When Human Intuition Fails: Using Formal Methods to Find an Error in the "Proof" of a Multi-Agent Protocol

Jennifer A. Davis[1], Laura R. Humphrey[2], Derek B. Kingston[3*]

[1] Collins Aerospace, Cedar Rapids, IA 52498, USA, `jen.davis@collins.com`
[2] Air Force Research Lab, Dayton, OH 45433, USA `laura.humphrey@us.af.mil`
[3] Aurora Flight Sciences, Manassas, VA 20110, USA `kingston.derek@aurora.com`

Abstract. Designing protocols for multi-agent interaction that achieve the desired behavior is a challenging and error-prone process. The standard practice is to manually develop proofs of protocol correctness that rely on human intuition and require significant effort to develop. Even then, proofs can have mistakes that may go unnoticed after peer review, modeling and simulation, and testing. The use of formal methods can reduce the potential for such errors. In this paper, we discuss our experience applying model checking to a previously published multi-agent protocol for unmanned air vehicles. The original publication provides a compelling proof of correctness, along with extensive simulation results to support it. However, analysis through model checking found an error in one of the proof's main lemmas. In this paper, we start by providing an overview of the protocol and its original "proof" of correctness, which represents the standard practice in multi-agent protocol design. We then describe how we modeled the protocol for a three-vehicle system in a model checker, the counterexample it returned, and the insight this counterexample provided. We also discuss benefits, limitations, and lessons learned from this exercise, as well as what future efforts would be needed to fully verify the protocol for an arbitrary number of vehicles.

Keywords: Multi-agent systems · distributed systems · autonomy · model checking

1 Introduction

Many robotics applications require multi-agent interaction. However, designing protocols for multi-agent interaction that achieve the desired behavior can be challenging. The design process is often manual, i.e. performed by humans, and generally involves creating mathematical models of possible agent behaviors and candidate protocols, then manually developing a proof that the candidate protocols are correct with respect to the desired behavior. However, human-generated

* Supported by AFRL/RQ contract #FA8650-17-F-2220 and AFOSR award #17RQ-COR417. DISTRIBUTION A. Approved for public release: distribution unlimited. Case #88ABW-2018-4275.

proofs can have mistakes that may go unnoticed even after peer review, modeling and simulation, and testing of the resulting system.

Formal methods have the potential to reduce such errors. However, while the use of formal methods in multi-agent system design is increasing [2], [6], [8], [11], it is our experience that manual approaches are still the norm. Here, we hope to motivate the use of formal methods for multi-agent system design by demonstrating their value in a case study involving a manually designed decentralized protocol for dividing surveillance of a perimeter across multiple unmanned aerial vehicles (UAVs). This protocol, called the Decentralized Perimeter Surveillance System (DPSS), was previously published in 2008 [10], has received close to 200 citations to date, and provides a compelling "proof" of correctness backed by extensive simulation results.

We start in Section 2 by giving an overview of DPSS, the convergence bounds that comprise part of its specification, and the original "proof" of correctness. In Section 3, we give an overview of the three-UAV DPSS model we developed in the Assume Guarantee REasoning Environment (AGREE) model checker [3]. In Section 4, we present the analysis results returned by AGREE, including a counterexample to one of the convergence bounds. Section 5 concludes with a discussion of benefits, challenges, and limitations of our modeling process and how to help overcome them, and what future work would be required to modify and fully verify DPSS for an arbitrary number of UAVs.

2 Decentralized Perimeter Surveillance System (DPSS)

UAVs can be used to perform continual, repeated surveillance of a large perimeter. In such cases, more frequent coverage of points along the perimeter can be achieved by evenly dividing surveillance of it across multiple UAVs. However, coordinating this division is challenging in practice for several reasons. First, the exact location and length of the perimeter may not be known a priori, and it may change over time, as in a growing forest fire or oil spill. Second, UAVs might go offline and come back online, e.g. for refueling or repairs. Third, inter-UAV communication is unreliable, so it is not always possible to immediately communicate local information about perimeter or UAV changes. However, such information is needed to maintain an even division of the perimeter as changes occur. DPSS provides a method to solve this problem with minimal inter-UAV communication for perimeters that are isomorphic to a line segment.

