Translating VDM to Alloy

Kenneth Lausdahl

Department of Engineering, Aarhus University
Finlandsgade 24, DK-8200 Aarhus N, Denmark
lausdahl@cs.au.dk

Abstract. The Vienna Development Method is one of the longest established formal methods. Initial software design is often best described using implicit specifications but limited tool support exists to help with the difficult task of validating that such specifications capture their intended meaning. Traditionally, theorem provers are used to prove that specifications are correct but this process is highly dependent on expert users. Alternatively, model finding has proved to be useful for validation of specifications. The Alloy Analyzer is an automated model finder for checking and visualising Alloy specifications. However, to take advantage of the automated analysis of Alloy, the model-oriented VDM specifications must be translated into a constraint-based Alloy specifications. We describe how a subset of VDM can be translated into Alloy and how assertions can be expressed in VDM and checked by the Alloy Analyzer.

1 Introduction

The Vienna Development Method (VDM) [1,2,3] supports modelling and analysis at various levels of abstraction, using a combination of implicit and explicit definitions of functionality, and has a strong record of industrial application [4] for design and specification of software systems. However, one of the limitations of the implicit VDM specifications is the lack of tool support. Existing VDM tools, Overture [5] and VDM-Tools [6], only provide limited help with the difficult task of validating that an implicit VDM specification captures the intended meaning. The existing tools include standard features like parsers and type checkers, but this only ensures that specifications are correct with regards to syntax and type constraints. The only tool support for semantic validation is a proof obligation generator but this still leaves the difficult task of discharging the proof obligations. Theorem provers such as [7] can be used to discharge the generated proof obligations through an automates translation of VDM to HOL [8,9], but using a theorem prover is usually complicated and requires an expert.

A different approach is to validate the specification through testing by running an interpreter [10]. This is possible for explicit VDM specifications which can be interpreted with actual values. The same approach can be used for implicit specifications through the use of a tool that is able to automatically create explicit definitions of all functions and operations based on their post-conditions, for a subset of the language, and then validate the generated specification through the standard interpreter. This approach has been taken for a subset of VDM that required post-conditions to follow a particular template using conjunctions to separate constraints on the return value [11,12]. While

E.B. Johnsen and L. Petre (Eds.): IFM 2013, LNCS 7940, pp. 46–60, 2013.

this enables interpretation, it still requires user input, like test cases, whereas an alternative based on model checking requires less or no user input and still provides a larger coverage than testing. Furthermore, such an approach enables easy detection of contradictions in pre- and post-conditions when functions and operations are combined at a system level.

The Alloy Analyzer is a bounded model finder that has proved to be useful for validating specifications in the Alloy language [13]. The analyzer can *find* instances of Alloy specifications, as well as checking user defined assertions. The analyzer can provide immediate visual feedback when an instance is found or present a *core* containing the top level formulas if no instance could be found.

In this paper, we present a translation of VDM to Alloy [13] thereby enabling VDM specifications to be checked by the Alloy Analyzer. This enables users to get immediate feedback both in the form of generated alloy instances, that can be visually displayed and from user-specified properties. We identify a subset of VDM that can be automatically translated and checked in the Alloy Analyzer and give a preserving semantics for the translation, thus identifying where a translation becomes infeasible. Others have already shown that languages like Z, B, Event-B and UML can be translated [14,15,16,17] to Alloy and benefit from the automated analysis the Alloy Analyzer provides. The VDM language does not contain any direct way to capture system properties, also called validation conjectures [18, p. 191], which is equivalent to the assertions used in model checking but we show a new way of using existing VDM expressions to express such properties.

The structure of this paper is as follows. Section 2 describes the languages VDM and Alloy and how they relate. Section 3 formally defines a subset of VDM and defines the translation rules and the limitations of the translation. Section 4 describes how VDM specifications can be checked using this translation. Section 5 describes the findings discovered by applying the translation to a number of VDM specifications. Finally, Section 7 discusses the contribution of this paper.

2 VDM Models and Alloy Instances

The Vienna Development Method is one of the longest established model-oriented formal methods, and was originally developed at the IBM laboratories in Vienna in the 1970's. The VDM Specification Language is a higher-order language which is standardised by the International Organization for Standardization (ISO), and has a formally defined syntax, and both static and dynamic semantics [19,20]. The VDM language employs a three valued logic; values may be true, false or bottom (undefined), using Logic of Partial Functions (LPF) where the order of the operands is unimportant[21][1]. Models in VDM are based on data type definitions built from simple abstract types using booleans, natural numbers, characters and type constructors for record, product, union, map, (finite) set and sequences. Type membership may be restricted by predicate invariants. Persistent state is defined by means of typed variables, again restricted by invariants. Operations that may modify the state can be defined implicitly, using standard

[1] The existing VDM interpreter is, however, depended on the operand order.