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

Jennifer Davis[1][0000−0003−4121−5297], Derek Kingston[2,3], and Laura Humphrey[2]

[1] Collins Aerospace, Cedar Rapids, USA, jen.davis@collins.com
[2] Air Force Research Laboratory, Dayton, USA, laura.humphrey@us.af.mil
[3] Aurora Flight Sciences, Manassas, 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 inter-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

* This work was supported by AFRL/RQ contract #FA8650-17-F-2220 and AFOSR award #17RQCOR417. 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) [3], an infinite-state model checker capable of analyzing systems with real-valued variables. 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 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 desireable).

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.

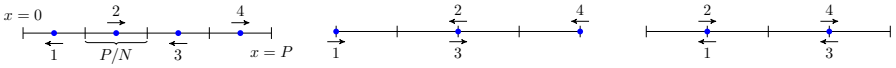

Fig. 1: Optimal DPSS configuration, in which UAVs are evenly spaced along the perimeter and synchronously oscillate between segment boundaries.

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 $\xi_i = [P_{R_i} \;\; P_{L_i} \;\; N_{R_i} \;\; N_{L_i}]^T$ of coordination variables that capture its beliefs 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 $N'_{R_i} = N_{R_{i+1}}$ from "right" UAV $i+1$, and likewise UAV $i+1$ updates its "left" variables $P'_{L_{i+1}} = P_{L_i}$ and $N'_{L_{i+1}} = N_{R_i}$. 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 segment boundaries. 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 it is a perimeter endpoint).

Eventually, leftmost UAV 1 will discover the actual left perimeter endpoint and accurately set $N_{L_1} = 0$ and $P_{L_1} = 0$, then turn around and update P_{L_1} continuously as it moves. A similiar 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. The additional process of UAVs escorting each other to shared segment boundaries leads to the synchronization required for the optimal configuration.

An important question is how long it takes DPSS to converge to the optimal configuration, given that it is intended to adjust to changes in perimeter length and team composition. The original proof of DPSS's convergence time from Ref. [10] (slightly modified for clarity) is contained in Appendix A. The proof strategy describes DPSS as two algorithms: Algorithm A, in which UAVs start with correct coordination variables, and Algorithm B, in which they do not. Let $T = P/V$ be the time it takes one UAV to traverse the entire perimeter. The proof strategy is to argue that Algorithm A converges in $2T$ (Theorem 1), and Algorithm B achieves correct coordination variables in $3T$ (Lemma 1). At that point, Algorithm B converts to Algorithm A, so the total convergence time is $2T + 3T = 5T$ (Theorem 2).

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

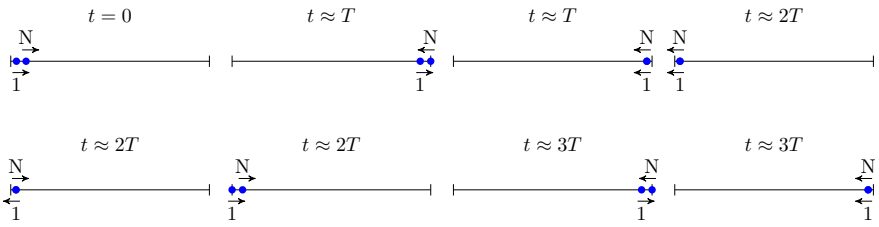

Fig. 2: Claimed worst case coordination variable convergence for Algorithm B.

reaches the right endpoint in time T. UAV N then turns around and through a fast series of meetings, correct "right" coordination variables are propagated to the other UAVs. Due to incorrect "left" coordination variables, the UAVs might all think their shared segment boundaries are infinitesimally close to the left endpoint. The UAVs turn around and travel left, and leftmost UAV 1 reaches 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 turns to the right just before learning the correct distance to the left endpoint. Leftmost UAV 1 will pass this information to the other UAVs, but the information will have to travel the perimeter once again to reach the rightmost UAV N at time $3T$. This situation is depicted in Fig. 6. 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. A more complete overview is contained in Appendix B, and complete models for Algorithms A and B can be found on GitHub [1][4].

