Expander2

Towards a Workbench for Interactive Formal Reasoning

Peter Padawitz

University of Dortmund,
Dortmund, Germany
peter.padawitz@udo.edu

Abstract. Expander2 is a flexible multi-purpose workbench for interactive rewriting, verification, constraint solving, flow graph analysis and other procedures that build up proofs or computation sequences. Moreover, tailor-made interpreters display terms as two-dimensional structures ranging from trees and rooted graphs to a variety of pictorial representations that include tables, matrices, alignments, piles, partitions, fractals and turtle systems.
Proofs and computations performed with Expander2 follow the rules and the semantics of swinging types. Swinging types are based on many-sorted predicate logic and combine *visible* constructor-based types with *hidden* state-based types. The former come as *initial* term models, the latter as *final* models consisting of context interpretations. Relation symbols are interpreted as least or greatest solutions of their respective axioms.
This paper presents an overview of Expander2 with particular emphasis on the system's prover capabilities.

1 Introduction

The following design goals distinguish Expander2 from many other proof editors or tools using formal methods:

- Expander2 provides several representations of formal expressions and allows the user to switch between linear, tree-like and pictorial ones when executing a proof or computation on formulas or terms.
- Proof and computation steps take place at three levels of interaction: the simplifier automates routine steps, axiom-triggered computations are performed by narrowing and rewriting, analytical rules like induction and coinduction are applied locally and stepwise.
- The underlying logic is general enough to cover a wide range of applications and to admit the easy integration of special structures or methods by adding or exchanging signatures, axioms, theorems or inference rules including built-in simplifications.
- Expander2 has an intelligent GUI that interprets user entries in dependence of the current values of global state variables. This frees the user from entering input that can be deduced from the context in which the system actually works.

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 236–258, 2005.

Proofs and computations performed with the system are correct with respect to the semantics of swinging types [16–19]. A swinging type is a functional-logic specification consisting of a many-sorted signature and a set of (generalized) Horn or co-Horn axioms (see section 3) that define relation symbols as least or greatest fixpoints and function symbols in accordance with the initial resp. final model induced by the specification.

Sortedness is only implicit because otherwise the proof and computation processes would become unnecessarily complicated. If used as a specification environment, the main purpose of Expander2 is *proof* editing and not *type* checking. Therefore, the syntax of signatures is kept as minimal as possible. The only explicit distinction between different types is the one between constants on the one hand and functions and relations on the other hand, expressed by the distinction between first-order variables (`fovars`) and higher-order variables (`hovars`). Proofs or computations that depend on a finer sort distinction can always be performed by introducing and using suitable *membership predicates*.

The prover features of Expander2 do not aim at the complete automation of proof processes. Instead, they support *natural* derivations, which humans can comprehend and thus control easily. Natural deduction avoids skolemization and other extensive normalizations that make formulas unreadable and thus inappropriate for interactive proving. For instance, the simplifier (see Section 5), which turns formulas into equivalent "simplified" ones, prefers implication to negation.

Of course, many conjectures can be proved both comprehensibly and efficiently without any human intervention into the proof process. Such proofs often follow particular schemas and thus may be candidates for derived inference rules. However, proofs of program correctness usually do not fall into this category, especially if induction or coinduction is involved and the original conjecture must be generalized in a particular way.

In fact, the simplifier of Expander2 performs certain normalizations. But they are in compliance with natural deduction and deviate from classical normalizations insofar as, for instance, implications and quantifiers are not eliminated by introducing negations and new signature symbols, respectively. On the contrary, the simplifier eliminates negation symbols by moving them to literal positions and then are removed completely by transforming negated (co)predicates into their complements. Axioms for relations and their complements can be constructed from each other: If P is a predicate specified by Horn axioms, then these axioms can be transformed systematically into co-Horn axioms for the copredicate *not_P*, and vice versa. This follows from the fact that relation symbols are interpreted by the least resp. greatest solutions of their axioms provided that these are negation-free and thus induce monotonic consequence operators [16–18].

Expander2 has been written in O'Haskell [12], an extension of Haskell [8] with object-oriented features for reactive programming and a typed interface to Tcl/Tk for developing GUIs. Besides providing a comfortable GUI the overall design goals of Expander2 were to integrate testing, proving and visualizing deductive methods, to admit several degrees of interaction and to keep the system open for extensions or adaptations of individual components to changing demands.