No More Garbage In: Validating Formal Models
A Short Course

Pamela Zave and Tim Nelson

May 25, 2023

1 Recommended reading before the lectures

Please read the Alloy tutorial at `https://hackmd.io/@lfs/BJ2sm-Eno`. You should also install Alloy (`https://alloytools.org`) on your laptop.

2 Glossary (for reference)

2.1 Terms about formal models (based on Alloy)

formal model
A *formal model* is a description of a state-transition system in a formal modeling language. Its semantics is a set of traces.

instance
An *instance* is a trace in the set of traces described by a formal model.

fact
A *fact* is a logical formula in a formal model. It is assumed to be true of all instances of the model.

assertion
An *assertion* is a logical formula in a formal model. It is usually intended to be true of all instances of the model, but this must be proved rather than assumed.

predicate
A *predicate* is a logical formula in a formal model. It is usually intended to be true of some instances of the model, but it must be instantiated to be sure.

2.2 Terms about computer systems

system

A *system* is the computer system (hardware or software) that we are interested in. All terms below are relative to this system. Note also that "state-transition system" in the definition of a formal model is just a standard term of art; our formal models will include specifications of the kind of system being defined here plus other parts relating to it.

domain

The *domain* is the environment of the system. It is the part of the world that surrounds and interacts with the system in a meaningful way.

domain knowledge

Domain knowledge is part of a formal model of a system. It is the part of the formal model that describes how the domain behaves all by itself, without the influence of the system.

specification

The *specification* is part of a formal model of a system. It is the part of the formal model that describes the behavior of the system, in a way that is simpler and more comprehensible than the implementation of the system.

requirements

The *requirements* are part of a formal model of a system. They are the part of the formal model that describes how the domain should behave, with the system implemented and installed.

interface

When a system is implemented and installed in a domain, some phenomena are shared, i.e., they are observable by both the domain and the system. These shared phenomena are the *interface* between the system and the domain. Note that each interface phenomenon is controlled (modified) by one entity, although it is observable from both.

implementation

The is the *implementation* of the system we are interested in.

validation

Validation is the partially informal and partially formal process of ensuring that a formal model of a system is accurate, precise, and comprehensible. Ideally, validation includes proving that the domain knowledge and specification, together, imply the requirements. In this proof, the specification is treated as a fact.

verification

Today, *verification* is just a synonym for "proof"—any kind of proof, with a strong connotation that the proof is completely or partially automated. In its original use, however, it meant "program verification," which is the formal process of proving that a system's implementation satisfies its specification. In this proof, the specification is treated as an assertion.

domain model

A *domain model* is a formal model of a family of systems, instead of just one. Validation of a domain model should also show that it is general (or extensible) and useful, as well as accurate, precise, and comprehensible.

3 Further reading

The order of these topics follows their order of introduction in the lectures.

About Alloy: The standard reference is the Alloy book [7]. In the earlier versions of Alloy it was most convenient to stick to traces with one or two states, while Alloy 6 supports full temporal logic. Many Alloy resources can be found at `https://alloytools.org`.

About networks: Compositional network architecture is the topic of a forthcoming book [22]. To manage your suspense until the book comes out, read the brief introduction in [21]—remembering that the book will be so much better! The formal model referred to in the lectures will appear on a Web site accompanying the book [19].

About predicates: Predicates are also great for testing software [3, 11, 16]. All the predicates in these papers are specifications of the software.

About visualization: Our labs show how valuable visualization can be in examining and understanding model instances. Here is some of the latest news on visualization: [5, 12].

About the Jackson-Zave model: The fullest description, with examples of many subtleties, is in the journal version [20]. Many people like the shorter version with one running example [8]. And the shortest version of all was published in a magazine [6].

About programming packet-processing hardware: P4 is a language that compiles to programmable hardware [2]. Lucid is a higher-level language that compiles to P4 [13]. Dafny is a modern programming language with built-in formal modeling and user-friendly verification [4, 9, 10].

About Chord: The best-known papers on Chord are [14] and [15]. A summary of what is known about the original Chord algorithm and its specification can be found in [17]. A correct version of Chord is proposed and verified in [18]; this is a particularly interesting example of formal modeling because the true invariant looks nothing like the obvious and necessary properties derived from it. The flawed conclusions are in [1], which is otherwise a very good paper.

References

[1] J.-P. Bodeveix, J. Brunel, D. Chemouil, and M. Filali. Mechanically verifying the fundamental liveness property of the Chord protocol. In *Formal Methods—The Next 30 Years*, pages 45–63. Springer, 2019.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communications Review*, 44(3), July 2014.

[3] K. Claessen and J. Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *International Conference on Functional Programming*, 2000.

[4] Dafny Documentation. `https://dafny-lang.github.io/dafny`. Last accessed 22 April 2022.

[5] T. Dyer and J. Baugh. Sterling: A web-based visualizer for relational modeling languages. In *Rigorous State Based Methods*, 2021.

[6] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

[7] D. Jackson. *Software Abstractions: Logic, Language, and Analysis.* MIT Press, 2006, 2012.

[8] M. Jackson and P. Zave. Deriving specifications from requirements: An example. In *Proceedings of the 17th International Conference on Software Engineering*, pages 15–24. ACM Press, April 1995.

[9] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In E. M. Clarke and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 348–370. Springer LNCS 6355, 2010.

[10] K. R. M. Leino. Modeling concurrency in Dafny. In J. P. Bowen, Z. Liu, and Z. Zhang, editors, *Engineering Trustworthy Software Systems*, pages 115–142. Springer LNCS 11174, 2018.

[11] T. Nelson, E. Rivera, S. Soucie, T. Del Vecchio, J. Wrenn, and S. Krishnamurthi. Automated, targeted testing of property-based testing predicates. In *The Art, Science, and Engineering of Programming*, 2022.

[12] A. Siegel, M. Santomauro, T. Dyer, T. Nelson, and S. Krishnamurthi. Prototyping formal methods tools: A protocol analysis case study. In *Protocols, Strands, and Logic*, pages 394–413, 2021.

[13] J. Sonchack, D. Loehr, J. Rexford, and D. Walker. Lucid: A language for control in the data plane. In *Proceedings of SIGCOMM.* ACM, 2021.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM.* ACM, 2001.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[16] J. Wrenn, T. Nelson, and S. Krishnamurthi. Using relational problems to teach property-based testing. *The Art, Science, and Engineering of Programming*, 5(3):Article 8, 2020.

[17] P. Zave. Using lightweight modeling to understand Chord. *ACM SIGCOMM Computer Communication Review*, 42(2):50–57, April 2012.

[18] P. Zave. Reasoning about identifier spaces: How to make chord correct. *ACM/IEEE Transactions on Software Engineering*, 43(12):1144–1156, December 2017. DOI 10.1109/TSE.2017.2655056.

[19] P. Zave. A formal model of Compositional Network Architecture. `http://compositionalnetarch.org`, to appear.

[20] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, January 1997.

[21] P. Zave and J. Rexford. The compositional architecture of the Internet. *Communications of the ACM*, 62(3):78–87, March 2019.

[22] P. Zave and J. Rexford. *The Real Internet Architecture: Past, Present, and Future Evolution.* Princeton University Press, 2024.