A Reduced Instruction Set Proof Environment

Holger Busch

SIEMENS AG, Corporate Research,81730 München,Germany
busch@zfe.siemens.de

Abstract. A general-purpose proof interface has been created on top of the higher-order-logic theorem prover LAMBDA in order to improve the efficiency of human interaction and minimize the learning overhead. Users are freed from tedious low-level interactions by way of extended proof automation routines. All essential LAMBDA functions for interactive proof development are accessible via a handy set of user commands.

1 Introduction

Higher-order-logic theorem proving has many advantages and potential applications in circuit design and other areas. Its integration in industrial design scenarios, however, has been restricted by a high learning overhead, an insufficient automation, efficiency problems, and the insuitability of proof scripts for the documentation and communication of verification results. Design engineers and even many formal experts therefore consider a higher-order-logic theorem prover to be inadequate for reasoning about commercial designs.

Driven by this discussion and our own experience, we created a prototypical general-purpose proof environment (Rispe - *Reduced instruction set proof environment*) which gives non-experts on the proof system LAMBDA [4] with minimal learning effort access to all important proof functions through a small number of intuitive proof commands and enables effective proof interaction without scanning large libraries and writing sophisticated tactics.

LAMBDA is supposed as a formal tool for hierarchical top-down system design. It includes the graphical interface DIALOG, which provides a schematics editor, various synthesis and hardware-specific proof functions. The kernel LAMBDA system is an interactive general-purpose theorem prover for classical higher-order logic. Its meta and implementation language is standard ML [5], to which the object language of logic terms has been adapted. LAMBDA comprises extended libraries of rules, tactics, and utilities for adding new proof procedures. We use the kernel prover without DIALOG.

2 General Approach

General measures are summarized to remedy problems encountered in many years' experience with the system LAMBDA. The problems match independent

reports from other higher-order-logic systems and reflects the criticism of advocates of automatic first-order provers.

Accessibility. The amount and variety of proof functions are an obstacle not only for novices. In Rispe, related proof activities which in LAMBDA require a variety of distinct functions are combined. This overloading is internally resolved by analysing the current state of the proof goal along with a simple user-specified parameter. Thus the appropriate internal proof function is invoked and the relevant rules and other parameters are extracted from invisible data bases.

Proof Granularity. Unless users program tactics, trivial preparatory manipulations are often required before available proof tools apply in LAMBDA. Rispe provides functions which automate these auxiliary transformations. Other proof routines in Rispe automate complex proof tasks which are even beyond the capabilities of automatic first-order procedures. Thus the required user interaction is reduced to the essential proof steps.

Goal Handling. Visual analysis of large proof goals is tedious but indispensable for assessing proof tasks and taking decisions. As a side effect of the improved automation, there are many fewer intermediate goals. Most of previously required syntactical analyses of proof goals are included in the internal proof routines. A convenient facility for syntactical abstraction, expansion and reabbreviation of subterms further improves the control of the complexity of goals.

Proof Documentation. Typically, ML-records of interactive LAMBDA sessions hardly serve as understandable documentation. In Rispe the significant reduction of proof steps and variety of proof comands already leads to better readable proof scripts. Owing to the uniform format of Rispe commands, Rispe scripts are suitable for automatic documentation generation.

3 Overview of Rispe Commands

A selection of the main Rispe commands are displayed in Fig.1. All Rispe commands take one string as parameter for specifying subterms to which a proof function is to be applied or supplying other information as guidance. The main groups of Rispe functions are characterized in the following.

3.1 Interactive Proof Functions

These commands are intended for transforming a goal in a stepwise manner. They allow the user direct influence on the creation of a subsequent goal state. The distinction to the automatic proof functions discussed later is fluent, though, for significant automation has been added internally.

Rule Application (**APPLY**). Rule schemes are compact and legible portions of

APPLY	: intelligent rule application
INST	: instantiation
GEN	: generalization
IND	: structural and well-founded induction
CASAN	: case analysis
ESIMP	: equational simplification + expansion
FOSC	: first-order calculus
ARITH	: arithmetic conversions
DEF	: adding logic definitions
AX	: definition of (temporary) axioms
PROVE	: start of a proof
POPRL	: store current top rule

Fig. 1. The most important Rispe commands

proof knowledge. Awkward pre-transformations for applying those in LAMBDA are automated by way of unifiability analyses in Rispe.

Instantiation (**INST**). A variety of instantiation functions for free or bound variables exist in LAMBDA. Even obvious instances of quantified variables have to be specified explicitly, often preceded by necessary hypothesis permutations. In Rispe, heuristics permute universally quantified hypotheses and guess instantiation terms fully automatically[3] invoked by one command.

