A Program Refinement Tool

D. Carrington\textsuperscript{1}, I. Hayes\textsuperscript{1}, R. Nickson\textsuperscript{2}, G. Watson\textsuperscript{3} and J. Welsh\textsuperscript{3}

\textsuperscript{1}Department of Computer Science and Electrical Engineering, and the Software Verification Research Centre, The University of Queensland, Australia;
\textsuperscript{2}School of Mathematical and Computing Sciences, Victoria University of Wellington, Wellington, New Zealand;
\textsuperscript{3}Software Verification Research Centre, The University of Queensland, Australia

Keywords: Refinement calculus; Refinement tools; Window inference

Abstract. The refinement calculus for the development of programs from specifications is well suited to mechanised support. We review the requirements for tool support of refinement as gleaned from our experience with existing refinement tools, and report on the design and implementation of a new tool to support refinement based on these requirements.

The main features of the new tool are close integration of refinement and proof in a single tool (the same mechanism is used for both), good management of the refinement context, an extensible theory base that allows the tool to be adapted to new application domains, and a flexible user interface.

1. Introduction

Formal methods are increasingly seen as an important technique for improving the quality of software development [BoH95]. The prime area for applying formal methods to software development is high-integrity systems, most importantly, safety-critical applications where the cost of failures is measured in terms of injuries and lives, but also systems where the cost of failure in economic terms is high.

Formal methods, by their rigorous nature, generate a lot of inter-related detail that needs to be managed if the benefits of formality are to be achieved. To date, tools that support formal methods have not been widely adopted even though tools are often seen as necessary to achieve industrial usage of formal methods.
This paper describes the result of a research project to improve the available tool support for the refinement calculus. The project investigated requirements for such tools and reviewed existing tools before developing a new tool known as PRT. PRT is a research prototype but it demonstrates some innovative ways of supporting the refinement calculus and managing the web of detail that results from using the calculus. The distinctive features of the tool are:

- the refinement tool and theorem prover are integrated, with the same development paradigm being used for both refinement and proof;
- support for contextual information representing preconditions, types and invariants;
- an extensible theory base, supporting the Z mathematical toolkit and allowing adaptation to new application domains;
- a flexible user interface that supports construction and reuse of program derivations.

The paper is structured as follows: Section 2 provides an overview of the refinement calculus and an example; Section 3 analyses roles for tools supporting users of the refinement calculus and hence determines tool requirements; Section 4 reviews previous tools for the refinement calculus; Section 5 describes the new tool PRT; and Section 6 assesses PRT relative to the requirements derived in Section 3.

2. Refinement

This section introduces the reader to the concept of refinement as a formal method of developing programs from specifications. It focuses on the method known as the refinement calculus which was developed by Back, Morgan, and Morris [Bac88, Mor94, Mor87] and discusses how the calculus is used.

2.1. Goal of Refinement

Refinement methods seek to establish a sequence of transformations that, when applied to a formal abstract specification, generates an executable program that satisfies the original specification. The sequence of transformations is an idealised record of the development process [PaC86] that provides explicit traceability between specification and code in both directions. Refinement cannot be completely automated: not all specifications can be implemented, and each transformation represents a design decision leading towards an executable program that meets any nonfunctional requirements such as performance bounds.

Software design involves two kinds of decisions: how to represent information and how to process it. Consequently, transformational approaches such as refinement need to be able to describe both types of decisions. Data refinement [Hoa72, Mor89, GaM93] is the class of transformations that replaces one data representation by another, while procedural refinement is the class of transformations that incorporates processing decisions.

---

1 PRT is available for anonymous FTP from ftp.cs.uq.edu.au/pub/SVRC/software/ — it runs under Solaris 2.4; a Linux version is expected to be available in the near future.