# Practical TLA+

## Planning Driven Development

**Hillel Wayne**

Apress®

*Practical TLA+: Planning Driven Development*

Hillel Wayne
Chicago, Illinois, USA

ISBN-13 (pbk): 978-1-4842-3828-8 ISBN-13 (electronic): 978-1-4842-3829-5
https://doi.org/10.1007/978-1-4842-3829-5

Library of Congress Control Number: 2018958706



Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail editorial@apress.com; for reprint, paperback, or audio rights, please email bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484238288. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

*For my teachers, Todd Fadoir and Larry McEnerney.*

# Table of Contents

# About the Author

**Hillel Wayne** is a software consultant who specializes in formal methods and specification. He also writes on empirical engineering, software history, and systems thinking. In his free time, he juggles and makes chocolate. He lives in Chicago. You can find his other work at hillelwayne.com or on Twitter at @hillelogram.

# About the Technical Reviewer

**Jud White** is a back-end and distributed systems engineer with 18 years of professional experience. He uses TLA+ to simplify designs and ensure the behavior and trade-offs of systems are well understood and codified. He currently works at Dell in Austin, Texas, and occasionally does Go training. He lives with his girlfriend and two lovable pit bulls. Follow him on GitHub: @judwhite; Instagram: @jud.white; and Twitter: @judson_white.

# Acknowledgments

Richard Whaling, Andrew Helwer, Murat Demirbas, Lorin Hochstein, and Sidharth Masaldaan were all kind enough to provide feedback on early drafts of the chapters. Discussions with Leslie Lamport, Ron Pressler, and Markus Kuppe helped clarify and refine sections of this book.

Jud White went above and beyond with his technical review. He, more than anyone else, made this actually worth reading.

Finally, Mark Powers, Matt Moodie, Steve Anglin, and Sherly Nandha all did a fantastic job editing this book.

# Introduction

This is a book about specification.

Most software flaws come from one of two places. When the code doesn't match our expectations, it could be that the code is wrong. Most software correctness techniques – types, tests, etc. – are used to check the code. But it could instead be that the code is correct and our expectations are wrong: there's a fundamental error in our design.

These errors, called **specification errors**, are some of the most subtle and dangerous bugs. They can span multiple independent programs, occur in convoluted race conditions, or depend on physical phenomena. Our regular tools simply can't find them.

Instead, we can find them with a **specification language** such as TLA+. TLA+ is the invention of Leslie Lamport, winner of the 2013 Turing Award and the inventor of Paxos and LaTeX. Instead of writing your design in code or plain English, you write it in TLA+'s special notation. Once you specify your core assumptions and requirements, it can explore how that system would evolve over time, and whether it actually has the properties you want.

What makes TLA+ more suitable for this than, say, Python? Python is designed to be run, and it is limited to what a computer can do. TLA+, though, is designed to be explored. By leveraging simple math, it can express concepts much more elegantly and accurately than a programming language can. For example, given a set of numbers, here is how we would return the numbers in that set that are the sum of two other numbers in it:

```
EXTENDS Integers

FilterSums(set) ==
  { x \in set: \E y, z \in set \ {x}: y /= z /\ x = y + z }
```

Instead of being compiled or interpreted, TLA+ is *checked*. We use a **model checker**, called TLC, to execute every possible behavior of our specification. For example, if it sees the lines

```
either
  with change \in 1..10 do
    counter := counter + change;
  end with;
or
  counter := 0;
end either;
```

TLC will split the model into 11 separate timelines and check them all for any issues. If the spec has multiple simultaneous processes, TLC will explore every possible ordering of their steps. If the spec has 100 possible initial states, TLC will explore every behavior from every single one of them. With TLA+ we can check that global properties are preserved, that distributed systems are fault-tolerant, or even that every behavior of an algorithm eventually terminates with a correct answer. We can cut out bugs before we've written a single line of code.

## What This Book Will Teach You

There are two benefits to learning TLA+. The first is model checking. Once you have written a specification in TLA+, you can use the model checker to find any inconsistencies in your spec. TLA+ can find bugs that span multiple systems and several nested race conditions.

