1

Introduction

Time is an illusion, lunchtime doubly so.
– Douglas Adams

Time plays a central role in our lives. In describing the world, or our activities within it, we naturally invoke temporal descriptions. Some of these are explicit, such as 'next week' or 'in 5 minutes', while others implicitly acknowledge the passing of time, for example 'during', 'did' or 'will do'. Not surprisingly, it is also important to be able to describe temporal aspects within the world of Computer Science: computations naturally proceed through time, and so have a *history* of activity; computational processes take time to act; some processes must finish before others can start; and so on. Consequently, being able to understand, and reason about, temporal concepts is central to Computer Science.

In this book, we will explain how some of these temporal notions can be described and manipulated. This, in turn, will allow us to carry out a temporal analysis of certain aspects of computation. To be precise in our temporal descriptions, we will use *formal logic*. These not only provide a concise and unambiguous basis for our descriptions, but are supported by many well-developed tools, techniques and results that we can take advantage of.

This book will provide an introduction to work concerned with formal logic for capturing temporal notions, called *temporal logic*, together with some of its applications in the formal development and analysis of computational systems. The name 'temporal logic' may sound complex and daunting. Indeed, the subject can sometimes be difficult because it essentially aims to capture the notion of time in a logical framework. However, while describing potentially complex scenarios, temporal logic is often based on a few simple, and fundamental, concepts. We aim to highlight these in this book.

An Introduction to Practical Formal Methods Using Temporal Logic, First Edition. Michael Fisher.

As we might expect, this combination of expressive power and conceptual simplicity has led to the use of temporal logic in a range of subjects concerned with computation: Computer Science, Electronic Engineering, Information Systems and Artificial Intelligence. This representation of dynamic activity via temporal formalisms is used in a wide variety of areas within these broad fields, for example Robotics [176, 452], Control Systems [317, 466], Dynamic Databases [62, 110, 467], Program Specification [339, 363], System Verification [34, 122, 285], and Agent-Based Systems [207, 429]. Yet that is not all. Temporal logic also has an important role to play in Philosophy, Linguistics and Mathematics [222, 470], and is beginning to be used in areas as diverse as the Social Sciences and Systems Biology.

But why is temporal logic so useful? And is it really so simple? And how can we use practical tools based on temporal logic? This book aims to (at least begin to) answer these questions.

1.1 Aims of the book

Our aims here are to

- provide the reader with some of the background to the development and use of temporal logic;
- introduce the foundations (both informal and formal) of a simple temporal logic; and
- describe techniques and tools based on temporal logic and apply them to sample applications.

This book is *not* deeply technical. It simply aims to provide sufficient introduction to a number of areas surrounding temporal logic to enable either further, in-depth, study or the use of some of the tools described. Consequently, we would expect the readership to consist of those studying Computer Science, Information Systems or Artificial Intelligence at either undergraduate or postgraduate level, or software professionals who wish to expand their knowledge in this area. Since this is an *introductory* text, we aim to provide references to additional papers, books and online resources that can be used for further, and deeper, study. There are also several excellent, more advanced, textbooks and monographs that provide much greater technical detail concerning some of the aspects we cover, notably [34, 50, 122, 224, 299, 327, 339, 363, 364].

While there are very few proofs in this book, some of the elements are quite complex. In order to support the reader in understanding these aspects, we have often provided both exercises and pointers to further study in each chapter. We have interspersed exercises throughout the text, and sometimes provide a further selection of exercises at the end of each chapter, with answers in Appendix B. In addition, further resources can be found on the Web pages associated with this book:

`http://www.csc.liv.ac.uk/~michael/TLBook`

This URL provides links not only to additional material related to the book, but also contains pointers to a range of systems that are, at least in part, based on temporal logic.

1.2 Why temporal logic?

As computational systems become more complex, it is often important to be able to describe, clearly and unambiguously, their behaviour. Formal languages with well-defined semantics are increasingly used for this purpose, with *formal logic* being particularly prominent. This logic not only presents a precise language in which computational properties can be described, but also provides well-developed logical machinery for manipulating and analysing such descriptions.

