phi}\ \neg\neg\mathrm{i}$$

```
:- lemma(dnprop:neganega:intro,p => (~ ~ p),
assume(p,contra(~ p,ff),~ ~ p)).
```

$$\frac{}{\phi \vee \neg\phi}\ \mathrm{LEM}$$

```
:- lemma(dnprop:law_of_excluded_middle, p \/ ~p, []]
```

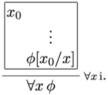
Adapted from:

Logic in computer science - modelling and reasoning about systems

Huth & Ryan – Cambridge University Press 2000

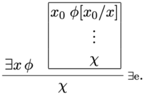
The LPTP-proof format is based on natural deduction (3)

```
:- lemma(dnpred:forall:elim,(all x:p(?x)) => p(a),[]).
```




```
:- lemma(dnpred:forall:intro,(all x:p(?x)) => (all y:p(?y)),
assume(all x:p(?x),[p(?y)],all y: p(?y))).
```

```
:- lemma(dnpred:exist:intro,p(a) => (ex x:p(?x)),[]).
```




```
:- lemma(dnpred:exist:elim,(ex x:p(?x)) => (ex z:p(?z)),
assume(ex x:p(?x), exist(x0,p(?x0),[],ex z:p(?z)),
 ex z:p(?z))).
```

Plan

ATP for Prolog Verification

ICLP'25

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* in the IDE

Idea:

- Go back to FOL by translating IND(P) in TPTP FOF (*First Order Form*) and invoke *any* FOF-compatible ATP
- *i.e.*,
  LPTP for Prolog verification
  & ATP for automating LPTP
  $\Rightarrow$ ATP for Prolog Verification

Experimentation:

- Try with E and Vampire on the LPTP lib

Plan

Automated Certification of LP Groundness Analysis

LOPSTR'25

Observation:

- Abstract interpreters automatically generates invariants
  E.g., $\forall x(\mathbf{S}\ nat(x) \Rightarrow gr(x))$
- But abstract interpreters are complex pieces of software
- Bugs?

Idea:

- Certify the invariants *a fortiori* using LPTP instead of trying to prove correctness of the abstract interpreter

Experimentation:

- Apply this to LP groundness analysis
  Compare:
  - the *ATP for Prolog Verification* approach
  - the automatic construction of propositional LPTP proofs

Plan

Project ideas:

- Exercises:
  - Read the first chapter of the LPTP manual
  - Prove in LPTP some of the P-99 Prolog Problems
- Intermediate problems:
  - Prove in LPTP that $\sqrt{2}$ is irrational
  - Prove in LPTP that the set of prime numbers is infinite
  - Implement a LaTeX output module for LPTP
  - Implement a Markdown output module for LPTP
  - Compile propositional resolution proofs to LPTP
  - Instrument the LPTP source code with Ciao-PP declarations
- Advanced problems:
  - Prove in LPTP the following case studies and compare the Pedreschi & Ruggieri's framework with the LPTP approach
  - Experiment logic-based abstract interpretation with LPTP
  - QuickCheck and counter model generation for LPTP
  - Rewrite FOL Vampire/E proofs into LPTP proofs
  - Prove LPTP in LPTP
  - Add SMT-solvers to LPTP - Generalize the JLP paper in Rocq
  - Adapt Curry-Howard to LPTP

Plan

Summary:

- LPTP is a FOL ITP for pure Prolog
- LPTP has an Emacs-based IDE with TeX/HTML output
- LPTP now runs directly in any modern web browser:
  - the Ciao Prolog Playground for LPTP
- LPTP provides a *unified framework* for natural deduction proofs applied to propositional logic, FOL, and pure Prolog
- Blending LPTP with modern technologies opens research opportunities

Please share your comments, bug reports and ideas about these slides: `frederic.mesnard@univ-reunion.fr`

Thank you!