Generalization (**GEN**). This feature allows conveniently abstracting unessential details and creating generic rule schemes [1]. In LAMBDA, basic term generalization utilities are available for programming purposes. In Rispe, a convenient command is offered which includes simultaneous generalization.

Induction (**IND**). The induction command of Rispe supports multiple structural and well-founded induction. Explicit induction rule schemes are constructed for well-founded induction [2]. Auxiliary information is generated for handling tupled induction variables. Partially specified induction schemes may be specialized at a later proof stage. Measure functions are computed if not supplied by the user. Tests for completeness of cases are included; conditions for excluding incomplete cases through all recursions are computed. The induction heuristics find most termination proofs for recursive function definitions automatically.

Case Analysis (**CASAN**). The case analysis command of Rispe combines and extends many different functions of LAMBDA. Its parameter specifies the term(s) to be analysed, which may be a free or bound variable, a term, two expressions to be compared according to a partial order relation, or just the key-word `if` for analysing test expressions. The current goal is used to internally select the appropriate case analysis function. Explicit assignments for analysis expressions are accepted. Incomplete cases are automatically supplemented. Automatic proof routines attempt to discharge the subcases.

3.2 Automatic Proof Functions

These functions allow the user to start various automatic proof mechanisms, which discharge or at least simplify a given goal.

Equational Simplification (**ESIMP**). Conversions in LAMBDA are strategies[1] for recursively replacing subterms by way of conditional equations. The simplifying effect of conditional equations highly depends on the order of subterms. In Rispe various auxiliary transformations including permutations and a subset of a first-order sequent calculus are applied. They yield favourable syntactical representations which greatly increase the effect of equational simplification.

Expansion of definitions (**ESIMP**). In LAMBDA different expansion mechanisms are used. Some of those have to be parameterized with explicit names of equational expansion rules. As frequently an expansion step is followed by equational simplification, it is included in the same Rispe command. It uniformly allows expansions by just specifying the identifier of the abbreviation or function as occurring in the proof goal. Wildcards are supported. It is possible to restrict the expansion to subterms. The expansion function may be applied reversely.

First-Order Automation (**FOSC**). First-order sequent calculus or related procedures are a valuable enhancement of a higher-order logic theorem prover. An according proof function is part of Rispe [3]. It even handles a restricted class of higher-order goals. Combined tactics for instantiating universal quantification, equational simplification, restricted case analysis, and others yield a higher degree of automation than achievable through pure first-order sequent calculus. Look-ahead-functions which avoid unnecessary calls of subtactics and fast conversions yield a good performance. If a goal is not discharged completely, often useful simplifications are obtained. The underlying subtactics are reconfigurable, reusable, and tunable to specific application areas.

Arithmetic Simplifications (**ARITH**). In Rispe, efficient procedures and conversions for arithmetics are combined with subtactics of other automatic Rispe functions including the instantiation of quantifications.

3.3 Proof Management.

Various functions to support the management of proofs starting from the insertion of logic definitions have been provided in Rispe.

Logic Definitions (**DEF**). The functions of LAMBDA for reading logic definitions generate a parser environment and a set of rules about the defined entities. The definition function of Rispe generates auxiliary information which is invisibly referenced by other Rispe functions. This information hiding relieves the user

[1] The standard ones are depth-first or breadth-first. Conversionals allow advanced users to program other strategies.

from managing a large amount of proof objects and from much tedious initialisation work before proofs about new definitions are started. For instance, the case analysis function has access to a dynamic list of case analysis rules, which are generated from datatype definitions. An automatic facility for proving the termination of recursive functions can be invoked at definition time or later.

Axioms (**AX**). A useful facility of LAMBDA keeps track of user-defined axioms. This facility has been extended in Rispe for supporting a top-down proof strategy. Dependencies of proofs on axioms can be discharged conveniently once a proof of a preliminary axiom has been given.

Goalstack functions. Auxiliary functions for pushing (**PROVE**) and popping (**POPRL**) goals, rules or axioms to and from the goalstack, saving them in rules, and restoring goal stacks in case of interruptions are called by convenient Rispe commands. An extension for tagging goal stacks with corresponding Rispe proofs is being considered.

Rispe *Procedures.* As Rispe is run in ML, the advanced user has access to all functions and programming facilities of LAMBDA and ML. Nevertheless a simple way of specifying Rispe procedures is supplied. It provides combinators for Rispe commands and accepts formal arguments which are replaced with actual arguments when the procedure is called. A procedure can be applied simultaneously to several specified subgoals.