Let the perimeter start as a line segment along the x-axis with its left endpoint at $x = 0$ and its right at $x = P$. Let N be the number of UAVs in the system or on the "team," indexed from left to right as $1, \ldots, N$. Divide the perimeter into segments of length P/N, one per UAV. Then the optimal configuration of DPSS as depicted in Fig. 1 is defined as follows (see Ref. [10] for discussion of why this definition is desirable).

Definition 1. *Consider two sets of perimeter locations: (1)* $\lfloor i + \frac{1}{2}(-1)^i \rfloor P/N$ *and (2)* $\lfloor i - \frac{1}{2}(-1)^i \rfloor P/N$*, where* $\lfloor \cdot \rfloor$ *returns the largest integer less than or equal* *to its argument. The optimal configuration is realized when UAVs synchronously oscillate between these two sets of locations, each moving at constant speed V.*

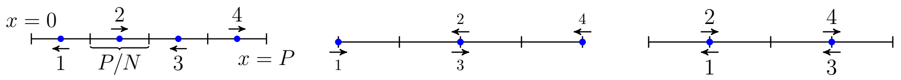

Fig. 1: Optimal DPSS configuration, in which UAVs are evenly spaced along the perimeter and synchronously oscillate between segment boundaries.

The goal of DPSS is to achieve the optimal configuration in the steady state, i.e. when the perimeter and involved UAVs remain constant. The DPSS protocol itself is relatively simple. Each UAV i stores a vector $\xi_i = [P_{R_i} \quad P_{L_i} \quad N_{R_i} \quad N_{L_i}]^T$ of coordination variables that capture its beliefs (which may be incorrect) about perimeter length P_{R_i} and P_{L_i} and number of UAVs N_{R_i} and N_{L_i} to its right and left. When neighboring UAVs meet, “left” UAV i learns updated values for its “right” variables $P'_{R_i} = P_{R_{i+1}}$ and $N'_{R_i} = N_{R_{i+1}} + 1$ from “right” UAV $i + 1$, and likewise UAV $i + 1$ updates its “left” variables $P'_{L_{i+1}} = P_{L_i}$ and $N'_{L_{i+1}} = N_{L_i} + 1$. While values for these variables may still be incorrect, the two UAVs will at least have matching coordination variables and thus a consistent estimate of their shared segment boundary. The two UAVs then “escort” each other to their estimated shared segment boundary, then split apart to surveil their own segment. Note that UAVs only change direction when they reach a perimeter endpoint or when starting or stopping an escort, which means a UAV will travel outside its segment unless another UAV arrives at the segment boundary at the same time (or the end of the segment is a perimeter endpoint).

Eventually, leftmost UAV 1 will discover the actual left perimeter endpoint, accurately set $N_{L_1} = 0$ and $P_{L_1} = 0$, then turn around and update P_{L_1} continuously as it moves. A similar situation holds for rightmost UAV n. Accurate information will be passed along to other UAVs as they meet, and eventually all UAVs will have correct coordination variables and segment boundary estimates. Since UAVs also escort each other to shared segment boundaries whenever they meet, eventually the system reaches the optimal configuration, in which UAVs oscillate between their true shared segment boundaries.

An important question is how long it takes DPSS to converge to the optimal configuration. Each time the perimeter or number of UAVs changes, it is as if the system is reinitialized; UAVs no longer have correct coordination variables and so the system is no longer converged. However, if DPSS is able to re-converge relatively quickly, it will often be in its converged state.

Ref. [10] claims that DPSS converges within $5T$, where $T = P/V$ is the time it would take a single UAV to traverse the entire perimeter if there were no other UAVs in the system. It describes DPSS as two algorithms: Algorithm A, in which UAVs start with correct coordination variables, and Algorithm B, in which they do not. The proof strategy is then to argue that Algorithm A converges in $2T$ (Theorem 1) and Algorithm B achieves correct coordination variables in $3T$

(Lemma 1)[4]. At that point, Algorithm B converts to Algorithm A, so the total convergence time is $2T + 3T = 5T$ (Theorem 2)[5].

