User Guide to Model Checking for Industrial Programmers with TLA+

v0.3 (9c338d44)

Gregory S. Miller
New York, NY
gshanemiller6@gmail.com

ABSTRACT

Systems software development comes with problems in number and complexity that are not easily solved through requirements or testing alone. Beyond organizational challenges coordinating human resources to good effect, software is beset with a large number of cross-interacting states across multiple threads of control. With an ever increasing emphasis on cloud-scale distributed systems, ensuring individual states compose well leading to correct system behavior is non-trivial. Model checking is a middle-way solution to these issues among a portfolio of choices from theorem provers to conventional iteration through best-effort testing at the other extreme.

Model checking comes with three salient features. It is constructive. Given a model coded in a simple language, the model verifier will report counter examples to any correctness claims. Second, verification explicitly constructs and analyses every reachable state over all possible execution orders. Unlike testing it is always complete. Third, model checking gives equal treatment to system states. While the details of a client database connection are always important, a database's ACID properties also depend on replication, leader election, quorums, and write-ahead logging. A model will formally describe how and when these distributed states are considered valid when taken together.

While most software developers are aware model checkers address many of these issues, it has yet remained alluringly locked away in niche safety critical sectors or academia. This paper throws back the curtain to address model checking through Microsoft's TLA (Temporal Logic of Actions) system. TLA is not the only choice in this space, but garners some attention at large tech companies where distributed software substantially underwrites engineering success.

This user guide is self-contained. No background is assumed. The approach here is practical on balance. Engineering is 51% right-doing, and 49% right-knowing. Doing each well is permanently caught-up in other, however, an extra nod to operational proficiency is the constant aim. To accomplish these goals minimal download and setup instructions are provided. If you have a recent version of the JAVA runtime installed, you're already done. A Github link is provided containing all the major examples herein including a pre-built TLA model checker, and Makefile. Once cloned simply run *make*.

To use TLA effectively the guide develops the knowledge component in four self-enforcing ways. The major distinguishing elements of modeling checking background is developed over ten pages. The middle part examines how correctness is assembled and specified in TLA model code feature by feature. The next two chapters work through non-trivial examples culminating in a 1800 line packet model. Finally, two appendices are provided describing the TLA language, and its procedural form called *PlusCal*. Numerous examples are provided throughout with extensive links.

Contents

1 MOTIVATION

It's interesting to watch organizations handle risk. Consider this fictitious story.

Cupertino 1985. On one corner of Main street, the big boss at *Acme Software Corp* calls an all-hands meeting.

"There's good news, and bad news. The good news is we've landed a major client in a lucrative deal. The bad news is they want our product line, but with numerous additional use-cases that'll change aspects of our architecture. Our sales chief, Bill, knows the use cases and is prepared to help coordinate with development. As with any major effort we are aware the customer will change their mind, refine, or de-emphasize requirements as the work comes into focus. It'll be a mess here and there. We'll need to be agile working closely with business, but I'm confident we can tech this out."

Meanwhile, the CEO at *Tech Corp* calls his own meeting across the street: