A new Proof-Manager and Graphic Interface for the Larch Prover

Frédéric Voisin

C.N.R.S. U.R.A. 410 and Université de Paris-Sud,
L.R.I., Bât. 490, F-91405 Orsay Cedex, France

Abstract. We present PLP, a proof management system and graphic interface for the "Larch Prover" (LP). The system provides additional support for interactive use of LP, by letting the user control the order in which goals are proved. We offer improved ways to investigate, compare and communicate proofs by allowing independent attempts at proving a goal, a better access to the information associated with goals and an additional script mechanism. All the features are accessible through a graphic system that makes the proof structure accessible to the user.

1 Introduction

The "proof-debugger" LP is part of the Larch project. It has been designed to help reasoning about algebraic specifications written in the Larch specification language by making it easier to prove properties of such specifications [2]. LP has also been applied to other domains such as the proof of circuits, of software components or of distributed algorithms [1, 3]. Here we focus on proof management since our system does not add any new logical mechanism to the ones already present in LP. We shall only recall that LP supports multi-sorted first-order formulas and offers various proof mechanisms, usually applied on user's request. The main operational mechanism is term rewriting with additional commands on top of it. Proof commands in LP are split in two groups: the "forward-inference" commands, used to enrich the current logical system without modifying the goal to be proved (like in critical-pairing or quantifier elimination), and the "backward-inference" commands, used to decompose the proof of a goal into the proofs of several subgoals (as in proof by cases or by induction), usually with some hypotheses. Therefore each subgoal is proved in a independent logical system formed by the initial axiomatization and the hypotheses corresponding to the various proof commands at the origin of a particular subgoal. The original formulas and rewrite systems can be altered as part of the proof process: orientation of equations into rewrite rules, inter-normalization of rewrite systems.

2 What's new with plp

The preliminary objectives and design of our system are described in [4]. Our system enriches LP with additional support for the interactive work on proofs

and provides better mechanisms to investigate and compare proofs. LP is guided by the "design, code, debug" approach and offers very efficient commands for running large proofs written as scripts, but we also need more interactive support for helping in completing unfinished proofs or in correcting failed ones. Part of the problem is that it is not easy to write a script from scratch and to guess the exact form of the subgoals to prove, or the associated rewrite systems, after a few proof commands. Moreover, for a given subgoal, one can sometimes think of several ways to prove it and we want to be able to compare them, their subgoals or the contexts in which they are proved, without having to discard one strategy for trying another one. Also, with LP, a subgoal is discarded once proved, and its logical system is no longer accessible to the user. When the user is blocked in the proof of some subgoal, there is no possibility of switching to another subgoal, for instance for gaining some experience on another subgoal, or to understand why the proof of some subgoal succeeds while the proof of another do not. This hinders the comparison of similar proofs and this is where our system can help !

User control on the order of proof steps: LP does not provide the user with the control over the order in which the subgoals are proved: Each subgoal must be proved as soon as it is introduced, and the relative order of the subgoals originating from a given command is imposed by the system. We use the same default ordering, but at any moment the user of PLP can switch to another subgoal without first completing the current goal. New conjecture can be introduced by the user, that rely on conjectures that have not yet been completed. Therefore a user can prove the subgoals in the order that is the most natural for he/her, skip parts of a long proof when wanting to focus on a subpart of it, or state a sequence of conjectures whose proofs are deferred to separate files. The system automatically records which goals are unproved and proposes a new goal when the current task is completed.

Multiple attempts at proving subgoals: We allow independent attempts at proving subgoals, using different proof strategies. Variants can be started, cancelled, left uncompleted and later resumed, and the user can switch back and forth among them. A variant at a node is logically compatible with any variant at a node in an independent subtree: The validity depends only on the formulas in the subgoals. All subgoals have a "current" variant, with respect to which commands are interpreted. Switching between variants is done only on user request to minimize the risk of confusion for the user.

Variants are also useful to "replay" part of a proof either to try to simplify it or to have a closer look at its execution. This may be more convenient than retrieving the corresponding part in the log file produced by LP.

Better access to proof information: With PLP, proved subgoals are not discarded automatically and it is possible to re-enter them to inspect their logical systems or to perform some computation (like normalization). This gives an easier way to compare the proofs of independent subgoals. Part of the information is recorded within the interface part and is accessible by mouse clicking without interaction with the proof engine (that can be working on a different subgoal).

This includes the basic information about subgoals: logical status, the formula as initially stated and its current form after processing by the proof engine, current hypothesis etc. The rest of the information for a subgoal, even proved, can be retrieved by selecting that subgoal as the current focus for LP. This gives access to additional information, like the associated rewrite system, that would be too large to record within the interface part. Being able to run a whole proof and later browse through it, while picking up local information easily, provides valuable help when trying to understand someone else's proof.