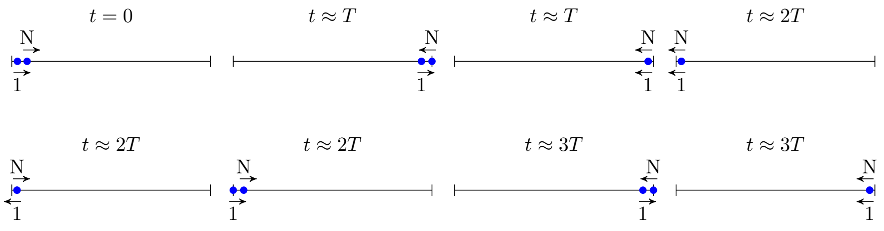

Fig. 2: Claimed worst-case coordination variable convergence for Algorithm B.

Informally, the original argument for Lemma 1 is that information takes time T to travel along the perimeter. The worst case occurs when all UAVs start near one end of the perimeter, e.g. the left endpoint, so that the rightmost UAV N reaches the right endpoint around time T. UAV N then turns around and through a fast series of meetings, correct "right" coordination variables are propagated to the other UAVs, all of which then start moving left. Due to incorrect "left" coordination variables, UAV $N-1$ and UAV N might think their shared segment boundary is infinitesimally close to the left endpoint. The UAVs travel left until they are almost at the left perimeter endpoint around time $2T$. However, since UAV N thinks its segment boundary is near the left endpoint, it ends its escort and goes right without learning the true location of the left perimeter endpoint. Leftmost UAV 1 learns the true location of the left perimeter endpoint and this information will be passed to the other UAVs as they meet, but the information will have to travel the perimeter once again to reach the rightmost UAV N around time $3T$. This situation is depicted in Fig. 2.

Through model checking, we were able to find a counterexample to this claimed bound, which will be presented in Section 4. But first, we overview the model used for analysis through model checking.

3 Formal Models

We briefly overview the formal models developed in AGREE for a three-UAV version of DPSS as described by Algorithm B. Models for Algorithm A and Algorithm B along with a more detailed description of the Algorithm B model are available on GitHub [1][6].

[4] We label this Lemma 1 for convenience; it is unlabeled in [10].

[5] A version of the original proof is on GitHub [1] in file dpssOriginalProof.pdf.

[6] AADL projects are in AADL_sandbox_projects. Algorithm A and B models for three UAVs are in projects DPSS-3-AlgA-for-paper and DPSS-3-AlgB-for-paper. A description of the Algorithm B model is in file modelAlgorithmB.pdf.

AGREE is an infinite-state model checker capable of analyzing systems with real-valued variables, as is the case with DPSS. AGREE uses assume/guarantee reasoning to verify properties of architectures modeled as a top-level system with multiple lower-level components, each having a formally specified assume/guarantee contract. Each contract consists of a set of assumptions on the inputs and guarantees on the outputs, where inputs and outputs can be reals, integers, or booleans. System assumptions and component assume/guarantee contracts are assumed to be true. AGREE then attempts to verify that (a) component assumptions hold given system assumptions, and (b) system guarantees hold given component guarantees. AGREE poses this verification problem as a satisfiability modulo theory (SMT) problem [4] and uses a k-induction model checking approach [7] to search for counterexamples that violate system-level guarantees given system-level assumptions and component-level assume/guarantee contracts. The language used by AGREE is an "annex" to the Architecture Analysis and Design Language (AADL) [5].

