1

Introduction

1.1 What Is a Cyber-Physical System?

The original computer was a stand-alone device focused on number crunching and information processing. While we continue to use computers for these tasks today, the more ubiquitous use of computers is within *embedded systems*. An embedded system consists of hardware and software integrated within a mechanical or an electrical system designed for a specific purpose. From watches to cameras to refrigerators, almost every engineered product today is an embedded system with an integrated microcontroller and software. The concept of a cyber-physical system is a generalization of embedded systems. A *cyber-physical system* consists of a collection of computing devices communicating with one another and interacting with the physical world via sensors and actuators in a feedback loop. Increasingly, such systems are everywhere, from smart buildings to medical devices to automobiles.

As an illustrative example of a cyber-physical system, consider a team of autonomous mobile robots tasked with the identification and retrieval of a target inside a house with an unknown floor plan. To achieve this task, each robot must be equipped with multiple sensors that collect the relevant information about the physical world. Examples of on-board sensors include a GPS receiver to track a robot's location, a camera to take snapshots of its surrounds, and an infrared thermal sensor to detect the presence of humans. A key computational problem then is to construct a global map of the house based on all the data collected, and this requires the robots to exchange information using wireless links in a coordinated fashion. The current knowledge of the positions of the robots, obstacles, and target then can be used to determine a motion plan for each of the robots. Such a motion plan includes high-level commands for each of the robots of the form "move in the north-west direction at a constant speed of 5 mph." This directive then needs to be translated to low-level control inputs for the motors controlling the robot's motion. The design objectives include safe operation (for instance, a robot should not bump into obstacles or other

robots), mission completion (for example, the target should be retrieved), and physical stability (for example, each robot should be stable as a dynamical system). Construction of the multi-robot system to meet these objectives requires design of strategies for *control*, *computing*, and *communication* in a synergistic manner.

Although certain forms of cyber-physical systems have been in industrial use since the 1980s, only recently has the technology for processors, wireless communication, and sensors matured to allow the production of components with impressive capabilities at a low cost. Realizing the full potential of these emerging computing platforms requires advances in tools and methodology for constructing reliable cyber-physical systems. This challenge of developing a systematic approach to integrated design of control, computation, and communication proved to be the catalyst for the formation of a distinct academic discipline of *cyber-physical systems* during the 2000s. The science of design for cyber-physical systems has been identified as a key research priority by government agencies as well as industries in automotive, avionics, manufacturing, and medical devices.

1.2 Key Features of Cyber-Physical Systems

Theory, methodology, and tools for assisting developers to build hardware and software systems in a systematic manner have been the central themes in computer science since its inception. The classical theory of computation, with its focus on computational complexity, and the methodology based on structured programming have both been instrumental in our ability to build today's complex software infrastructure. These principles for design of traditional software systems, however, are of no direct help in building cyber-physical systems due to significant differences in design concerns. The distinguishing characteristics of cyber-physical systems are discussed below.

Reactive Computation

In the classical model of computation, a computing device produces an output when supplied with an input. An example of such a computation is a program that, given an input list of numbers, outputs a sorted version of the input list. The notion of correctness for such a program can be captured mathematically as a *function* from input values to output values. Theory of computability and complexity gives us an understanding of which functions are computable and which functions are computable efficiently. The traditional programming abstractions of functions and procedures allow us to write programs for computing complex functions by composing simpler functions.

A *reactive* system, in contrast, interacts with its environment in an ongoing manner via inputs and outputs. As a typical example of reactive computation, consider a program for a cruise controller in a car. Such a program receives high-level input commands for turning the cruise controller on and off and for

changing the desired cruising speed. The control program needs to respond to such inputs by changing its output, which corresponds to the force that is applied to the engine throttle. The behavior of such a system then is naturally described by a *sequence* of observed inputs and outputs, and the notion of correctness specifies which input/output sequences correspond to acceptable behaviors. Cyber-physical systems are reactive systems, and thus the focus of this book is on reactive computation.

Concurrency

In a traditional *sequential* model of computation, the computation consists of a sequence of instructions executed one at a time. In *concurrent* computation, multiples threads of computation, usually called components or processes, are executing concurrently, exchanging information with one another to achieve the desired goal of the computation. Concurrency is fundamental to cyber-physical systems. In our example of a team of autonomous mobile robots, the robots themselves are separate entities and are thus executing concurrently. Each robot has multiple sensors and processors, and computing tasks such as constructing a map of the environment based on vision data and motion planning based on the map of the environment can be executing on separate processors in parallel. The motion planning task can be subdivided into logically concurrent subtasks such as local planning to avoid obstacles and global planning for optimal progress toward the target.