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 languaged used by AGREE is an "annex" to the Architecture Analysis and Design Language (AADL) [5].

[4] AADL projects are in AADL_sandbox_ projects. Algorithm A and B models for three UAVs are in DPSS-3-AlgA-for-paper project and DPSS-3-AlgB-for-paper project.

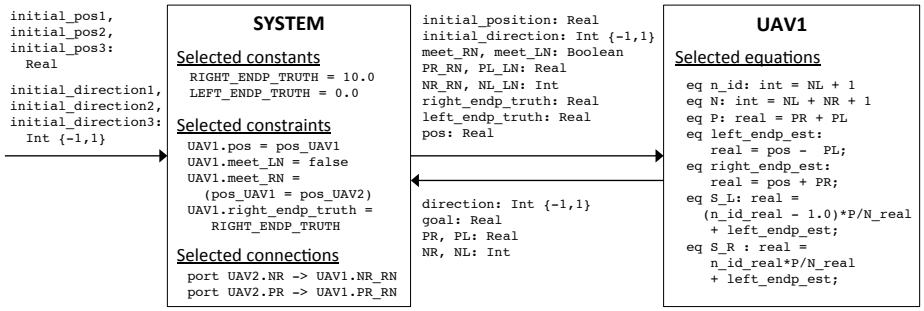

Fig. 3: Inputs and outputs for the System and UAV 1, with selected representative constants, constraints, data connections, and equations used by each.

The model consists of a single top-level system model and a component-level UAV model that is instantiated three times. We henceforth refer to these simply as the "System" and the "UAV(s)." System and UAV inputs and outputs, along with a small number of selected representative constraints, data connections, and equations used by each, are shown in Fig. 3.

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 in order from left to right (i.e., `initial_pos1 < initial_pos2` and `initial_pos2 < initial_pos3`) and between the perimeter endpoints (e.g., `initial_pos_UAV1 >= LEFT_ENDP_TRUTH` and `initial_pos_UAV1 <= RIGHT_ENDP_TRUTH`). System assumptions also constrain UAV initial directions to be either left (`-1`) or right (`1`), though a UAV might have to change this value, e.g., if it is initialized at the left endpoint headed left. These values become inputs to the UAVs. The System constrains specific values for other UAV inputs, including whether a UAV is co-located with its right or left neighbor (`meet_RN` and `meet_LN`), and the true values for the left and right perimeter endpoints (`right_endp_truth` and `left_endp_truth`). 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 right neighbor (`PR_RN` and `NR_RN`) or left neighbor (`PL_LN` and `NL_LN`) as inputs and use them if they are co-located.

The last System output that serves as a UAV input is `pos`. 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 `deltaT` until the next event. It also updates `pos` for each UAV based on `V` and `deltaT`. Determining the time of the next event requires outputs `direction` and `goal` from each UAV, where `goal` is the UAV's next anticipated event location (i.e., estimated perimeter endpoint or shared segment boundary). The System can

also view each of the UAV's outputs `PR`, `PL`, `NR`, `NL`. Note that we bound Integers `NR` and `NL` because in order to calculate estimated boundary segments, we must implement a lookup table that converts these to Real values. These values are used in the System guarantees (contained in Appendix B), which formalize Theorem 1, Lemma 1, and Theorem 2 of Section 2.

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 [4], 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 `NR` and `NL` 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],[5] but we outline the steps in Fig. 4. 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 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,

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

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 would happen if the perimeter suddenly shrinks and the UAVs must re-learn correct coordination variables.

