Case Study of Liveness Verification in IronFleet

Lucas Peña and Manasvi Saxena

University of Illinois at Urbana-Champaign,
{lpena7, msaxena2}@illinios.edu

Abstract. As more complicated software systems are becoming prevalent in today's world, more sophisticated program verification techniques are needed to reason about these. In particular, verification of liveness properties, or asserting that a property must occur, is traditionally difficult. In this paper, we survey the IronFleet methodology [2], a technique for building distributed systems that are provably correct, as it applies to verification of liveness properties. We consider the embedding of the Temporal Logic of Actions used in IronFleet, and discuss some limitations in their implementation.

1 Introduction

Many tools are used for ensuring correctness of a system. These range from testing to full formal verification.

1.1 Safety Verification

Most formal verification techniques focus on verifying safety properties. Informally, a safety property asserts that "nothing bad ever happens." More formally, this means if a system violates a safety property P at any point in its execution, the full execution must also violate P. Thus, safety properties have *finite length counterexamples*. Because of this crucial fact, many techniques such as model checking and monitoring focus specifically on verifying safety properties.

1.2 Liveness Verification

In contrast, a liveness property asserts that "somerthing good will happen." This means one must have the entire execution in hand in order to assert the absence of a liveness property. Thus, liveness properties have *infinite length counterexamples*. Thus, though liveness verification has the same theoretical complexity as safety verification (cite), in practice it is much more difficult to verify liveness properties. Clearly, it is fruitless to do any kind of brute force search for the absence of such a counterexample. Instead, specific techniques for verifying liveness properties have been developed, which we discuss in Section **??**.

The IronFleet methodology uses a proof strategy based on Lamport's Temporal Logic of Actions (TLA) that chains smaller proofs together to assert the existence of some final condition (see Section 2.4). Unlike the related work on liveness verification, IronFleet is the first to verify nontrivial liveness properties on a large-scale distributed system.

2 IronFleet

2.1 Dafny

2.2 End-to-end Verification

2.3 Refinement

2.4 Liveness Verification in IronFleet

Limitations

3 Temporal Logic of Actions

The Temporal Logic of Actions (TLA) (cite) is an extension of Linear Temporal Logic introduced to reason about concurrent systems. TLA introduces "primed" variables, which represents the value of a variable in the next state. For example, the TLA *action*

$$x' = x + y$$

specifies that the value of x in the next state is equal to the value of $x + y$ in the current state. An action is satisfied by a pair of states $\langle s, t \rangle$, where s is a valuation on the unprimed variables and t is a valuation on the primed variables. The notion of primed variables allow us to specify a rich class of formulas not expressible in normal LTL.

First, we can represent the action $Unchanged\ f \equiv f' = f$ for any function f without primed variables, which states that f is a stuttering step. Using this, we can also represent the derived actions $[\mathcal{A}]_f$ and $\langle \mathcal{A} \rangle_f$ for any action $\mathcal{A}$. These respectively represent that an action either satisfies $\mathcal{A}$ or stutters (w.r.t. f), and that an action that satisfies $\mathcal{A}$ necessarily changes f. These are defined as follows:

$$[\mathcal{A}]_f \equiv \mathcal{A} \vee Unchanged\ f \qquad\qquad \langle \mathcal{A} \rangle_f \equiv \mathcal{A} \wedge \neg(Unchanged\ f)$$

TLA also introduces the action $Enabled\ \mathcal{A}$ which specifies that for any s, there is a t such that $\langle s, t \rangle$ satisfies $\mathcal{A}$.

TLA *formulas* add the temporal operator $\Box$, like in LTL. In addition, quantification over formulas is allowed. Derived formulas such as $\Diamond$ are as in LTL. Other derived formulas include $F \rightsquigarrow G$ (*leads to*), $\mathrm{WF}_f(\mathcal{A})$ (*weak fairness*), and $\mathrm{SF}_f(\mathcal{A})$ (*strong fairness*). These are defined as follows:

$$\begin{aligned} F \rightsquigarrow G &\equiv \Box(F \Rightarrow \Diamond G) \\ \mathrm{WF}_f(\mathcal{A}) &\equiv \Box\Diamond\langle \mathcal{A} \rangle_f \vee \Box\Diamond\neg Enabled\ \langle \mathcal{A} \rangle_f \\ \mathrm{SF}_f(\mathcal{A}) &\equiv \Box\Diamond\langle \mathcal{A} \rangle_f \vee \Diamond\Box\neg Enabled\ \langle \mathcal{A} \rangle_f \end{aligned}$$

For use in verification, these derived formulas can express most liveness properties a user may want to express. Of course, for verification, we need proof rules allowing us to reason about these formulas, which we present below.

WF1.

$$\frac{P \wedge [\mathcal{N}]_f \;\Rightarrow\; (P' \vee Q') \qquad P \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_f \;\Rightarrow\; Q' \qquad P \;\Rightarrow\; \mathit{Enabled}\ \langle \mathcal{A} \rangle_f}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \;\Rightarrow\; (P \rightsquigarrow Q)}$$

WF2.

$$\frac{\langle \mathcal{N} \wedge \mathcal{B} \rangle_f \;\Rightarrow\; \langle \mathcal{M} \rangle_g \qquad P \wedge P' \wedge \langle \mathcal{N} \wedge \mathcal{A} \rangle_f \;\Rightarrow\; \mathcal{B} \qquad P \wedge \mathit{Enabled}\ \langle \mathcal{M} \rangle_g \;\Rightarrow\; \mathit{Enabled}\ \langle \mathcal{A} \rangle_f \qquad \Box[\mathcal{N} \wedge \neg\mathcal{B}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \;\Rightarrow\; \Diamond\Box P}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \;\Rightarrow\; \mathrm{WF}_g(\mathcal{M})}$$

These rules allow us to reason about both strong fairness and weak fairness, both as a hypothesis and a conclusion. The use of these rules is demonstrated via a simple program that nondeterministically chooses one of two variables to increment ad infinitum. One can define a naive implementation of this program, as well as a more complex one using semaphores.

The TLA rules above can be used to prove weak fairness of the first implementation, strong fairness of the second, and it can even be used to prove that the semaphore implementation is a refinement of the naive implementation. Notably, all four of the above rules must be used for these proofs.

4 Related Work

Though verification of liveness properties is notoriously difficult and rare, the IronFleet methodology was of course not the first to formally verify liveness properties. Previous techniques include BDD-based model checking (cite), converting liveness properties to safety properties (cite), and simulation-based liveness checking (cite). See [3].

5 Conclusion

References

1. C. Hawblitzel, J. Howell, J. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill, "Ironclad apps: End-to-end security via automated full-system verification," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, USENIX Advanced Computing Systems Association, October 2014.
2. C. Hawblitzel, J. Howell, M. Kapritsos, J. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, "Ironfleet: Proving practical distributed systems correct," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, ACM Association for Computing Machinery, October 2015.
3. P. K. Nalla, R. K. Gajavelly, H. Mony, J. Baumgartner, and R. Kanzelman, "Effective liveness verification using a transformation-based framework," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, Mumbai, India, January 5-9, 2014*, pp. 74–79, 2014.