Formal Methods Overview

Ana Funes
Universidad Nacional de San Luis, Argentina

Aristides Dasso
Universidad Nacional de San Luis, Argentina

1. INTRODUCTION

In general, all engineering branches use mathematics or mathematical based tools in some if not all of their applications. Mathematical based tools have been proved exceedingly good in helping to describe, predict, design and develop both large and small engineering projects. Although this is not a very common practice in Software Engineering nowadays, this trend seems to be changing by the use of Formal Methods, especially by the application of Lightweight Formal Methods and Model Checking.

Formal Methods (FM) is an area of Software Engineering that encompasses a wide range of methodologies and related tools geared to the production of software employing a mathematical basis. There are a number of different FM each having its own methodology and tools, specially a formal specification language. This article gives an overview of the subject describing the main aspects of FM, the main differences between FM, formal systems and formal languages; it presents a classification of FM and formal specifications and it discusses different degrees of formality in software development. We also discuss the use of Lightweight FM, Model Checking and the combined application of FM with other traditional methods. Finally, an example of a formal specification is presented and pro and cons in the use of FM are analyzed in the conclusion of the article.

2. BACKGROUND

Formal Methods (FM) cover a wide range of methodologies that employ mathematical tools in Software Engineering. FM are a collection of methodologies and related tools, geared to the production of software employing a mathematical basis. There are a number of different FM each having its own methodology and tools, specially a formal specification language.

Hinchey and Bowen (1995) establish that "Formal Methods allow us to propose properties of the system and to demonstrate that they hold. They make it possible for us to examine system behavior and to convince ourselves that all possibilities have been anticipated. Finally they enable us to prove the conformance of an implementation with its specification."

Most FM are based mainly on the use of formal specifications –for which they normally have a language to express them– and the possibility of achieving formal verification of all or part of the developed software. Sometimes they propose also a process to be followed using a formal language during software development as well as other tools for verification such as an assisted prover or a model checker.

The aims of FM can vary according to the different methodologies they support, but they all share a common goal: the production of software with the utmost quality. To achieve this, different FM have developed not only a theory but also different tools to support their respective formal processes. The existence of a large and growing number of languages, methods and tools that promote the application of formal specifications shows the growing interest in this area (see http://formalmethods.wikia.com/wiki/Formal_methods).

3. FORMAL LANGUAGES, SYSTEMS, AND METHODS

There is some confusion in the use of terms *formal language*, *formal notation*, *formal system* and *formal method*. Alagar and Periyasamy (2011) make clear the differences among a formal language, a formal system

DOI: 10.4018/978-1-4666-5888-2.ch704

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

Formal Method	**Formal System**	**Formal Notation**	*Formal Language:* formal syntax + formal semantics.
		Proof system: axioms + inference rules	
	Automatic tool support for specification, proof assistance, code generation, etc.		

and a formal method (see Table 1). According to them, a *formal notation or language* is a language that has both its syntax and semantics formally defined.

The language syntax establishes which the valid constructions in the language are. The semantics associates a meaning to each valid language construction. Essentially, the semantics of a language L is given by a pair (U, I) where U is the universe of possible values (e.g. Integers, Reals, Booleans, etc.) and I is an interpretation function $I: L \rightarrow U$ such that it assigns a value in U to each element in L. In other words, the language's syntax and semantics determine what kind of expressions we can write in the language and what these expressions mean.

When a formal language also has a deductive mechanism that allows conducting formal proofs of system properties by using a set of axioms and rules of inference then we can say that we are in the presence of a *formal system*.

An axiom is a property that is considered valid in the formal system while a rule of inference is a syntactic rule o function that given some premises returns a conclusion. Examples of rules of inference are *Modus Ponens* and *Modus Tollens.*

Modus Ponens

```
If P, then Q.
P.
Therefore, Q
```

Modus Tollens

```
If P, then Q.
¬Q
Therefore, ¬P.
```

Finally, we can say we are in the presence of a *formal method* if the formal system has automatic support for writing specifications and helping in proving properties. The method also tell us what steps we need to follow in order to write and validate the specifications, going from the most abstract specifications that capture customer requirements to more concrete specifications and eventually to an implementation.

4. STYLES AND DEGREES IN FORMALITY

An interesting question is why there are so many flavors of FM. There are undoubtedly several possible answers to this question. Besides the growing interest in FM, we can argue that there are many ways to describe a system and not everyone agrees on a particular style. Why are there so many different programming languages? The answer to this question could be also the answer to our question. Sometimes the selection of a particular specification language has a lot to do with its visual appearance. Some people feels comfortable using graphical notations, others prefer the use of text. Graphics are more intuitive and facilitates communication with non-experts while textual notations are well suited for machine support. Some users adopt a mixture of them in order to get the benefits of both. In any case, it is important to note that not all FM address the same problem. Several characteristics of a specification language affect its applicability in software development. They determine its scope, focus, and the extent of automation and proof support. Some of them are geared to system design, others to domain description, some deal with time while others do not include it, etc. These characteristics are discussed in this section.

We can write formal specifications by using a formal language; however, we can adopt different styles depending on the characteristics of language employed. In Table 2, we show one possible classification of formal specification styles. These styles are not unique and they can be used in combination.

Some languages support property-oriented (also referred as algebraic or axiomatic) specification style