The second benefit of TLA+ is subtler. Specifying a system forces you to be precise in what you actually want. "Select the first element" is different from "select an arbitrary element" or "select any element" in ways that could lead to a spec being correct or not. By unambiguously writing your specification, you understand it better. Problems become obvious even without the model checker. The more you work with TLA+, the more you intuitively see the failure modes in systems.

For most people, the biggest challenge to learning TLA+ is the change of perspective you need. While programming is a necessary prerequisite to specification, it's a very different approach and the adjustment takes some getting used to. How do you specify an algorithm? A distributed system? How do you make the jump from knowing TLA+ in theory to using it in practice, finding actual bugs in actual production systems?

This book is aimed at addressing that. I've written over a dozen examples spread across a wide range of problem domains, from low-level threading to large-scale distributed systems. Shorter examples are part of larger chapters, while the longer ones

are chapters of their own. By showing you how a specification is defined and written, I hope to help you build an intuition for how to use TLA+ in practice. The examples also provide hands-on experimentation, and, if you decide to continue with TLA+, templates you can use to write your own specs.

## What This Book Won't Teach You

This book will not teach you programming. It will not teach you how to test code nor how to write mathematical proofs that your code is correct. Formally proving code correct is much more difficult and high effort than proving designs are correct. This book will not teach you how to directly convert TLA+ into production code. Much of TLA+'s flexibility and power comes from it *not* having to match a programming language. A few dozen lines of TLA+ can match hundreds or thousands of lines of code. No tool can replace your insight as an engineer.

Finally, this book is not a comprehensive resource on how to use TLA+. In particular, we focus on using **PlusCal**, the main algorithm language that compiles to TLA+. PlusCal adds additional constructs that make TLA+ easier to learn and use. While powerful and widely used in the TLA+ community, PlusCal nonetheless has a few limitations that raw TLA+ does not.

## Prerequisites

You should already be an experienced programmer. While TLA+ can be used with any programming language, it is not a programming language. Without having something to program with, there's really no reason to use TLA+. With this assumption, we can also move faster: we don't need to learn what a conditional is, just what it looks like in TLA+.

Knowing some logic and math is going to help. You don't have to be particularly experienced with it, but TLA+ borrows its syntax heavily from mathematics. If you know what `(P => Q) \/ R` means, you're fine. If you don't know, this should still be accessible, but Appendix A will also teach you everything else you need.

# How This Book Is Structured

This book consists of two parts. In Chapters 1–6, we cover the semantics of TLA+ and PlusCal. In Part 2, we cover the application of TLA+, showing you how to write effective operators and specifications. While the chapters in Part 1 are intended to be read in sequence, the chapters in Part 2 can be read in any order.

1. *Chapter 1* is a whirlwind tour of the language. We specify a bank transfer algorithm and show how it can overdraft or lose money if you start multiple simultaneous wires.
2. *Chapter 2* teaches the basics of the algorithm language so you can specify simple, nonconcurrent systems. We also introduce the data structures and nondeterministic language constructs, allowing us to model basic concurrency and state machines.
3. *Chapter 3* teaches operators and expressions, invariants, and functions. Combined with the PlusCal constructs, this allows us to specify complex systems and test algorithms, such as constraint optimization problems.
4. *Chapter 4* covers how to organize your modules into larger-scale specifications and test your specs on different state spaces and requirements.
5. *Chapter 5* shows how to model concurrent systems, race conditions, and deadlocks.
6. *Chapter 6* covers the last bits of TLA+ we need for this book: how to model the temporal properties of a system, such as resource guarantees, crashes, and requirements about what states the system will eventually reach.
7. *Chapter 7* teaches how to use TLA+ to verify abstract algorithms are correct, as well as prove simple runtime properties, like worst-case algorithmic complexity and avoiding integer overflow.
8. *Chapter 8* shows how to create reusable data structure libraries, such as linked lists, and how to use them as part of larger specs.