Understanding models and design principles for distributed and concurrent computation thus is critical for cyber-physical systems. For sequential computation, the model of *Turing machines* is accepted as the canonical model of computation. No such agreement exists for concurrent computation with a rich variety of proposals of formal models. Broadly speaking, these models fall into two categories: in *synchronous* models, components execute in lock-step, and the computation progresses in a logical sequence of synchronized rounds; and in *asynchronous* models, components execute at independent speeds, exchanging information by sending and receiving messages. Both types of models are useful for the design of cyber-physical systems. In our example, the system of robots can be viewed as an asynchronous system consisting of individual robots exchanging messages, whereas for simplicity of design, the computation on a single robot can be divided into concurrent activities executing in a logically synchronous manner.

Feedback Control of the Physical World

A *control system* interacts with the physical world in a feedback loop by measuring the environment via sensors and influencing it via actuators. For example, a cruise controller is constantly monitoring the speed of the car and adjusts the throttle force so that the speed stays close to the desired cruising speed. Controllers are components of a cyber-physical system, and this integration of

computing devices with the physical world sets cyber-physical systems apart from the traditional computers.

Design of controllers for the physical world requires modeling the dynamics of the physical quantities: to adjust the throttle force, a cruise controller needs a model of how the speed of the car changes with time as a function of the throttle force. The theory of dynamical control systems is a well-developed discipline with a rich set of mathematical tools for design and analysis, and a basic understanding of these principles is valuable to designers of cyber-physical systems. The traditional control theory focuses on *continuous-time* systems. In a cyber-physical system, the controller consists of discrete software comprising concurrent components operating in multiple possible modes of operation, interacting with the continuously evolving physical environment. Such systems with a mix of discrete and continuous dynamics are sometimes called *hybrid systems*, and the emerging principles of design and analysis of controllers for such systems will be studied in this book.

Real-Time Computation

Programming languages, and the supporting infrastructure of operating systems and processor architectures, typically do not support an explicit notion of real time. This offers a convenient abstraction for traditional computing applications such as document processing, but real-time performance is critical for cyber-physical systems. For example, for a cruise controller to satisfactorily control the speed of the car, its design should take into account the time it takes its subcomponents to execute the necessary computations and communicate the results.

Modeling timing delays, understanding their impact on the correctness requirements and system performance, timing-dependent coordination protocols, and resource-allocation strategies to ensure predictability have been the subject of study in the sub-discipline of *real-time systems*. A principled approach to design and implementation of cyber-physical systems thus builds on these techniques.

Safety-Critical Applications

While designing and implementing a cruise controller, we expect a high level of assurance in the correct operation of the system because errors can lead to unacceptable consequences such as loss of life. Applications where the *safety* of the system has a higher priority over other design objectives such as performance and development cost are called *safety-critical*. Computing devices now control aircrafts, automobiles, and medical devices and thus are all examples of cyber-physical systems for safety-critical applications. In this context, establishing that the system works correctly at design time is of paramount importance and sometimes mandatory due to government regulations for certification of systems.

The traditional route to system development is design and implementation, followed by extensive testing and validation to detect bugs. The more principled

approach to system development involves writing mathematically precise requirements of the desired system, designing models of system components along with the environment in which the system is supposed to operate, and using analysis tools to check that the system model meets the requirements. Compared to the traditional approach, this methodology can detect design errors in early stages and ensure higher reliability. Such an approach based on formal models and verification is appealing in safety-critical applications, is being increasingly adopted by industry, and will be a central theme in this book.

1.3 Overview of Topics

The goal of this textbook is to provide an introduction to the principles of design, specification, modeling, and analysis of cyber-physical systems. Due to the distinguishing characteristics of cyber-physical systems, these principles are drawn from a diverse set of sub-disciplines including model-based design, concurrency theory, distributed algorithms, formal methods for specification and verification, control theory, real-time systems, and hybrid systems. Research conferences and textbooks are devoted to each of these sub-disciplines, and this book is aimed at explaining the core ideas relevant to system design and analysis in each of these in a coherent manner. The topics are discussed by interweaving the three themes of *formal models*, *model-based design*, and *specification and analysis* as discussed below.