For example, it is increasingly important to *verify* that a computational system behaves as required. These requirements can be captured as a formal specification in an appropriately chosen formal logic, with this specification then providing the basis for *formal verification*. While a system can be *tested* on many different inputs, formal verification provides a comprehensive approach to potentially establishing the correctness of the system in *all* possible situations. Verification within formal logic is aided by a logic's machinery, such as proof rules, normal form and decision procedures. Alternatively, we may wish to use the logical specification of a system in other ways, such as treating it as a *program* and directly executing it. Again, the well-developed logical machinery helps us with this.

Though logical specifications are clearly an important area to develop, the increased complexity of contemporary computational systems has meant that specifications in terms of traditional logic can become inappropriate and cumbersome. Consequently, much of the recent work concerning the use of formal logic in Computer Science has concentrated on developing logic that provides an appropriate level of abstraction for representing complex dynamic properties. It is precisely for this reason that *temporal logic* has been developed. Temporal logic has been used in Linguistics since the 1960s. In particular, temporal logic was originally used to represent tense in natural language [420]. However, in the late 1970s, temporal logic began to achieve a significant role in the formal specification and verification of concurrent and distributed systems [411, 412]. This logic is now at the heart of many specification, analysis and implementation approaches.

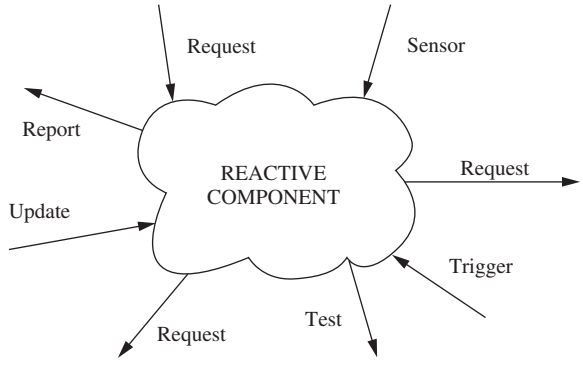
1.2.1 Motivation: evolution of computational systems

The way computational systems are designed and programmed has evolved considerably over the last 40 years. Correspondingly, the abstractions used to characterize such systems have changed during that time. When formal approaches to program development were initially envisaged, the key abstraction was that of a *transformational system* [260]. Transformational systems are essentially those whose behaviour can be described in terms of each component's input/output behaviour:

In other words, each component in a system receives some input, carries out some operation (typically on data structures), and terminates having produced some output. The

Formal Methods that have been developed for such systems describe the data structures and the behaviour of operations (via pre- and post-conditions) on these structures. Specification notations particularly relevant to this type of system were developed in the late 1960s and came to prominence in the 1970s. Typical examples include Floyd-Hoare Logics [214, 274, 418], weakest precondition semantics [146], VDM [304], Z [135], and (more recently) B [7, 340, 446], as well as the functional programming metaphor.

While the use of Formal Methods for transformational systems has been very effective in many areas, it became clear in the 1970s that an increasing number of systems could not easily be categorized as 'transformational'. Typically, this was because the components were either non-terminating, continuously reading input (not just at the beginning of computation), continuously producing output (not just at the end), or regularly interacting with other concurrent or distributed components. These have been termed *reactive systems* [260] and can be visualized in a more complex way, for example:



This diagram highlights the fact that multiple inputs can be received, and multiple outputs can be generated, by reactive systems. Such systems are typically interacting, evolving, non-terminating systems.

Formal Methods for *reactive systems* often require more sophisticated techniques than the pre- and post-conditions provided in notations such as VDM or Z. In particular, in the late 1970s, *temporal logic* was applied to the specification of reactive systems, with this approach coming to prominence in the 1980s [363, 414]. It is widely recognized that *reactive* systems [260], as described above, represent one of the most important classes of systems in Computer Science and, although the analysis of such systems is difficult, it has been successfully tackled using temporal representations [168, 411, 460], where a number of useful concepts, such as *safety*, *liveness* and *fairness* properties can be formally, and concisely, specified [363]. Such a logical representation of a system then permits the analysis of the system's properties via logical methods, such as *logical proof*. A specific proof method for deciding whether a temporal formula is true or false is one of the aspects that we will examine later in this book.

