LPTP - A Logic Program Theorem Prover

LPTP is an **interactive** theorem prover for the formal verification of **pure Prolog programs**. Programs may contain negation, if-then-else and built-in predicates like is/2, integer/1, call/n+1, arg/3, etc. Non-logical predicates and control operators like cut (!), assert /1, retract/1, var/1 are forbidden, since they destroy either the lifting property or modify the program during run-time. It is assumed that the occurs check is done during unification.

The **specification language** of LPTP is first-order predicate logic. The types and basic relations in specifications are Prolog predicates. (Prolog is both a programming language and a specification language.)

Properties of programs which can be proved in LPTP are:

- left-termination (universal termination),
- equivalence of predicates,
- existence of solutions,
- uniqueness of solutions,
- equivalence of programs and specifications.

LPTP's notion of termination includes non-floundering, i.e. negative goals are ground and built-in predicates are instantiated the right way, when they are called.

The underlying **logic** of LPTP is the **inductive extension** of logic programs (IND). Unlike the Clark completion of logic programs the first-order theory IND contains induction principles and is always consistent. The code of **a Prolog predicate is translated into three positive inductive definitions** of relations expressing **success, finite failure and universal termination** of the predicate. IND has the same proof-theoretic strength as Peano Arithmetic.

IND is hard-wired into LPTP. Given the code of a Prolog predicate, LPTP generates the **induction scheme** automatically. The number of induction steps in an induction scheme depends on the number of clauses and nested if-then-elses in the definition of the predicate. Even for small programs (100 lines) the number of induction steps can be big (40 steps).

In order to apply the theorems proved in IND to Prolog's depth-first, left-to-right query evalutaion procedure, it is necessary to prove termination of the predicates. Termination implies that the Prolog evaluation does not depend on the order of the clauses in the program.
This is not a restriction, since for most programs the order of clauses is not important. (Some people consider it as good Prolog style to write programs in such a way that the evaluation does not depend on the order of the clauses listed in the file.)

The proof format of LPTP is a version of **natural deduction**. Proofs are written in a text editor and submitted to a LPTP process in the background. LPTP checks the correctness of the proof and tries to close the **gaps** in the proof according to one of 10 **tactics** the user can choose from. It is not necessary to remember the names of previously proved theorems and lemmas. LPTP knows them. LPTP is able to find a matching lemma in the internal database automatically.

LPTP is implemented in Prolog and runs under **SICStus Prolog, GNU Prolog, SWI-Prolog. Quintus Prolog, ECLiPSe and C-Prolog**. The Emacs editor is used as a graphical user interface (GNU Emacs or XEmacs). The Emacs mode for LPTP includes automatic indentation of proofs and syntax coloring. Other features are: double clicking on a quantifier highlights the scope of the quantifier, etc. LPTP creates **TeX and HTML-output**.

The distribution of LPTP includes the source code, a **user manual** (130 pages) and 47000 lines of **example proofs** such as:

- 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 deterministic **parser for ISO standard Prolog** with respect to a DCG.
- the correctness of a fast **union-find based unification algorithm**.

The fully formalized correctness proof of the ISO standard parser is

13000 lines long. The parser together with its specification has 635 lines. Its correctness proof has been created in three weeks. The proof contains theorems like the following:

- A token list is a correct expression according to the ISO Prolog grammar if and only if the token list is accepted by the parser.
- If the prefix, infix and postfix operators satisfy the conditions stated in the ISO Prolog standard, then a token list can be parsed into a parse tree in at most one way.
- if a parse tree T1 is transformed into a token list using write and the token list is parsed back into a tree T2 using read, then T1= T2.

LPTP is able to find short derivations automatically. Larger proofs have to be created interactively.

References

1. R. F. Stärk, **The theoretical foundations of LPTP (a logic program theorem prover)**. Journal of Logic Programming, 36(3):241-269, 1998.
   [Theoretical foundations of LPTP. Soundness and completeness.]
2. R. F. Stärk, **Formal verification of logic programs: foundations and implementation**. In S. Adian and A. Nerode, editors, Logical Foundations of Computer Science LFCS '97 - Logic at Yaroslavl, Russia. Springer-Verlag, Lecture Notes in Computer Science 1234, pages 354--368, 1997.
   [A short survey on LPTP.]
3. R. F. Stärk, **The finite stages of inductive definitions**. In P. Hájek, editor, GÖDEL'96, Logical Foundations of Mathematics, Computer Science and Physics - Kurt Gödel's Legacy, Bro, Czech Republic.
   Springer-Verlag, Lecture Notes in Logic 6, pages 267-290, 1996.
   [The proof-theoretic strength of LPTP.]
4. R. F. Stärk, **Total correctness of pure Prolog programs: A formal approach**. In R. Dyckhoff, et. al. editors, Proceedings of the 5th International Workshop on Extensions of Logic

Programming, ELP
'96, Leipzig, Germany, Springer-Verlag, Lecture Notes in
Artificial Intelligence 1050, pages 237--254, 1996.
[A case study in LPTP: a tautology checker.]