Formal Models

The goal of modeling in system design is to provide mathematical abstractions to manage the complexity of design. In the context of reactive systems, the basic unit of modeling is a component that interacts with its environment via inputs and outputs. Different forms of interaction lead to different classes of models. We begin in chapter 2 by focusing on *synchronous* modeling, where all components execute in lock-step in a sequence of rounds. Then in chapter 4, we switch to *asynchronous* models, where different activities execute at independent speeds. In chapter 6, we study *continuous-time* models of dynamical systems that are suitable for capturing the evolution of the physical world. Chapter 7 introduces *timed* models, where the interaction among components is facilitated by the knowledge of concrete bounds on timing delays. Finally, chapter 9 considers *hybrid* systems by integrating models of discrete interaction and dynamical systems.

To describe models, we use a combination of block diagrams, code fragments, state machines, and differential equations. We define our models *formally*, that is, in a mathematically precise manner. The formal semantics allows us to answer questions such as, "what are the possible behaviors of a component" and "what does it mean to compose two components" rigorously. Examples of modeling concepts covered in this book include nondeterministic behavior,

input-output interfaces of components, time-triggered and event-triggered communication, await dependencies for synchronous composition, communication using shared memory, atomicity of synchronization primitives, fairness for asynchronous systems, equilibria for dynamical systems, and Zeno behaviors for timed and hybrid systems.

Specification and Analysis

To check that the design (or system implementation) works correctly as intended, the designer first needs to express the requirements capturing correctness in a mathematically precise manner. Analysis tools then allow the designer to check that the system satisfies its requirements. This textbook covers a range of specification formalisms and associated techniques for formal verification.

Chapter 3 introduces *safety* requirements. A safety requirement asserts that "nothing bad ever happens" and can be formalized using invariants and monitors. We first consider the general technique of *inductive invariants* for proving that a system satisfies its safety specification and then *state-space exploration* algorithms for automatically establishing safety properties. Both *enumerative* and *symbolic* search algorithms are developed, including the symbolic exploration using the data structure of Ordered Binary Decision Diagrams (BDDs) commonly used in hardware verification. The presence of continuous-time dynamics of the physical quantities poses new challenges for safety verification of cyber-physical systems. For verifying systems with hybrid dynamics, we study the proof method based on *barrier certificates* and symbolic search algorithms for two special classes, namely, *timed automata* and *linear hybrid automata*.

Chapter 5 introduces *liveness* requirements: such a specification asserts that "something good eventually happens." We introduce the temporal logic Linear Temporal Logic (LTL) to formally express such correctness requirements and show how the notion of monitors can be generalized to Büchi automata so as to capture LTL requirements. The problem of automatically verifying LTL requirements of system models is known as *model checking*. Both enumerative and symbolic state-space exploration techniques are generalized to solve the model-checking problem, and the method of *ranking functions* is developed as a general proof principle for proving liveness requirements.

For dynamical systems, a basic design requirement concerns *stability*, which informally means that small perturbations to system inputs should not cause a disproportionate change in its observed behavior. This classical topic from control theory is studied in chapter 6, with a particular focus on *linear systems* for which tools from linear algebra are shown to be useful for mathematically establishing stability.

While implementing embedded systems, a key analysis problem is to establish that the time it takes to execute different tasks in the system model on a given computational platform is consistent with the model-level assumptions regarding the timing delays. *Real-time scheduling* theory is aimed at formalizing and

solving this problem and is the subject of chapter 8. We focus primarily on understanding two fundamental scheduling algorithms: Earliest Deadline First (EDF) and Rate Monotonic.

Model-Based Design and Case Studies

Principles of modeling, specification, and analysis are illustrated by constructing solutions to representative design problems from distributed algorithms, network protocols, control design, and robotics. We illustrate how modeling differs from programming, for instance, by allowing the specification of *nondeterministic* behavior and including explicit models of how the *environment* behaves. While designing model-based solutions, we emphasize two principles:

1. *Structured* design: simple components can be composed to perform more complex tasks, and, conversely, a design problem can be decomposed into simpler subtasks.

2. *Requirements-based* design: correctness requirements are specified precisely up front and are used to guide the exploration among design alternatives and for debugging of the design in early stages.