AGREE's ability to analyze systems modeled as a top-level system with multiple lower-level components provides a natural fit for DPSS. The three-UAV AGREE DPSS model consists of a single top-level system model, which we call the "System," and a component-level UAV model that is instantiated three times, which we call the "UAV(s)." The System essentially coordinates a discrete event simulation of the UAVs as they execute the DPSS protocol, where events include a UAV reaching a perimeter endpoint or two UAVs starting or stopping an escort. In the initial state, the System sets valid ranges for each UAV's initial position through assumptions that constrain the UAVs to be initialized between the perimeter endpoints and ordered by ID number from left to right. System assumptions also constrain UAV initial directions to be either left or right (though a UAV might have to immediately change this value, e.g., if it is initialized at the left endpoint headed left). These values become inputs to the UAVs. The System determines values for other UAV inputs, including whether a UAV is co-located with its right or left neighbor and the true values for the left and right perimeter endpoints. Note the true perimeter endpoints are only used by the UAVs to check whether they have reached the end of the perimeter, not to calculate boundary segment endpoints. The System also establishes data ports between UAVs, so that each UAV can receive updated coordination variable values from its left or right neighbor as inputs and use them (but only if they are co-located).

The last System output that serves as a UAV input is the position of the UAV. At initialization and after each event, the System uses the globally known constant UAV speed V and other information from each UAV to determine the amount of time δt until the next event, and then it updates the position of each UAV. Determining the time of the next event requires knowing the direction and next anticipated "goal" location of each UAV, e.g. estimated perimeter endpoint or shared segment boundary. Each UAV outputs these values, which become inputs to the System. Each UAV also outputs its coordination variables P_{R_i}, P_{L_i}, N_{R_i}, and N_{L_i}, which become System inputs that are used in System guarantees that formalize Theorem 1, Lemma 1, and Theorem 2 of Section 2. Note that we bound integers N_{R_i} and N_{L_i} because in order to calculate estimated boundary segments, which requires dividing perimeter length by the number of UAVs, we must implement a lookup table that copies the values of N_{R_i} and N_{L_i} to real-valued versions of these variables. This is due to an interaction between AGREE and the Z3 SMT solver [4] used by AGREE. If we directly cast N_{R_i} and N_{L_i} to real values in AGREE, they are encoded in Z3 using the `to_real` function. Perimeter values P_{R_i} and P_{L_i} are directly declared as reals. However, Z3 views integers converted by the `to_real` function as constrained to have integer values, so it cannot use the specialized solver for reals that is able to analyze this model.

4 Formal Analysis Results

In this section, we discuss the analysis results provided by AGREE for Algorithm A and Algorithm B, though we focus on Algorithm B.

Algorithm A: Using AGREE configured to utilize the JKind k-induction model checker [7] and the Z3 SMT solver, we have proven Theorem 1, that Algorithm A converges within $2T$, for N = 1, 2, 3, 4, 5, and 6 UAVs. Computation time prevented us from analyzing more than six UAVs. For reference, N = 1 through N = 4 ran in under 10 minutes each on a laptop with two cores and 8 GB RAM. The same laptop analyzed N = 5 overnight. For N = 6, the analysis took approximately twenty days on a computer with 40 cores and 128 GB memory.

Algorithm B: We were able to prove Theorem 2, that DPSS converges within $5T$, for N = 1, 2, and 3 UAVs and with each UAV's coordination variables N_{R_i} and N_{L_i} bounded between 0 and 20. In fact, we found the convergence time to be within $(4+\frac{1}{3}T)$. However, AGREE produced a counterexample to Lemma 1, that every UAV obtains correct coordination variables within $3T$, for N = 3. In fact, we incrementally increased this bound and found counterexamples up to $(3+\frac{1}{2})T$ but that convergence is guaranteed in $(3+\frac{2}{3})T$.

One of the shorter counterexamples provided by AGREE shows the UAVs obtaining correct coordination variables in 3.0129T. Full details are available on GitHub[1],[7] but we outline the steps in Fig. 3. In this counterexample, UAV 1 starts very close to the left perimeter heading right, and UAVs 2 and 3 start in the middle of segment 3 headed left. UAVs 1 and 2 meet near the middle of the perimeter and head left toward what they believe to be their shared segment boundary. This is very close to the left perimeter endpoint because, due to initial conditions, they believe the left perimeter endpoint to be much farther away than it actually is. Then they split, and UAV 1 learns where the left perimeter endpoint actually is, but UAV 2 does not. UAV 2 heads right and meets UAV 3 shortly afterward, and they move to what they believe to be their shared segment boundary, which is likewise very close to the right perimeter endpoint. Then they split, and UAV 3 learns where the right perimeter endpoint is, but UAV 2 does not. UAV 2 heads left, meets UAV 1 shortly after, and