Graphical presentation of proofs: We provide an explicit view of the tree that is the natural representation of a proof, with a proof command connecting a goal to the list of its subgoals. The selection of goals is done by mouse clicking or by name. The tree structure is used for representing backward-inference commands, the only ones that introduce subgoals. Forward-inference commands, which are not undoable in LP and which do not introduce subgoals, are displayed with a square box whose opening lists all the forward-inference commands issued for the subgoal. Different displays for completed and uncompleted subgoals make clear where unfinished parts are. Pointing at a node provides information about it (logical status, associated hypothesis, etc) while selecting it as the current focus for LP allows to (re-)enter the associated logical context and make it ready to accept new commands.

Variants can also be displayed in separate windows. This helps the comparison between different attempts at proving a goal. No proof action can be issued from the windows associated with variants, to prevent confusion about the node at which a proof action will take place. A variant must be selected as the current variant at a goal before one can issue a command for it. A "stack" display of commands for subgoals with variants makes explicit the presence of variants.

The tree structures can be dumped in Postscript format for later printing or inclusion into documents, in a form more readable than textual scripts.

New script mechanism: An additional script mechanism complements the one that exists in LP which provides an on-line recording of all user's actions, even the ones that have no impact on the proof (displays, cancelled actions, errors, etc). The new mechanism traverses the proof tree structure and lists only the commands that are necessary to rebuild the tree structure (or a selected part of it), cleared of all superfluous commands.

3 Conclusion

The new prototype system runs on SUN workstations. It is based on a customized release of LP, built in collaboration with Steve Garland from MIT. The proof engine is in CLU, the proof-manager part is in C and uses Tcl/Tk for the graphic manipulation. This prototype can be viewed as a first step towards a "proof editor" that would take advantage of the explicit proof structure to provide additional facilities. Among them we can mention dynamic annotation of scripts, scratch-pad facilities for performing computations at subgoals, a "replay" mechanism for reusing a proof at some subgoal for another subgoal, or the

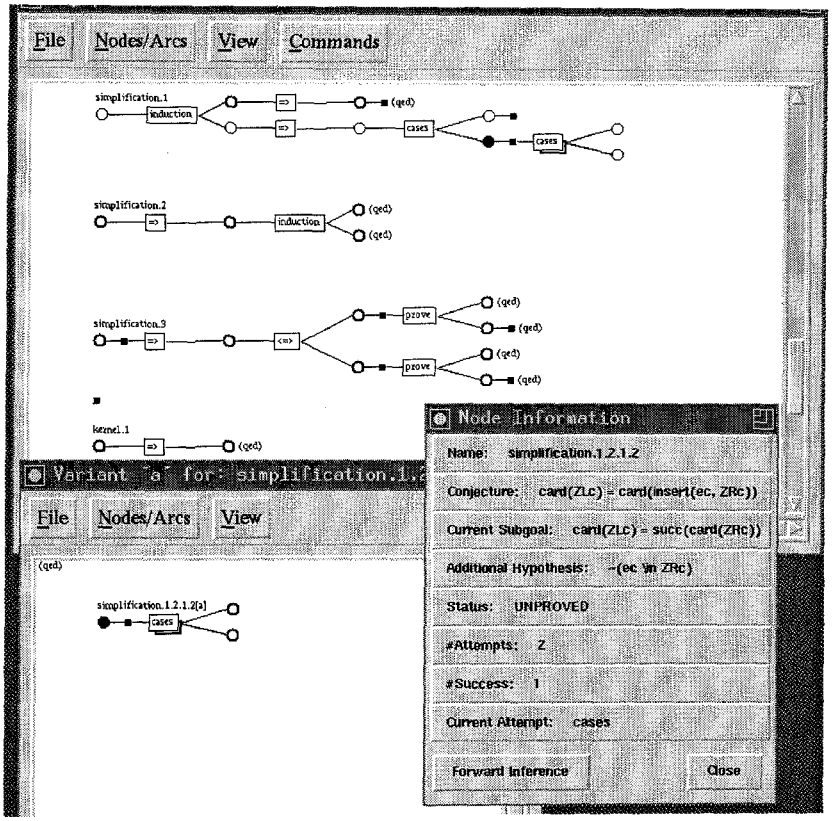

Fig. 1. A snapshot of the system

dynamic reshaping of proofs like when moving lemmas higher in a proof tree to make them sharable by several subgoals.