1.3 What is temporal logic?

Temporal logic is an extension of classical logic[1], specifically adding operators relating to time [168, 196, 210, 224, 225, 279, 433, 460, 478]. Modal logic [67, 68, 109, 226, 291]

[1] A very brief review of classical logic is provided in Appendix A.

provides some of the formal foundations of temporal logic, and many of the techniques used in temporal logic are derived from their modal counterparts. In addition to the operators of classical logic, temporal logic often contains operators such as '$\bigcirc$', meaning *in the next moment in time*, '$\Box$', meaning *at every future moment*, and '$\Diamond$', meaning *at some future moment*. These additional operators allow us to construct formulae such as

$$\Box(try\_to\_print \Rightarrow \Diamond\neg try\_to\_print)$$

to characterize the statement that

> "*whenever* we try to print a document then at *some* future moment we will *not* try to print it".

The flexibility of temporal logic allows us also to provide formulae such as

$$\Box(try\_to\_print \Rightarrow \bigcirc(printed \vee try\_to\_print))$$

meant to characterize

> "*whenever* we try to print a document *then*, at the *next* moment in time, *either* the document will be printed *or* we again try to print it"

and

$$\Box(printed \Rightarrow \bigcirc\Box\neg try\_to\_print)$$

meaning

> "*whenever* the document has been printed, the system will *never* try to print it (ever again)".

Given the above formulae then, if we try to print a document, i.e.

$$try\_to\_print$$

we *should* be able to show that, eventually, it will stop trying to print the document. Specifically, the statement

$$\Diamond\Box\neg try\_to\_print$$

can be inferred from the above formulae. We will see later how to establish automatically that this is, indeed, the case.

Although there are many different temporal logics [168, 196, 279], we will mainly concentrate on one very popular variety that is:

- *propositional*, with no explicit first-order quantification;
- *discrete*, with the underlying model of time being isomorphic to the Natural Numbers (i.e. an infinite, discrete sequence with distinguished initial point); and
- *linear*, with each moment in time having at most one successor.

Note that the infinite and linear constraints ensure that each moment in time has *exactly* one successor, hence the use of just one form of '$\bigcirc$' operator. If we allow several immediate successors, then we typically require other operators. (More details concerning such logics will be provided in Chapter 2.)

1.4 Structure of the book

The book comprises four parts, of very different sizes.

- In the first part, we introduce temporal logic (Chapter 2), and show how it can be used to specify a variety of computational systems (Chapter 3).
- In the second part, we then describe techniques that use these temporal specifications, namely deductive verification (proof; Chapter 4), algorithmic verification (model checking; Chapter 5), and model building (direct execution; Chapter 6).
- In the third part (Chapter 7), we provide an overview of some of the areas where temporal-based formal techniques are being used, not only in Computer Science, but also in Artificial Intelligence and Engineering.
- Finally, in Appendices A and B, we provide a review of classical logic and sample answers to exercises, respectively.

In the first part, we essentially introduce the basic concepts of temporal logic and temporal specification. Throughout the chapters in this first part we provide a range of examples and exercises to reinforce the topics. In the second part, comprising chapters describing verification and execution approaches, we split each chapter into:

- an *introductory* section conveying the motivation for, and principles of, the approach being tackled;
- details of a particular *technique* epitomizing the approach;
- an overview of a particular *system* implementing the technique described; and
- a selection of *advanced topics* concerning this area and an overview of *alternative* systems tackling this problem.

Again, within this structure, examples and (some) exercises will be provided.

To give an idea of the substructure of Chapters 4, 5 and 6, we can give the following broad distribution of topics.

- Chapter 4 (Deduction)

INTRODUCTORY:	the idea behind deductive verification and temporal proof.
TECHNIQUE:	clausal temporal resolution.
SYSTEM:	TSPASS.
ADVANCED:	including alternative approaches, such as tableaux and extensions.

- Chapter 5 (Model Checking)

INTRODUCTORY:	the idea behind algorithmic temporal verification and model checking.
TECHNIQUE:	the automata-theoretic view of model checking.
SYSTEM:	`Spin`.
ADVANCED:	including alternative approaches, such as extended, bounded, or abstracted model checking.

- Chapter 6 (Execution)

INTRODUCTORY:	model construction through direct execution.
TECHNIQUE:	METATEM execution algorithm.
SYSTEM:	`Concurrent MetateM`.
ADVANCED:	including alternative approaches, such as temporal logic programming and interval-based execution.

Finally, in Chapter 7, we provide an overview of selected applications where temporal-based formal methods have been, or are being, used. Some use the techniques described in the book, others use alternative temporal approaches, but all help to highlight the wide applicability of temporal methods.