9. *Chapter 9* is about the state machine pattern, a common technique we use to turn high-level specifications into lower-level ones that more closely match our production code.
10. *Chapter 10* takes an informal business request and, in trying to specify it, shows how an ambiguous requirement can lead to very different specifications and runtime properties.
11. Finally, in *Chapter 11* we will specify the MapReduce algorithm and make it both correct and fault tolerant.
12. *Appendix A* is a crash course of mathematics, including simple set theory and logic that you will find useful when writing TLA+ specs.
13. *Appendix B* is a copy of the PT module in case the Internet burned down between you downloading the toolbox and downloading PT. See below for a description of PT.
14. *Appendix C* is the math underpinning TLA+, such as modal logic and actions. It is unnecessary to understand anything in the book but will be helpful if you want to understand the ideas behind TLA+.

# Initial Setup

## The Toolbox

To use TLA+, we need the PlusCal compiler, the syntactic checker, and the TLC model checker. Everybody uses the official IDE, called the *TLA+ Toolbox*. You can download the toolbox at `https://github.com/tlaplus/tlaplus/releases/`. As of the time of this writing the toolbox version is 1.5.7. You will also need to install Java. Once set up, create a new module under `File` ➤ `Open Spec` ➤ `Add New Spec`. You should see what is shown in Figure 1.

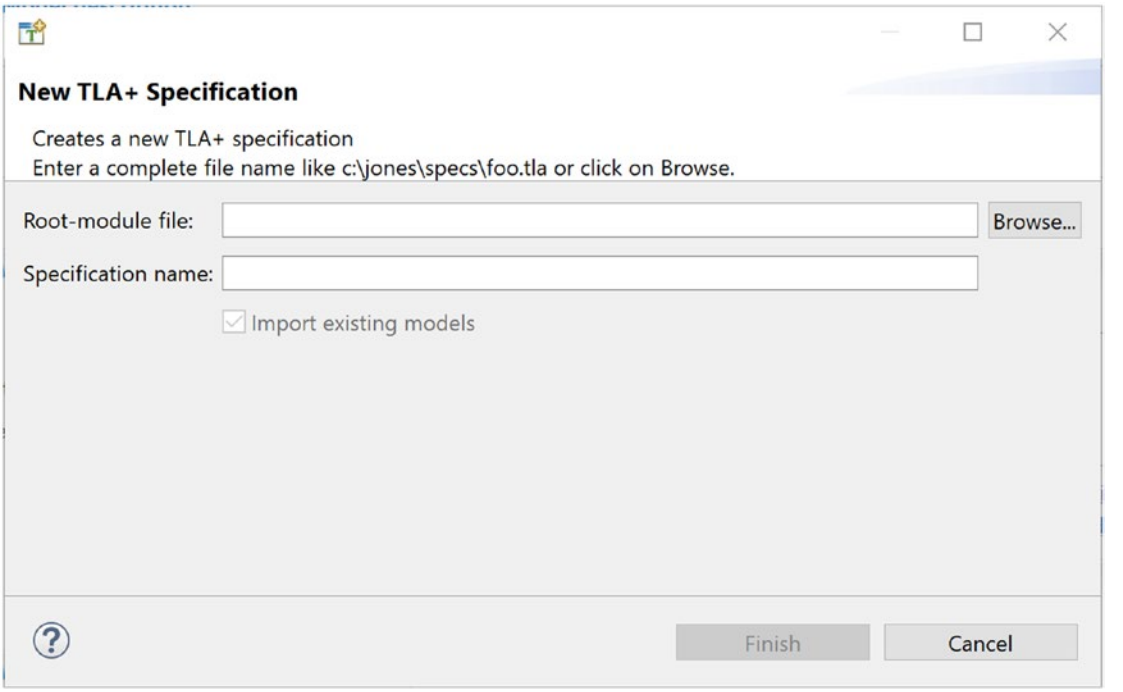


***Figure 1.*** *Add New Spec*

If you do, you've set up TLA+ correctly. If you plan to start with the example in the next chapter, name your first file `wire.tla` and the specification name to "wire" (or whatever you'd like to call the specification).

# PT

The PT library is a collection of useful operators and definitions that will make learning and using the language easier. Instead of having to spend time writing all of the utility operators, we can focus on the core idea of specification and delay the operator gymkata until later. You will have to manually download and set up PT:

1. Download the module from https://github.com/Apress/practical-tla-plus. Move it to wherever you plan on storing your TLA+ specs. You can also copy it from Appendix B.
2. Go to `File` ➤ `Preferences` ➤ `TLA+ Preferences`. You should see a control labeled "TLA+ library path locations."