Example 1

Let P_1 be:

```
p :- p.
```

IND(P_1) contains:

$$\mathbf{S}p \leftrightarrow \mathbf{S}p \tag{1}$$

$$\mathbf{F}p \leftrightarrow \mathbf{F}p \tag{2}$$

$$\mathbf{T}p \leftrightarrow \mathbf{T}p \tag{3}$$

$$\neg(\mathbf{S}p \wedge \mathbf{F}p) \tag{4}$$

$$\mathbf{T}p \rightarrow (\mathbf{S}p \vee \mathbf{F}p) \tag{5}$$

There are five models of IND(P_1):

```
?- sat(~(S*F)*(T =< (S+F))),labeling([S,F,T]).
S = F, F = T, T = 0 ;                    % 0 0 0
S = T, T = 0, F = 1 ;                    % 0 1 0
S = 0, F = T, T = 1 ;                    % 0 1 1
S = 1, F = T, T = 0 ;                    % 1 0 0
S = T, T = 1, F = 0.                     % 1 0 1
```

Example 2

Let P_2 be:

```
p :- \+ p.
```

IND(P_2) contains:

$$\mathbf{S}p \leftrightarrow \mathbf{F}p \tag{6}$$
$$\mathbf{T}p \leftrightarrow \mathbf{T}p \tag{7}$$
$$\neg(\mathbf{S}p \wedge \mathbf{F}p) \tag{8}$$
$$\mathbf{T}p \rightarrow (\mathbf{S}p \vee \mathbf{F}p) \tag{9}$$

There is only one model of IND(P_2):

```
?- sat((S =:= F)* ~(S*F)*(T =< (S+F))),labeling([S,F,T]).
S = F, F = T, T = 0.                                    % 0 0 0
?-
```

Example 2

Here are the proofs:

```
:- lemma(not_s_p, ~ succeeds p,
contra(succeeds p, [fails p, ff])).

:- lemma(not_f_p, ~ fails p,
contra(fails p, [succeeds p, ff])).

:- lemma(not_t_p, ~ terminates p,
contra(terminates p,
   [succeeds p \/ fails p,
    cases(succeeds p,
            [~ succeeds p by lemma(not_s_p), ff],
          fails p,
            [~ fails p by lemma(not_f_p),ff],
          ff),
    ff])).
```

Example 3

Let P_3 be:

```
p :- q.      p :- \+ q.      q :- q.
```

$\mathrm{IND}(P_3)$ contains:

$$\mathbf{S}p \leftrightarrow \mathbf{S}q \vee \mathbf{F}q \tag{10}$$

$$\mathbf{F}p \leftrightarrow \mathbf{F}q \wedge \mathbf{S}q \tag{11}$$

$$\mathbf{T}p \leftrightarrow \mathbf{T}q \wedge \mathbf{T}q \tag{12}$$

$$\neg(\mathbf{S}p \wedge \mathbf{F}p), \neg(\mathbf{S}q \wedge \mathbf{F}q) \tag{13}$$

$$\mathbf{T}p \rightarrow (\mathbf{S}p \vee \mathbf{F}p), \mathbf{T}q \rightarrow (\mathbf{S}q \vee \mathbf{F}q) \tag{14}$$

Example 3

There are five models for IND(P_3):

```
?- sat((Sp =:= Sq+Fq)*(Fp =:= Fq*Sq)*(Tp =:= Tq)*
 ~(Sp*Fp)*  ~(Sq*Fq)*(Tp =< (Sp+Fp))*(Tq =< (Sq+Fq))),L=
[Sp,Fp,Tp,Sq,Fq,Tq],labeling(L),writeln(L),fail.
[0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 1, 0]
[1, 0, 1, 0, 1, 1]
[1, 0, 0, 1, 0, 0]
[1, 0, 1, 1, 0, 1]
false.
?-
```

Note that IND(P_3) models $\neg \mathbf{F}p$ and $\mathbf{T}q \to \mathbf{S}p$.

Example 3

Here are the proofs:

```
:- lemma(n_f_p, ~ fails p,
contra(fails p,[def fails p by completion,
                fails q & succeeds q, ff])).

:- lemma(t_q_imp_s_p, terminates q => succeeds p,
assume(terminates q,
 [succeeds q \/ fails q,
  cases(succeeds q, succeeds p by sld,
        fails q, succeeds p by sld,
        succeeds p)],
succeeds p)).

:- lemma(t_q_imp_s_p:alt, terminates q => succeeds p, []).
```