TAS and IsaWin: Tools for Transformational Program Development and Theorem Proving

Christoph Lüth, Haykal Tej, Kolyang, and Bernd Krieg-Brückner*

Bremen Institute of Safe Systems (BISS), FB 3, Universität Bremen
{cxl,ht,kol,bkb}@informatik.uni-bremen.de

Introduction

We will present two tools, TAS and IsaWin, for transformational program development and theorem proving respectively, based on the theorem prover Isabelle [9]. Their distinguishing features are a graphical user interface based on the principle of direct manipulation, and a generic, open system design which allows the development of a family of tools for different formal methods on a sound logical basis with a uniform appearance.

The aim of these tools is to provide a user-friendly framework for formal program development, supported by a powerful theorem prover, and generic over the actual formal method employed. By embedding a formal method into the theorem prover Isabelle, instantiating the generic graphical user interface, and adding customised proof support, we obtain a tool for that formal method which hides the details of the encoding; we call this an *encapsulation* of the formal method.

The Transformation Application System TAS

TAS is a system for formal transformational program development. It hides the details of the encoding of transformational development in Isabelle from the user, leaving him with the main design decisions of transformational program development: which rule to apply, and how to instantiate its parameters. TAS further allows to abstract transformational developments to transformation rules.

Figure 1 shows a screenshot of TAS. In the centre, we see the *notepad* window where icons represent transformation rules, transformational developments and parameter instantiations. The notepad does not contain all known transformations, only those which are explicitly placed there by the user. The user can select from all known transformations by the transformation chooser.

A transformational development can be opened in the *construction area*, where it can be further developed. Transformations are applied by dragging the icons into the construction area from the notepad (or directly from the transformation chooser). The history of the transformational development, and the

* This work has been supported by the German Ministry for Education and Research (BMBF) as part of the project UniForM under grant No. FKZ 01 IS 521 B2.

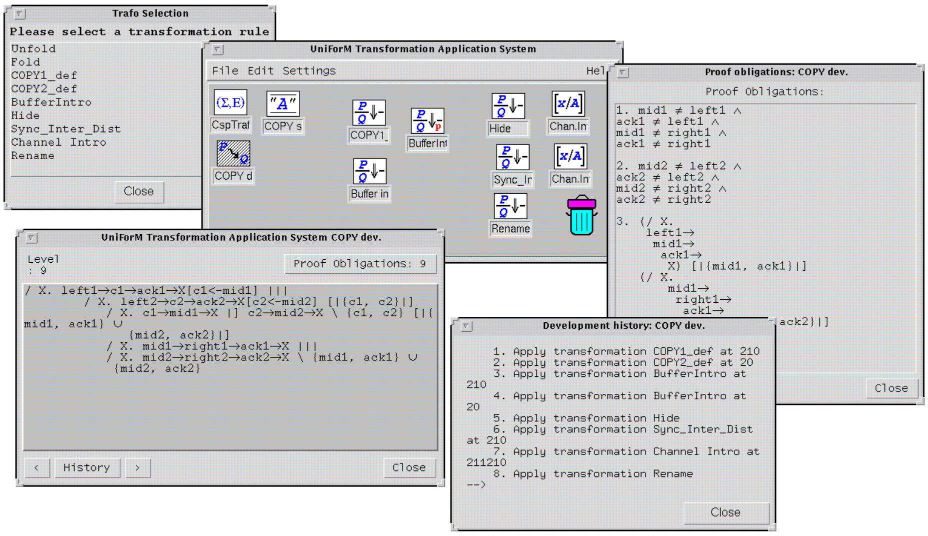

Fig. 1. Transformational Program Development with TAS. Windows, clockwise: transformation chooser, notepad, proof obligations, history and the construction area.

current proof obligations can be displayed in separate windows. Proof obligations are discharged by dragging them into IsaWin, where they turn into ongoing proofs.

IsaWin

IsaWin is a graphical user interface for the theorem prover Isabelle. Here, it is also used to show the proof obligations arising from transformational developments in TAS. It allows access to all of Isabelle's basic functionality, such as forward and backward resolution, simplification, and construction of tactics. It further provides a sophisticated replay mechanism in which external changes are propagated to all objects depending on the changed external objects.

Figure 2 shows a screenshot of IsaWin. We can see the notepad in the upper window, where icons represent theorems, rewriting sets, theories, ongoing proofs and texts. The construction area in the lower window shows an ongoing proof. Theorems and rewriting sets are applied by dragging them into the construction area, where tactical operations such as backward resolution or simplification are triggered, depending on the settings of the tactical buttons on the left of the construction area, and on the type of the object.