3. Add the directory with PT. You should see something like what is shown in Figure 2.

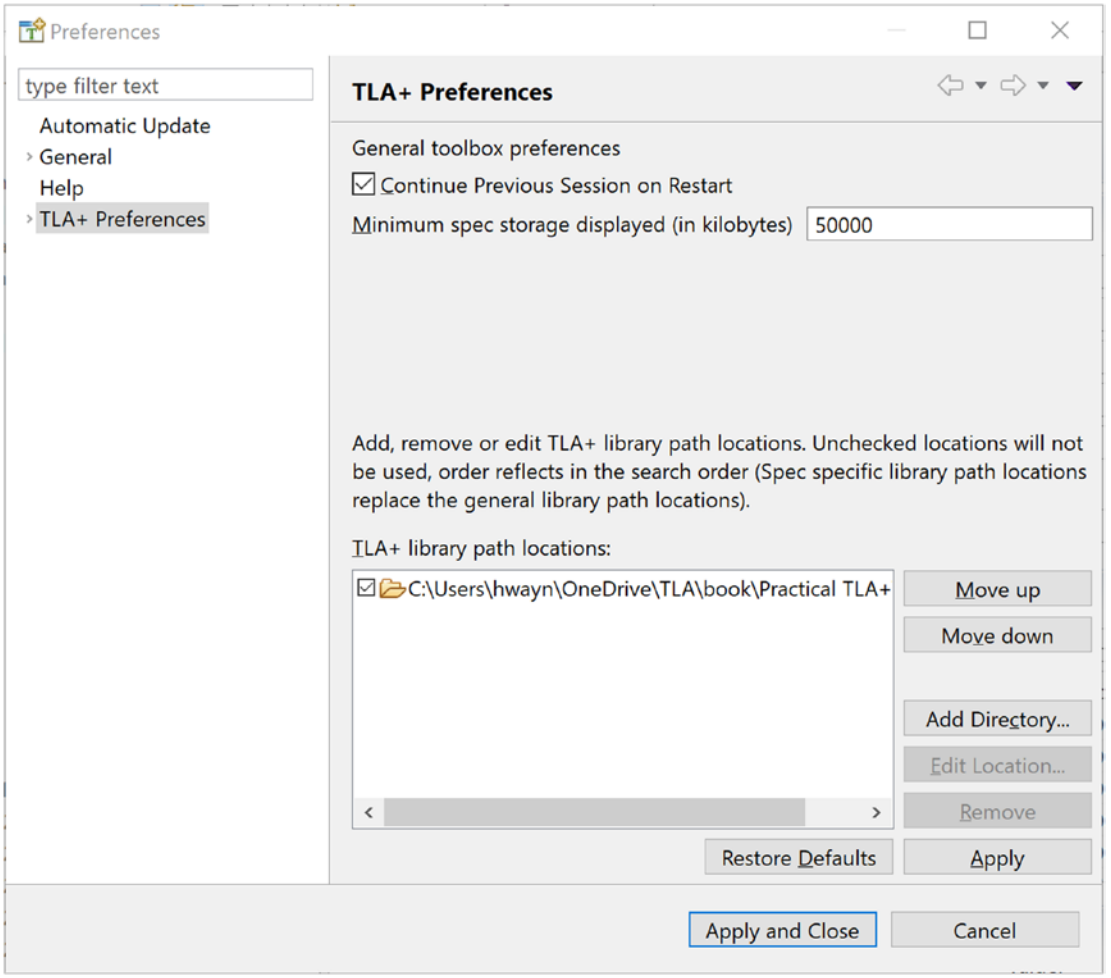


***Figure 2.*** *Library Path Location*

When you add PT to a specification, you will see it appear in the toolbox. I annotated all of the contents with descriptions of how it all works. Don't worry about understanding them just yet, but when you feel more confident, I'd recommend reading how they work. And don't be afraid to tweak them to make your own operators.

With that, we're ready to begin. Welcome to TLA+.