Chapter 19
Quantitative Reasoning About Dependability in Event-B: Probabilistic Model Checking Approach

Anton Tarasyuk
Åbo Akademi University, Finland & Turku Centre for Computer Science, Finland

Elena Troubitsyna
Åbo Akademi University, Finland

Linas Laibinis
Åbo Akademi University, Finland

ABSTRACT
Formal refinement-based approaches have proved their worth in verifying system correctness. Often, besides ensuring functional correctness, we also need to quantitatively demonstrate that the desired level of dependability is achieved. However, the existing refinement-based frameworks do not provide sufficient support for quantitative reasoning. In this chapter, we show how to use probabilistic model checking to verify probabilistic refinement of Event-B models. Such integration allows us to combine logical reasoning about functional correctness with probabilistic reasoning about reliability.

INTRODUCTION
Formal approaches provide us with rigorous methods for establishing correctness of complex systems. The advances in expressiveness, usability and automation offered by these approaches enable their use in the design of wide range of complex dependable systems. For instance, Event-B (Abrial, 1996; Abrial, 2010) provides us with a powerful framework for developing systems correct-by-construction. The top-down development paradigm based on stepwise refinement adopted by Event-B has proved its worth in several industrial projects (Craigen, Gerhart, & Ralson, 1994; RODIN: IST FP6 project, 2004).

While developing system by refinement, we start from an abstract system specification and, in a number of correctness-preserving refinement steps, implement the system’s functional requirements. In our recent work (Tarasyuk, Troubitsyna,
& Laibinis, 2010) we have extended the Event-B modelling language with probabilistic assignment. Moreover, we have strengthened the notion of Event-B refinement by additionally requiring that the refined model would be more reliable. However, while the Event-B framework provides us with a powerful development platform (RODIN Platform), quantitative assessment of non-functional system requirements is sorely lacking. In this chapter we demonstrate how to overcome this problem. Specifically, we show how the Event-B development process can be complemented by probabilistic model checking to ensure the correctness of probabilistic refinement. We exemplify our approach by refinement and reliability evaluation of a simple monitoring system.

The remainder of the chapter is structured as follows. We start with a short introduction into our modelling formalism – the Event-B framework. We continue by briefly overviewing our approach to probabilistic modelling in Event-B. Next, we explain how probabilistic verification of Event-B models can be done using the PRISM symbolic model checker and also summarise our approach proposing a number of modelling guidelines. In the last two sections we exemplify our approach by presenting a case study and give concluding remarks, respectively.

INTRODUCTION TO EVENT-B

The B method (Abrial, 1996) is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications. Event-B is a formal framework derived from the B Method to model parallel, distributed and reactive systems. The Rodin platform provides automated tool support for modelling and verification (by theorem proving) in Event-B. Currently Event-B is used in the EU project Deploy (DEPLOY: IST FP7 project, 2008) to model several industrial systems from automotive, railway, space and business domains.

In Event-B a system specification is defined using an abstract (state) machine notion. An abstract machine encapsulates the state (the variables) of a model and defines operations on its state. The general form of an Event-B machine is shown on Figure 1.

The machine is uniquely identified by its name $M$. The state variables, $v$, are declared in the Variables clause and initialised in the $init$ event. The variables are strongly typed by the constraining predicates $I$ given in the Invariants clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the Events clause. Generally, an event can be defined as follows:

$$\text{evt} \triangleq \text{when } g \text{ then } S \text{ end},$$

where the guard $g$ is a conjunction of predicates over the state variables $v$ and the action $S$ is an assignment to the state variables. In its general form, an event can also have local variables as well as parameters. The guard defines the conditions under which the action can be executed, i.e., when the event is enabled. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.