

## Object-Oriented Specification: Analysable Patterns & Change Management

Formal techniques have been shown to be useful in the development of correct software. But the level of expertise required of practitioners of these techniques prohibits their widespread adoption. However, attempts to integrate formal specification techniques with modern, often agile, software development practices are becoming more successful. For example, the lightweight modelling framework Alloy and so-called behavioural interface specification languages, such as the Java Modelling Language (JML) for Java and Spec $\sharp$  for C $\sharp$ , are designed for programmers to specify software modules incrementally as those modules are developed. Several easy-to-use and complementary tools, such as the Alloy Analyzer and static and runtime checkers for JML and Spec $\sharp$ , also provide effective support for testing and verification.

However, these new techniques have at least the following three shortcomings:

- Behavioural interface specification languages, tailored to programmers, do not yet have development environments that facilitate the construction of consistent specifications, as provided for Alloy by the Alloy Analyzer.
- Many of the tools that support the analysis of specifications expressed in these languages give misleading feedback in cases where the specification is inconsistent. For example, a tool may simply report a property to be true without indicating that it is vacuously true because of an inconsistency. For the non-expert specifier, it is easy to introduce errors accidentally in a specification through editing, since many existing tools do not explicitly check for consistency.
- Moreover, when a specification is changed, with properties either added, removed or altered, the results of previous analyses typically become invalid and testing or verification must be repeated.

This thesis is therefore concerned with the development of an environment to facilitate the construction of correct specifications. In particular, three specific contributions address the above shortcomings:

- The development of a lightweight declarative object-oriented specification language, called *Loy*, tailored to the specification of object-oriented programs, but which, unlike JML and Spec $\sharp$ , can be automatically analysed by the Alloy Analyzer, through a formal encoding of Loy into Alloy. In particular, an implementation is presented that automates the encoding and allows a specifier to check Loy specifications for consistency by harnessing the SAT technology underlying the Alloy Analyzer. This implementation is the environment in which the second two contributions are employed.
- The identification and implementation of *patterns of analysis* that guide a non-expert specifier through some of the pitfalls of analysing a declarative specification; in particular, the patterns expose many cases of vacuity in a specification. Support for the patterns of analysis is integrated into the above implementation.
- The development of foundations for a change management system for declarative specifications that minimises the number of SAT calls needed to recheck an edited specification. The system monitors the values and interdependencies of expressions in a specification and, following a change, infers new values for expressions from unedited parts of the specification, where possible, rather than rechecking through SAT. Although the work is presented in a SAT setting and integrated into the above implementation for illustration, it should be applicable to any logic with a potentially incomplete decision procedure.