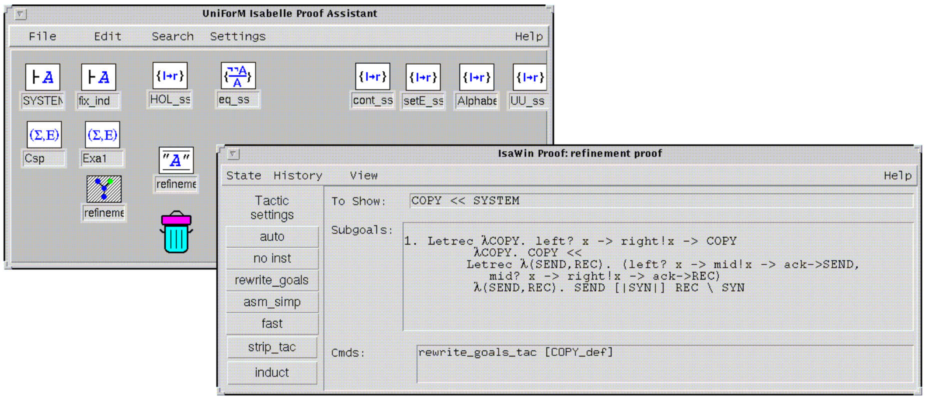

Fig. 2. Theorem Proving with IsaWin.

System Architecture

Both TAS and IsaWin are entirely implemented in Standard ML (SML) as a conservative extension of Isabelle. This approach allows us to take advantage of SML's powerful modularisation mechanisms, and at the same time preserve logical consistency. In particular, we provide a generic graphical user interface GenGUI which is a parameterised module that yields graphical user interfaces by instantiating its parameter. For its implementation we have developed a functional encapsulation of the interface-building library Tcl/Tk into SML, called sml_tk [6]. Consistency is preserved since SML's type system restricts the construction of new theorems. For more details of the system architecture, we refer to [7].

A consequence of this approach is that all tools obtained by instantiating the generic graphical user interface GenGUI have a uniform visual appearance (cf. Fig. 1 and Fig. 2). Their main windows always comprise the notepad and the construction area. The notepad contains icons representing the objects, which can be dragged, moved and dropped onto each other, whereas the construction area allows a more refined manipulation of an object's internals, providing history navigation and replay mechanisms at a generic level.

Moreover, both TAS and IsaWin can be instantiated with any formal method or logic embedded into Isabelle. For TAS, the formal method has to provide a notion of correctness-preserving refinment, such as logical implication in higher-order logic or process refinement in CSP. We will present instantiations with the encapsulations of CSP [10], Z [2, 4] and CASL [8].

The UniForM Workbench

TAS and IsaWin have been developed during the UniForM project [3], the aim of which is to integrate different formal methods in a logically consistent way. The

UniForM Workbench [1, 5] embeds TAS and IsaWin into an integrated software development environment, providing servives such as type-safe communication with cooperating tools, persistent and distributed storage, version and configuration management.

Obtaining TAS and IsaWin

In time for the presentation, prototypical versions of TAS and IsaWin will be available from `http://www.informatik.uni-bremen.de/~agbkb/`.

References

1. E. W. Karlsen. *Tool Integration in a Functional Setting.* PhD thesis, Universität Bremen, 1998.
2. Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics — 9th International Conference*, number 1125 in LNCS, pages 283 –298. Springer, 1996.
3. B. Krieg-Brückner. UniForM perspectives for formal methods. In *International Workshop on Current Trends in Formal Methods*, To appear in LNCS. Springer, 1998.
4. C. Lüth, E. W. Karlsen, Kolyang, S. Westmeier, and B. Wolff. Hol-Z in the UniForM-workbench – a case study in tool integration for Z. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98: The Z Formal Specification Notation*, volume 1493 of *LNCS*, pages 116–134. Springer, 1998.
5. C. Lüth, E. W. Karlsen, Kolyang, S. Westmeier, and B. Wolff. Tool integration in the UniForM workbench. In *Workshop on Tool Support for System Specification, Development, and Verification*, Advances in Computing Science. Springer, June 1998. To appear.
6. C. Lüth, S. Westmeier, and B. Wolff. sml_tk: Functional programming for graphical user interfaces. Technical Report 7/96, Universität Bremen, 1996. See also the sml_tk home page at `http://www.informatik.uni-bremen.de/~cxl/sml_tk/`.
7. C. Lüth and B. Wolf. Functional design and implementation of graphical user interfaces for theorem provers. *Journal of Functional Programming.* To appear.
8. T. Mossakowski, Kolyang, and B. Krieg-Brückner. Static semantic analysis and theorem proving for CASL. In *12th Workshop on Algebraic Development Techniques*, number 1376 in LNCS, pages 349– 364. Springer, 1997.
9. L. C. Paulson. *Isabelle - A Generic Theorem Prover.* Number 828 in LNCS. Springer, 1994.
10. H. Tej and B. Wolff. A corrected failure-divergence model for CSP in Isabelle/HOL. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Proceedings of the FME '97 — Industrial Applications and Strengthened Foundations of Formal Methods*, number 1313 in LNCS, pages 318– 337. Springer, 1997.

System Requirements

Hardware:

- Sun Sparc or ULTRA-Sparc; on other systems, availability of Standard ML of New Jersey (Version 109) is a precondition;
- At least 96MB of memory to run, more to compile (only necessary on non-Sparc systems);
- Disk space: 100 MB.

Software:

- Solaris 2.5 or 2.6;
- Tcl/Tk Version 8.0, Netscape Version 4 (can be installed by us if required).