Chapter XVII
Formal Methods for Specifying and Analyzing Complex Software Systems

Xudong He
Florida International University, USA

Huiqun Yu
East China University of Science and Technology, China

Yi Deng
Florida International University, USA

ABSTRACT

Software has been a major enabling technology for advancing modern society, and is now an indispensable part of daily life. Because of the increased complexity of these software systems, and their critical societal role, more effective software development and analysis technologies are needed. How to develop and ensure the dependability of these complex software systems is a grand challenge. It is well known that a highly dependable complex software system cannot be developed without a rigorous development process and a precise specification and design documentation. Formal methods are one of the most promising technologies for precisely specifying, modeling, and analyzing complex software systems. Although past research experience and practice in computer science have convincingly shown that it is not possible to formally verify program behavior and properties at the program source code level due to its extreme huge size and complexity, recently advances in applying formal methods during software specification and design, especially at software architecture level, have demonstrated significant benefits of using formal methods. In this chapter, we will review several well-known formal methods for software system specification and analysis. We will present recent advances of using these formal methods for specifying, modeling, and analyzing software architectural design.

INTRODUCTION

It is wildly agreed that the main obstacle to "help computers help us more" and relegate to these helpful partners even more complex and sensitive tasks is not inadequate speed and unsatisfactory raw computing power in the existing machines, but our limited ability to design and implement complex systems with sufficiently high degree of confidence in their correctness under all circumstances (Clarke, Grumberg, & Peled, 1999). This problem of design validation—ensuring the correctness of the design at the earliest stage possible—is the major challenge in any responsible system development process, and the activities intended for its solution occupy an ever increasing portion of the development cycle cost and time budgets.

Two major approaches to analyze the system quality are *testing* and *verification*. Traditional and widely used quality assurance techniques based on software testing are inadequate to ensure the reliability of complex systems. In addition to the inherent limitation of testing from being able to guarantee system properties, many of today's software systems are designed to adapt in a wide range of environments and evolve over time. Because of this, the range of possible testing scenarios at code level becomes extremely large and potentially uncontrollable.

Formal methods (Harel, 1987; Hoare, 1985; Manna & Pnueli, 1992; Milner, 1989; Murata, 1989) for software specification and *verification* have been viewed as a promising way to address the problems associated with testing. These methods are precise and rigorous and can prevent and detect system defects introduced at the early stages of development, which are often more costly to fix and have more severe consequences. Despite tremendous advances (Clarke & Wing, 1996), however, widely spread application of *formal methods* in practical system development still remains to be seen (Craigen, Gerhart, & Ralston, 1995). A major cause for the problem is that results on *formal methods* are to large extent fragmented. Formal techniques are viewed as difficult and expensive to use because their application is ad hoc, and they are too fine grained to deal with the complexity in practical-sized development. Thus it is necessary to precisely define, measure, and analyze software dependability at a level higher than source code. Recent research (Knight, 2002) has shown that it is especially important to explore technologies how to handle dependability attributes at the *software architecture* level for the following reasons:

- A *software architecture* description presents the highest-level design abstraction of a system (Shaw & Garlan, 1996). As a result, it is relative simple compared to a detailed system design. Thus it is more likely to develop an effective methodology to study dependability attributes.
- As the highest-level design abstraction, a *software architecture* description precedes and logically and structurally influences other system development products. Thus an error in a *software architecture* has a much larger impact than an error introduced at a later development stage. Prevention and detection of errors at software architectural level are thus extremely important. Hence, it is necessary to study and measure dependability attributes before the actual software systems are developed and deployed.

Many studies, especially those done at the Software Engineering Institute at Carnegie Mellon University (Kazman, Klein, & Clements, 2000), have shown that a *software architecture* reveals, influences, or even dictates many system dependability features such as reliability, performance, security, and faulty-tolerance. Therefore, the dependability attributes measured at *software architecture* level can serve as the basis to predict and validate the dependability attributes of the developed and deployed systems.

In this chapter, we will review several well-known *formal methods* for complex software