We study the classical distributed coordination problems of *mutual exclusion*, *consensus*, and *leader election*. These problems are revisited throughout the textbook, highlighting how the power of synchronization primitives influences the design. Another set of problems focuses on message communication, including how to reliably transmit messages in the presence of a lossy network and how to synchronize a sender and a receiver in the presence of timing uncertainties introduced by imperfect clocks. The design of a cruise controller illustrates how to integrate synchronous design using block diagrams with the design of a low-level PID controller. We conclude with case studies representative of cyber-physical systems: design of a pacemaker monitoring and responding to timing patterns of heart pulsations, obstacle avoidance for a team of robots using coordination, and design of stabilizing controllers communicating over a multi-hop network.

1.4 Guide to Course Organization

This textbook is suitable for a semester-long course aimed at upper level undergraduate or first-year graduate students in computer science, computer engineering, or electrical engineering. This section gives some suggestions regarding the organization of such a course.

Prerequisites

The textbook emphasizes principles of modeling, design, specification, and analysis. These principles are drawn from a range of mathematical topics such as calculus, discrete mathematics, linear algebra, and logic. Most of the concepts

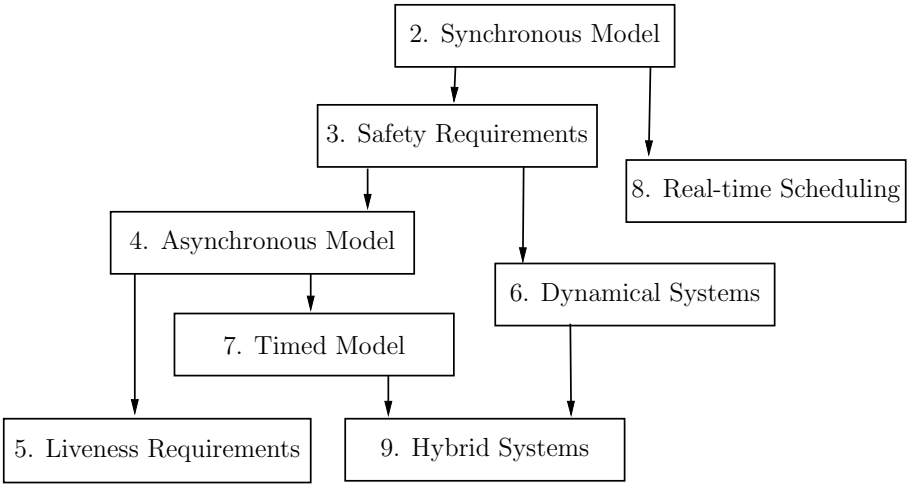

Figure 1.1: Dependencies among Chapters

are developed from the first principles, and thus courses in these topics are not prerequisites. However, some basic level of mathematical maturity is essential for understanding the course material. The course is suitable for students who have completed the required theory courses either in the computer science curriculum (such as *Discrete Mathematics* and *Theory of Computation*) or in the electrical engineering curriculum (such as *Signals and Systems* and *Dynamical Systems*).

Throughout the textbook, we discuss design problems from a range of applications such as control systems, distributed coordination, network protocols, and robotics. Analogous to the necessary mathematical principles, in each of our case studies, the basic constraints of the application domain are explained without assuming any explicit background knowledge. However, prior experience with design and implementation of software and systems is necessary to appreciate these examples. This experience can be gained by completing project-oriented undergraduate courses: for example, the courses on programming and operating systems in the computer science curriculum and those on mechanical or control systems in the engineering curriculum.

Selection of Topics

Not all topics from the textbook can be covered during a semester-long course. Figure 1.1 shows dependencies among chapters that can be used to guide the selections of topics. In each chapter, the essential concepts can be covered even if one skips the sections marked with an asterisk. We describe three possible courses below.

A fast-paced course that aims to cover all the themes, namely, modeling, design, specification, and verification, is feasible and has been taught at University of Pennsylvania for many years. For such a course, we recommend that the following sections be skipped: 3.3, 5.3, 6.4, 7.2, and 9.3.

A course primarily focused on modeling and design can omit techniques for analysis and verification. In particular, sections 3.3, 3.4, 5.2, 5.3, 6.4, 7.3, and 9.3, can be skipped. However, it is recommended that even such a course should emphasize the role of formally specified requirements in principled design and, thus, should include the specification formalisms.

A third possibility to narrow the scope of the course is by omitting chapters 6, 8, and 9. Such a course focuses on modeling, design, specification, and verification of reactive systems but does not include modeling of the interaction with the physical world.

