Developing reuse technology in proof engineering (position paper)

Zhaohui Luo
Department of Computer Science
University of Durham

February 1, 1995

1 Introduction

In the last several years type theory has emerged as an important technology for computer-assisted formal reasoning. The intensive international effort to develop type theory and the associated technology for proof development has resulted in a number of impressive systems based on type theory for program verification and formalisation of mathematics, including ALF [ACN90], Coq [D$^+$91], NuPRL [C$^+$86], and Lego [LP92].

The experience of proof development shows that, to apply our technology of proof development such as systems based on type theory to large real-world application problems (e.g., in formalisation of a large body of mathematics (cf. [Pol94]) and in verification of safety-critical software components), it is very useful and indeed necessary to develop methods and supporting tools for reuse of proofs, programs, and the formalisation or development processes. Many proofs and many proof development processes have striking similarities. Effective reuse will make proof development more efficient and cost-effective so that 'proving in the large' may become feasible in reasonable time limits.

It is important to develop the technology of proof reuse as a step to 'bridge the gap between theory and practice' in computer-assisted formal reasoning. Type theory provides rich abstraction, modularisation and inheritance mechanisms which form a very good basis to support reuse of parameterised proofs, generic theories, and various design decisions during proof and program development (cf. [Luo94a, Luo93]). The potential of type theory has not been sufficiently exploited for reuse and, we believe, is very useful to do so to enhance the productivity and cost-effectiveness of proof development.

2 Reuse in proof development based on type theory

Reusable entities in proof development include not only proof objects, but also proof sketches (or incomplete proofs) and proof strategies adopted in proof development. In a proof development system which supports explicit proof objects, one might classify reusable entities into two different levels, that is, those expressible in the object logical language (e.g. proof objects) and those which may be more conveniently considered at the meta-level (e.g. proof sketches and tactics). However, it is sometimes difficult to make this object/meta level distinction since an expressive object language such as that of dependent types may have the power to express, for example, incomplete proofs or even proof strategies which may usually be treated as meta-level entities.

In fact, type theory does provide useful abstraction and modularisation mechanisms that support representation of proof sketches as well as parameterised/generic proofs. Such an expressiveness has been studied in the context of program specification and data refinement [Luo93], where it is shown how incomplete programs and design decisions such as divide-and-conquer with sharing can be expressed and manipulated in type theory for modular program development. This suggests an interesting approach to reuse in proof development based on type theory and is also a good basis to develop useful libraries of reusable generic proofs and proof sketches. Based on the notions of theory and theory morphism (similar to Burstall's notion of deliverable), research is in progress to design such a library of abstract data types with an inheritance and refinement structure on the basis of the current library of simple inductive types in the Lego system. It will allow users to reuse generic proofs and generic proof sketches to prove their concrete theorems simply by instantiation.

Our suggestion to exploit the power of the object language (type theory) in reuse is not meant to say that the meta-level study of reuse becomes less important. On the contrary, we believe many important issues may only be addressed at the meta-level. For instance, many reusable proof strategies involve application of generic forms of (meta-level) inference rules. It is interesting to investigate the combination of object-level and meta-level reuse techniques. An example is that, in systems that support inductive data types, there are striking similarities (or genericity) in the algorithms that are developed for different inductive types. This similarity is reflected in the general schemata (see, e.g. [CPM90][Luo94a]) and suggests a useful reuse strategy in various algorithm development and their correctness proofs. Such a genericity can only be considered at the meta-level and is being investigated in the context of generic programming and program transformation proofs [Luo94b].

A related interesting topic is to study how a language (e.g. with powerful type structures) more expressive than the current programming languages may be used to implement a proof development system so that the reusable proof scripts and proof strategies such as tactics may be expressed and manipulated [Pol95].

3 Supporting tools and reuse in software engineering

Reuse is an engineering issue, but it needs a sound basis to be supported properly. To develop suitable tools to support reuse in proof development is especially important and some researchers have already done some experimental work in this (e.g. [FH94]). To support our generic library of proofs and proof sketches as described above, we need to build a good navigation tool for the user to find desirable generic proofs/sketches in the library. The existing experience of software reuse is of particular interest in both tool design and methodology study for proof reuse. We believe that proof engineering offers a good (and specific) context to apply the idea of reuse and will hopefully contribute to the methodology and techniques of software reuse as well.

References

[ACN90] L. Augustsson, Th. Coquand, and B. Nordström. A short description of another logical framework. In G. Huet and G. Plotkin, editors, *Preliminary Proc. of Logical Frameworks*, 1990.

[C$^+$86] R.L. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System.* Pretice-Hall, 1986.

[CPM90] Th. Coquand and Ch. Paulin-Mohring. Inductively defined types. *Lecture Notes in Computer Science*, 417, 1990.

[D$^+$91] G. Dowek et al. *The Coq Proof Assistent: User's Guide (version 5.6).* INRIA-Rocquencourt and CNRS-ENS Lyon, 1991.

[FH94] A. Felty and D. Howe. Generalisation and reuse of tectic proofs. *Proc. of 5th Inter. Conf. on Logic Programming and Automated Reasoning*, 1994.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh, 1992.

[Luo93] Z. Luo. Program specification and data refinement in type theory. *Mathematical Structures in Computer Science*, 3(3), 1993.

[Luo94a] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, 1994.

[Luo94b] Z. Luo. Generic programming and program transformation. In preparation, 1994.

[Pol94] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions.* PhD thesis, Edinburgh University, 1994.

[Pol95] R. Pollack. Are tactics feasible? *2nd Conf. on Typed Lambda Calculi*, 1995.