Example. In order to give a flavour of the use of Rispe commands, in Fig.2 the proof script of a simple hardware proof is displayed. The symbols *FA*, *BV* here for succinctness replace large subterms of actual intermediate goals. The tactics underlying Rispe are still being extended. As a result scripts will be even shorter. Moreover, Rispe as presented in this paper does not address any specific application domain. Proof functions internally using Rispe subtactics along with hardware-specific extensions would add further automation.

4 Conclusion

Most of the work done so far has centred on the numerous ML routines underlying Rispe rather than on case studies. Comparisons of LAMBDA subproofs of hardware verifications and in other areas redone in Rispe are very encouraging, though. The raised level of user interaction, which is enabled by the powerful proof automation functions, leads to reductions up to a factor of 10 in proof development time and script sizes; Rispe scripts are significantly better readable. Users without knowledge of LAMBDA need some basics of ML for being able to write own specifications. Learning about 20 intuitive and uniform Rispe commands suffices for effectively guiding proofs. A menu provides help information.

The PVS approach [6] employs decision procedures for automated reasoning in higher-order logic. The subtactics and convérsions of Rispe are not only

```
PROVE"...";        ⊢ bvVal (n,bv1) + bvVal (n,bv2) + bVal cin =
                     2^(n+1) * bVal (bv_cout(n,cin,bv1,bv2)) +
                     bvVal (n,bv_sum(n,cin,bv1,bv2))
IND"n";
(base case):       ⊢ bVal(bv1 0) + bVal(bv2 0) + bVal cin =
                     2 * bVal(FA_cout(...)) + bVal (FA_sum (...))
ESIMP"{*}";        ⊢ bvVal(bv1 0) + bVal(bv2 0) + bVal cin =
                     2 * bVal(...) + bVal (cin xor bv1 0 xor bv2 0)
CASAN"";           ⊢ TRUE
(step case):         bvVal(n,bv1) + ...+ bVal cin =
                     2^(n+1) * bVal (bv_cout(n,...)) + bvVal(n,bv_sum(...)))
                   ⊢ bvVal(1+n,bv1) + ...+ bVal cin =
                     2^(n+2) * bVal (bv_cout(1+n,...) + bvVal(1+n,bv_sum(...)))
ESIMP"{*}";          ...
ARITH"";           ⊢ bVal(bv1 (1+n)) + bVal(bv2 (1+n)) + bVal(BV_cout n) =
                     2 * bVal(BV_cout (1+n) + bVal(BV_sum(1+n))
CASAN "";          ⊢ TRUE
POPRL "nAddT";
```

Fig. 2. Rispe proof for an n-bit adder

safe and efficient, but also reconfigurable and customizable. Decision procedures can be made part of conversions and tactics, which in Rispe has been done for arithmetic reasoning only. Both approaches demonstrate that the automation achievable in a higher-order system competes well with a first-order prover, while specification and interactive proof steps clearly benefit by higher-order concepts.

Rispe is a first prototype for general-purpose proofs. Application-specific extensions have been started recently, reusing and reconfiguring Rispe tactics. First experience indicates further gains in automation.

References

1. H.Busch, 'Transformational Design in a Theorem Prover', in *THEOREM PROVERS IN CIRCUIT DESIGN*, IFIP Transactions A-10, edited by V. Stavridou, T.F. Melham, and R.T. Boute, pp. 175–196, North-Holland, 1992.
2. H. Busch, 'Rule-Based Induction', in *FORMAL METHODS IN SYSTEM DESIGN - Special Issue on HOL'92*, Kluwer, Vol. 5, Issue 1 & 2, July/August 1994.
3. H. Busch, 'First-Order Automation for Higher-Order-Logic Theorem Proving', *HOL 1994 - 7th International Conference on Higher Order Logic Theorem Proving and its Applications*, edited by T. Melham and J. Camilleri, LNCS 859, Springer, September 21-24, 1994, Malta.
4. S. Finn, M. Fourman, M. Francis, B. Harris, R. Hughes, and E. Mayger, Abstract Hardware Limited, LAMBDA Documentation, 1993.
5. R. Harper, R. Milner, and M. Tofte, 'The Definition of Standard ML, Version 3', University of Edinburgh, LFCS Report Series, ECS-LFCS-89-81, May 1989. '
6. D. Cyrluk. S. Rajan, N. Shankar, and M.K. Srivas, 'Effective Theorem Proving for Hardware Verification', in *2nd Int. Conf. on THEOREM PROVERS IN CIRCUIT DESIGN*, edited by R. Kumar and T. Kropf, LNCS, Springer, 1994.