Homework and Projects

Each section has a number of exercises at its end, and students are expected to answer these questions with mathematical rigor. The more challenging exercises are marked with an asterisk.

Besides solving theoretical exercises, coursework should include design projects using software for modeling and analysis. The textbook discusses modeling concepts in a generic manner and is not tied to the concrete syntax of any specific tool. The following are some examples of design projects:

1. Synchronous modeling and symbolic safety verification (chapters 2 and 3): A project focused on synchronous hardware designs, such as arbiters and on-chip communication protocols, offers an opportunity to understand how to structure a design as a hierarchical composition of subcomponents. Industry-standard hardware description languages such as VHDL and Verilog (see `vhdl.org`) can be used for such a modeling project. Alternatively, the academic tool NuSmv (see `nusmv.fbk.eu`) can be used for modeling and verification of requirements using BDD-based symbolic state-space exploration.

2. Asynchronous modeling and model checking (chapters 4 and 5): A distributed protocol, such as a cache coherence protocol used for coordinating accesses to global shared memory on modern multiprocessor systems, can be modeled as communicating asynchronous processes in the modeling tool Spin (see `spinroot.com`). The tool allows the user to specify both safety and liveness requirements in temporal logic and to debug the correctness of the protocol using model checking.

3. Control design for dynamical systems (chapter 6): A traditional project in a course on control systems involves building a model of a physical system, designing a controller for it, and then establishing the stability

of the composed system by using tools of linear algebra. Such a project is quite suitable for this course also. The most commonly used software for such a project is MATLAB (produced by Mathworks, `mathworks.com`), and a typical problem is to design a controller to maintain a pendulum attached to a moving cart in a vertically inverted position.

4. Modeling and verification of timed systems (chapter 7): The modeling tool UPPAAL (see `uppaal.org`) supports modeling using interacting timed automata and verification of safety properties using symbolic state-space exploration. A suitable case study in this domain consists of requirements-based design and analysis of a control algorithm for a medical device such as an autonomous infusion pump or a detailed design of a pacemaker.

5. Modeling and simulation of hybrid systems (chapter 9): The modeling tools such as STATEFLOW and SIMULINK (see `mathworks.com`), MODELICA (see `modelica.org`), and PTOLEMY (see `ptolemy.org`) allow structured modeling of hybrid systems, and multi-robot coordination offers a fertile problem domain to set up projects in design and analysis of cyber-physical systems using these modeling tools. The goal of analysis in such a project is to understand the trade-offs among different design parameters using numerical simulation.

Supplementary Reading

The case for a new *science* of design for embedded and cyber-physical software systems, with an emphasis on high assurance, has been made by many researchers over the last decade (see [Lee00, SLMR05, KSLB03, HS06, SV07]). Now there is a vibrant academic research community in this sub-discipline, and the annual conferences *Embedded Systems Week* (see `esweek.org`) and *Cyber-Physical Systems Week* (see `cpsweek.org`) reflect the current trends in research in cyber-physical systems.

The textbook *Introduction to Embedded Systems* [LS11] is the closest to ours in terms of focus and selection of topics and, thus, is a valuable supplementary textbook. For comparisons, [LS11] covers a broader range of topics, for instance, it discusses processor architectures for embedded applications, whereas this textbook includes a more in-depth development of analysis and verification techniques as well as case studies.

Specialized books on each of the topics covered in this textbook are also useful for a deeper study of that topic. For principles of model-based design, [Hal93] is focused on synchronous models, [Mar03] is devoted to design of embedded systems, and [Pto14] highlights design by integrating heterogeneous modeling styles. Among the rich literature on distributed systems, [Lyn96] and [CM88] introduce a wide range of distributed algorithms emphasizing formal modeling, correctness requirements, and verification. For an introduction to formal logic and its applications to software verification, we recommend [HR04] and

[BM07]. Textbooks devoted to automated verification and model checking include [CGP00] and [BK08], and [Lam02] explains how logic can be used for specification and development of reactive systems. Dynamical systems, with a focus on design of controllers for linear systems, is a classical topic with many textbooks such as [AM06] and [FPE02]. For an introduction to real-time systems, with an emphasis on schedulability, see [But97] and [Liu00]. Finally, the research monographs [Tab09], [Pla10], and [LA14] focus on formal modeling, control, and verification of hybrid systems.