The What, How, and When of Formal Methods

S

Aristides Dasso
Universidad Nacional de San Luis, Argentina

Ana Funes
Universidad Nacional de San Luis, Argentina

INTRODUCTION

Formal Methods (FM) is an area of Software Engineering. They comprise a collection of methodologies and related tools, employing a mathematical basis –as do most engineering disciplines–, to construct its products. Although FM are part of Software Engineering, they extend their scope to the development not only of software products but also of hardware systems.

The goal of this article is to give answers to questions such as *what are Formal Methods*, *how are Formal Methods implemented*, *how are they used in Software Engineering*, *when should they be used*, among other related questions.

The chapter starts answering the question of *what are FM*; also the aims of FM are stated at the same time that their main characteristics are presented.

An example that shows *how* FM can be used to help specifying software requirements as well as the rest of the stages in a software development process is given as answers to the questions *how are FM implemented* and *how FM are used in Software Engineering*.

A discussion about *when they should be used*, explaining why they should be employed when the software system is required to be as secure and reliable as possible and how they can also be used as a complement to traditional development methods, is also provided.

A section on the state of the art in FM, providing an analysis of Lightweight FM and the growing impact that Model Checking is having in the software and hardware industry as an automatic FM for system verification, is presented. Finally, a discussion on the use of automatic analyzers like Alloy, which replace conventional analysis based in theorem proof by a "non-complete" analysis based in the examination of cases, is also given.

BACKGROUND

The 'What' of FM

What Are Formal Methods

FM contain a wide range of methods and related tools oriented to the production of secure and reliable software and hardware systems by employing a logic-mathematical basis.

As traditional development methods, FM consist of a set of techniques and supporting tools to assists developers during the whole software development process. The fundamental difference with traditional methods is that FM are based on mathematics and formal logic leading to unambiguous specifications, where desired properties of a system under development can be formally expressed and verified.

The adoption of FM makes possible the specification and the verification of software and hardware systems. They provide the mathematical tools to develop new concrete formal specifications and, eventually, executable code from abstract

DOI: 10.4018/978-1-5225-2255-3.ch662

formal specifications, where all the development steps can be formally verified.

Therefore, in contrast to traditional development methods, FM use mathematical proofs as a complement to software testing in order to verify the correctness of the system under development.

There are a number of different Formal Methods, each having its own notation, methodology and supporting tools; a comprehensive list of FM can be found in Formal Methods Wiki (Bowen). Formal notations or formal specification languages are used to produce formal specifications of software and hardware systems. There are different styles in formal notations, and there also are different degrees of rigor in development. Formal specifications can be written either using *abstract* or *concrete* style. Some formal specification languages adopt a *property-oriented* style, allowing the creation of *algebraic specifications*, where the desired properties of the system under development are given by axioms, in a purely declarative way. This kind of specifications is called *algebraic* because specifications are seen like heterogeneous algebras. A different style for specifications is the use of *model-based* notations, which make use of concrete data types (integers, reals, sets, list, etc.). They are more concrete than property-oriented specifications. However, in general, formal specification languages favors abstraction, being oriented to answer the question *what are the software requirements* of a system more than *how the requirements are going to be implemented*, i.e. they are oriented to describe what a system should do more than how it should be done.

Formality in the use of FM can go from the maximum degree of rigor, i.e. the use of formal specification and formal verification of the whole system to the use of formalizations for requirements specification only.

Some confusion can arise with the use of terms *formal language*, *formal notation*, *formal system* and *formal method*. In (Alagar, V. S. & Periyasami, K., 2011) the differences are clearly stated. In Table 1 a synoptic view of the differences between those terms is shown.

A *formal notation* is a language that has both its *syntax* and *semantics* formally defined. Specification languages used in FM and programming languages are formally defined by their syntax and semantics. Syntax drives the validation of the language constructs. Strictly speaking the syntax of a language is given by the formal grammar of the language. There are a number of methods to do this, among them BNF (Backus – Naur Form). These methods are widely used to represent the syntax of programming languages, specification languages, etc.

Semantics, on the other hand, is a way of associating meanings to the language's valid constructs. Roughly speaking it can be said that given a language $\mathcal{L}$, the semantics of $\mathcal{L}$ can be represented by a pair $\left(\mathcal{U}, \mathcal{I}\right)$, where $\mathcal{U}$ is the universe of values, i.e. the set of all possible values (numbers, characters, Boolean values, etc.), and $\mathcal{I} : \mathcal{L} \rightarrow \mathcal{U}$ is an interpretation function that assigns to each construct of the language $\mathcal{L}$ a value in the universe of values $\mathcal{U}$.

There are different styles for giving language semantics. *Operational semantics* is concrete; it may define an abstract machine, and is not well suited for proofs. In *Denotational semantics* a denotation (usually a mathematical object) is given to each phrase or construct of the language; since it is abstract, it is well suited for proofs; *Axiomatic semantics* is also abstract; here each phrase or

Table 1. A view of formal methods, formal systems and formal languages

<table>
<tr><td rowspan="3">Formal Method</td><td rowspan="2">Formal System</td><td>Formal Notation</td><td>Formal Language: formal syntax + formal semantics.</td></tr>
<tr><td colspan="2">Proof system: axioms + inference rules</td></tr>
<tr><td colspan="3">Automatic tool support for specification, proof assistance, code generation, etc.</td></tr>
</table>