1

Introduction

Nowadays, electronic devices largely depend on complex hardware and software systems. Among them, medical instruments, traffic control units, and many more safety-critical systems are subject to particular quality standards. They all come along with the absolute need for reliability, as, in each case, the consequences of a breakdown may be incalculable.

1.1 Formal Methods

Many existing systems were unthinkable some years ago and their complexity is still rapidly growing so that it becomes more and more difficult to detect errors or to predict their incidence. Consequently, *formal methods* play an increasing role during the whole system-design process. The term "formal methods" hereby covers a wide range of mathematically derived and ideally mechanized approaches to system design and validation. More precisely, research on formal methods attempts to develop mathematical models and algorithms that may contribute to the tasks of *modeling*, *specifying*, and *verifying* software and hardware systems. Let us go into these subareas in more detail:

Modeling

To make a system (or the idea of a system) accessible to formal methods, we require it to be modeled mathematically. Unfortunately, we are faced with a dilemma: on the one hand, a model ideally preserves and reflects as many properties of the underlying system as possible. On the other hand, it should be compact enough to support algorithms for further system analysis. However, in general, a good balance between detailed modeling and abstraction will pay off. But not only does the modeling process lead to further interesting conclusions, it may also help, itself, to get a better understanding of the system at hand. Thus, the purposes of modeling a system are twofold. One is to understand and document its essential features. The other is to provide the

formal basis for a mathematical analysis. Both are closely related and usually accompany each other.

Preferably, the modeling takes place in an early stage of system design. The starting point, at a high level of abstraction, may be a rough, even if precisely defined, idea of the system to be, which is subsequently refined stepwise towards a full implementation. While, as mentioned, the latter might be too detailed to draw conclusions from, previous stages of the design phase can be consulted for that purpose. The models considered in this book are *communicating automata*, which, though they might abstract from many details, reflect the operational behavior of a distributed system in a suitable manner to make it accessible to formal methods.

Specification

Correctness of a system is always relative to a *specification*, a property or requirement that must be satisfied. Embedded into the formal-methods framework, a specification is often expressed within a logical calculus whose formulas can be interpreted over system models, provided they are based on a common semantic domain. Prominent examples are monadic second-order (MSO) logic [8, 44], the temporal logics LTL [83] and CTL [22], and the μ-calculus [54]. A specification might also be given in a high-level language that is closer to an implementation and often allows us to *synthesize* a system directly and automatically. In this regard, let us mention some process-algebra based languages such as CCS [71], ACP [9], and LOTOS [18] and other formal design notions like VHDL [81]. In this book, we focus on a monadic second-order logic, which might be used to formulate properties that a *given* system should satisfy, and *high-level message sequence charts*, which are employed at a rather early stage of system development.

Verification

Once a system is modeled and a specification is given, the next task might be to check if the specification is satisfied by the model. If the system or, rather, the model of a system passes successfully through a corresponding validation process, it may be called correct in a mathematical sense. Preferably, the verification process runs automatically. However, many frameworks are too complex to support fully mechanized algorithms. In this respect, we can distinguish two general approaches to verification: *model checking* [23], which is fully automatic, and *theorem proving* [85], which requires human assistance. If, otherwise, a system is synthesized directly from its specification, then it can be assumed to be correct a priori, provided the translation preserves the semantics of the specification.

Several phases of system design are depicted in Fig. 1.1, which, in addition, features the stage of *code generation* to gain from the system model an effective implementation thereof.