[7] A spreadsheet with counterexample values for all model variables is located under AADL_sandbox_projects/DPSS-3-AlgB-for-paper/results_20180815_eispi.

learns correct "left" coordination variables. However, UAV 2 still believes the right perimeter endpoint to be farther away than it actually is, so UAV 1 and 2 estimate their shared segment boundary to be near the middle of the perimeter. They then head toward this point and split apart, with UAV 1 headed left and still not having correct "right" coordination variables. UAV 2 and 3 then meet, exchange information, and now both have correct coordination variables. They go to their actual shared boundary, split apart, and UAV 2 heads left toward UAV 1. UAV 1 and 2 then meet on segment 1, exchange information, and now all UAVs have correct coordination variables.

The counterexample reveals a key intuition that was missing in Lemma 1. The original argument did not fully consider the effects of initial conditions and so only considered a case in which UAVs came close to *one* end of the perimeter without actually reaching it. The counterexample shows it can happen at *both* ends if initial conditions cause the UAVs to believe the perimeter endpoints to be farther away than they actually are. This could happen if the perimeter were to quickly shrink, causing the system to essentially "reinitialize" with incorrect coordination variables.

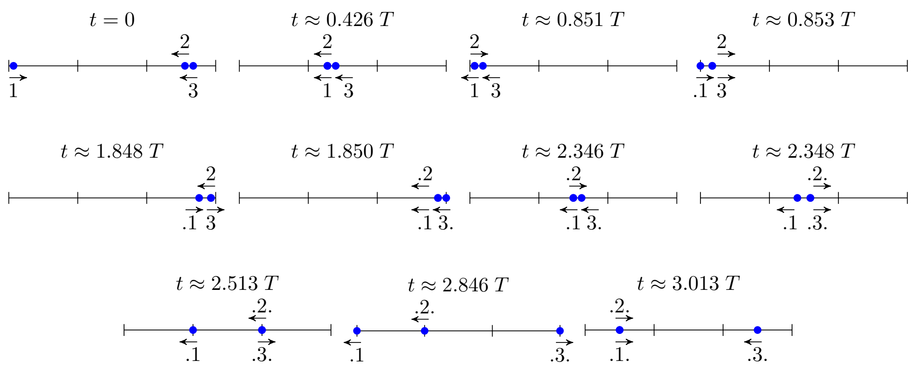

Fig. 3: Counterexample to Lemma 1. Dots to the left of a UAV number indicate it has correct "left" variables, and likewise for the right.

Analysis for three UAVs for Algorithm B completed in 18 days on a machine with 256 GB RAM and 80 cores.

5 Discussion and Conclusions

Formal modeling and analysis through AGREE had many benefits. First, it allowed us to analyze DPSS, a decentralized protocol for distributing a surveillance task across multiple UAVs. Though the original publication on DPSS provided a convincing human-generated proof and simulation results to support claims about its convergence bounds, analysis revealed that one of the key lemmas was incorrect. Furthermore, the counterexample returned by AGREE provided insight into why it was incorrect. Second, formal modeling in and of itself allowed us to find what were essentially technical typos in the original paper. For example, the formula for dividing the perimeter across UAVs only accounted for changes in estimates of the right perimeter endpoint and not the left, so we corrected the formula for our model. We also discovered that certain key aspects of the protocol were underspecified. In particular, it is unclear what should happen if more than two UAVs meet at the same time. Analysis showed this occurring for as little as three UAVs in Algorithm B, and simulations in the original paper showed this happening frequently, but this behavior was not explicitly described. Here, we decided that if all three UAVs meet to the left of UAV 3's estimated segment, UAV 3 immediately heads right and the other two follow the normal protocol to escort each other to their shared border. Otherwise, the UAVs all travel left together to the boundary between segments 2 and 3, then UAV 3 breaks off and heads right while the other two follow the normal protocol.

