Mechanised support for sound refinement tactics

Frank Zeyda¹, Marcel Oliveira² and Ana Cavalcanti¹

¹ Department of Computer Science, University of York, Heslington, York, UK. E-mail: zeyda@cs.york.ac.uk
² Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, Brazil.

Abstract. ArcAngel is a tactic language devised to facilitate and automate program developments using Morgan’s refinement calculus. It is especially well suited for the specification of high-level refinement strategies, and equipped with a formal semantics that additionally permits reasoning about tactics. In this paper, we present an implementation of ArcAngel for the ProofPower theorem prover. We discuss the underlying design, explain how it implements the semantics of ArcAngel, and examine the interplay between ArcAngel tactics and the native reasoning support of the prover. We also discuss several extensions of ArcAngel that have been entailed by our implementation effort. They are of practical importance and provide a unification of the related tactic languages Angel and ArcAngelC. Our main result is a mechanisation that reflects directly the ArcAngel semantics, and can be used with any programming model for refinement. The approach can be used to support other formal tactic languages using other theorem provers.

Keywords: Tactic language; Refinement; Automation; Z; Unifying theories

1. Introduction

Morgan’s refinement calculus [Mor98] supports the derivation of programs from specifications by a series of correctness-preserving refinement steps. Each step is justified by the application of a refinement law that guarantees that the program obtained correctly implements its specification.

The ArcAngel language [OCW03] supports the documentation and automation of recurring sequences of refinement steps. It has a formal semantics, and an extensive set of algebraic laws that support reasoning about tactics. ArcAngel is an extension of Angel [Mar94, MGW96], a general-purpose tactic language; they owe their name to their support for the use of angelic choices in the process of solving proof goals.

A number of tools have been proposed that address the issue of interactive and automatic refinement [CHN’94]. Groves et al. and Vickers [GNU92, Vic90] present tools that can support refinement via tactics in Morgan’s calculus. They, however, do not give a formal semantics to the underlying refinement language, and thereby it is not possible to independently verify the laws. A similar restriction applies to the Gabriel extension of the interactive refinement tool REFINE [OXC04]. At the other end of the spectrum, von Wright presents a proof-based approach that mechanises refinements in HOL [Wri94]. Derivation is sound in this work, however, the concern is not to provide a high-level language for refinement. Use of the tool requires direct interaction.
with the window-inference toolkit that drives the tool underneath. The PRT tool [CHN+98] aims to address both extensibility and support for sound derivation, but focuses on user interaction rather than automation. The Ergo theorem prover [UW94], which is implemented in Prolog, was extended to use Angel, but not ArcAngel, as a tactic language [MNU97].

In this paper, we present an implementation of ArcAngel; our main contribution is an approach to provide sound automated support for tactic-based refinement. Our mechanisation (and its underlying design) is distinctive. (1) It is based on a tactic language with a formal semantics, so that we enable reasoning about the tactics themselves. (2) It supports the sound derivation of refinements with the protection of the LCF approach. (3) It enables independent verification of refinement laws with respect to a program model. (4) It is extensible and can be used in conjunction with arbitrary refinement languages and underlying semantic models. (5) It handles proof obligations and their discharge via the same mechanisms that apply refinement tactics. (6) It supports the seamless integration of native reasoning facilities of the prover.

None of the existing tools do justice to all these features at once. On the one hand, we have refinement editors tailored to a particular language. They do not provide proof support to validate refinements, and are often interactively driven with limited support for automation. On the other hand, there are works that tie in with automatic provers, but do not provide a level of abstraction that facilitates the development of readable and powerful high-level strategies nor permit integration of custom languages. Our work bridges this gap by supporting a user-friendly, high-level tactic language that is suitable for the automation of complex strategies, and allows the user to combine native proof support and tactics in a flexible manner.

Main-stream provers do not generally perform automatic backtracking upon tactic failure. Our tool makes this feature available by faithfully implementing ArcAngel’s angelic choice. In addition, conventionally tactics apply to goals (sequents). The purpose of a refinement strategy, on the other hand, is to transform program expressions. Our tool provides support for coupling transformation-based reasoning and goal-based proof. An ArcAngel tactic is defined in terms of refinement laws, and, whenever possible, produces a theorem that establishes that a given original specification is refined by a program (derived by the application of the tactic). Proof obligations are either discharged or kept as hypotheses; they can be predicates over the program variables (or logical constants) or refinement conjectures.

We also identify contributions to ArcAngel itself: we propose four extensions. A first extension is the notion of a program model, which formalises the assumptions underlying the sound use of ArcAngel. This allows the core of our tool to be used with computational models other than Morgan’s calculus. It also makes the tool useful in proving equivalences of programs in addition to genuine refinements.

Another extension to ArcAngel is a new tactical that supports the combination of tactics to refine programs with tactics to discharge (or simplify) the raised proof obligations. This allows us to take full advantage of the capabilities of a theorem prover in mechanising and automating sound refinements.

As a further extension, we have also addressed the treatment of expressions used as arguments of tactics and law applications. The original semantics of ArcAngel makes a simplifying assumption that all expressions are fully evaluated. From a practical point of view, this is not always adequate; for instance, the shape of expressions may have an effect on the use of laws that are applicable to a program. We provide an annotation mechanism that supports fine control of expression evaluation.

Finally, we consider the practical treatment of termination in ArcAngel. We define and implement extra tactics for definition of recursive tactics. They limit the number of recursive calls to avoid nontermination. They also allow us to decide whether, by exceeding the allowed number of recursive calls, the tactic aborts or fails. This gives more control to the construction of useful tactics.

A third contribution of this paper is a mechanisation of Morgan’s refinement calculus in ProofPower-Z, which is based on an encoding of the Unifying Theories of Programming [HJ98, ZC08]. It completes the implementation of ArcAngel by defining its program model, but can also be used independently. In addition, it provides further validation for the work in [ZC08], which supports the definition of programming theories.

Our work leads in parts to a unification of ArcAngel with its kin Angel, as well as the more specialised derivate ArcAngelC, a variant tailored for action and process refinement in the Circus language [OC08]. A by-product of this unification is a framework that fosters the development of other derivatives of Angel.

To demonstrate our approach, we have used ProofPower, a flexible and extensible theorem prover based on HOL. It has an open architecture, and has been successfully employed on industrial projects (for example, in the verification of avionics control systems [AC05]). ProofPower also provides an embedding of Z [WD96] known as ProofPower-Z. This is useful in defining semantic models for refinement languages, and we also take advantage of the expressiveness of Z in examples discussed later on.