This brings us to a discussion of challenges and limitations. First, in terms of more than two UAVs meeting at a time, simulations in the original paper implement a more complex behavior in which UAVs head to the closest shared boundary and then split apart into smaller and smaller groups until reaching the standard case of two co-located UAVs. This behavior requires a more complex AGREE model that can track "cliques" of more than two UAVs, and it is difficult to validate the model due to long analysis run times. Second, we noted in Section 4 that in our model, UAV coordination variables N_{R_i} and N_{L_i} have an upper bound of 20. In fact, with an earlier upper bound of 3, we found the bound for Lemma 1 to be $(3+\frac{1}{3})T$ and did not consider that it would depend on upper bounds for N_{R_i} and N_{L_i}. We therefore cannot conclude that even $(3+\frac{2}{3})T$ is the convergence time for Lemma 1. Third and related to the last point, model checking with AGREE can only handle up to three UAVs for Algorithm B. Due to these limitations, we cannot say for sure what the upper bound for DPSS actually is, even if we believe it to be $5T$. If it is higher, then it takes DPSS longer to converge, meaning it can handle less frequent changes than originally believed. We are therefore attempting to transition to theorem provers such as ACL2 [9] and PVS [12] to develop a proof of convergence bounds for an arbitrary number of UAVs, upper bound on N_{R_i} and N_{L_i}, and perimeter length (which was set to a fixed size to make the model small enough to analyze).

In terms of recommendations and lessons learned, it was immensely useful to work with the author of DPSS to formalize our model. Multi-agent protocols like DPSS are inherently complex, and it is not surprising that the original paper contained some typos, underspecifications, and errors. In fact, the original paper explains DPSS quite well and is mostly correct, but it is still challenging for formal methods experts to understand complex systems from other disciplines, so access to subject matter experts can greatly speed up formalization.

Acknowledgment

We thank John Backes for his guidance on efficiently modeling DPSS in AGREE and Aaron Fifarek for running some of the longer AGREE analyses.

References

1. OpenUxAS GitHub repository, dpssModel branch, `https://github.com/afrl-rq/OpenUxAS/tree/dpssModel`
2. Alur, R., Moarref, S., Topcu, U.: Compositional synthesis of reactive controllers for multi-agent systems. In: International Conference on Computer Aided Verification. pp. 251–269. Springer (2016)
3. Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional verification of architectural models. In: NASA Formal Methods Symp. pp. 126–140. Springer (2012)
4. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
5. Feiler, P.H., Lewis, B.A., Vestal, S.: The SAE Architecture Analysis & Design Language (AADL): A standard for engineering performance critical systems. In: IEEE Int. Conf. Computer Aided Control System Design. pp. 1206–1211. IEEE (2006)
6. Fisher, M., Dennis, L., Webster, M.: Verifying autonomous systems. Communications of the ACM **56**(9), 84–93 (2013)
7. Gacek, A., Backes, J., Whalen, M., Wagner, L., Ghassabani, E.: The JKind model checker. In: Int. Conf. Computer Aided Verification. pp. 20–27. Springer (2018)
8. Guo, M., Tumova, J., Dimarogonas, D.V.: Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints. In: Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on. pp. 75–80. IEEE (2014)
9. Kaufmann, M., Moore, J.S.: An industrial strength theorem prover for a logic based on Common Lisp. IEEE Trans. Software Engineering **23**(4), 203–213 (1997)
10. Kingston, D., Beard, R.W., Holt, R.S.: Decentralized perimeter surveillance using a team of UAVs. IEEE Trans. Robotics **24**(6), 1394–1404 (2008)
11. Kupermann, O., Vardi, M.: Synthesizing distributed systems. In: Proc. 16th Annual IEEE Symp. Logic in Computer Science. pp. 389–398. IEEE (2001)
12. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: Int. Conf. Automated Deduction. pp. 748–752. Springer (1992)