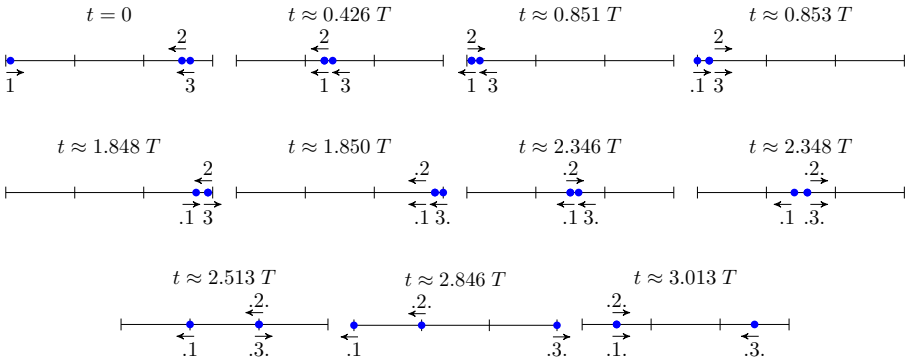

Fig. 4: Counterexample to Lemma 1. Dots to the left of a UAV number indicate it has correct "left" variables, and likewise for the right.

Analysis for three UAVs 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 extensive simulation results to support claims about its convergence bounds, analysis revealed that one of the key lemmas used in the proof was incorrect. Furthermore, the counterexample returned by AGREE provided insight into why this lemma 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 in the original paper only 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 in a middle UAV's line segment. Analysis showed that this can occur for as little as three UAVs in Algorithm B, and simulations in the original paper showed this happening frequently, but the assumed 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 groups, repeating the process for subgroups 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 that the model behaves as expected due to long analysis run times. Regardless, we are planning to formalize this behavior in AGREE. Second, we noted in Section 4 that analysis results assume UAV coordination variables `NR` and `NL` have an upper bound of 20. In fact, with an earlier upper bound of 3, we found the convergence time for Lemma 1 to be $(3 + \frac{1}{3})T$ and did not fully consider that it would depend on upper bounds for `NR` and `NL`. We can therefore not necessarily 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. We are therefore attempting to transition to theorem provers such as ACL2 [9] and PVS [12] to develop a proof of convergence bounds for DPSS for an arbitrary number of UAVs and an arbitrary upper bound on `NR` and `NL`.

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 technical 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. As an aside, we estimate the time to develop the AGREE model as approximately 7 man-weeks. About 50 hours were spent developing a model in a different tool before we realized that AGREE was more appropriate.

Acknowledgment

We would like to 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, architecture branch, `https://github.com/afrl-rq/OpenUxAS/tree/architecture`
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)

Appendix A

The following presents the original description and proofs of DPSS, building off of the terms and definitions introduced in Section 2. It has been slightly modified from Ref. [10] to explicitly label certain theorems and lemmas. There have also been some minor changes for brevity and clarity, but the original errors have been preserved. We include this because it is our experience that the style of the algorithm descriptions and proof strategies are fairly representative of those found in multi-agent system literature, and so we believe it has educational value to formal methods practitioners looking to work with multi-agent system designers.

if UAV i rendezvous with neighbor j **then**
  Calculate team size $N = N_{R_i} + N_{L_i} + 1$.
  Calculate perimeter length $P = P_{R_i} + P_{L_i}$.
  Calculate UAV i's relative index $n = N_{L_i} + 1$.
  Calculate UAV i's segment endpoints:
    $\mathcal{S}_i = \left\{ \lfloor n - \frac{1}{2}(-1)^n \rfloor P/N, \lfloor n + \frac{1}{2}(-1)^n \rfloor P/N \right\}$.
  Communicate $\mathcal{S}_i$ to neighbor j and receive $\mathcal{S}_j$.
  Calculate shared border position $p_{i,j} = \mathcal{S}_i \cap \mathcal{S}_j$.
  Travel with neighbor j to shared border position $p_{i,j}$.
  Set direction to monitor own segment.
else if reached perimeter endpoint **then**
  Reverse direction.
else
  Continue in current direction.
end if

Alg. A: Neighbor escort from the perspective of UAV i.

Algorithm A Consider the protocol described in Algorithm A, where each UAV escorts any neighboring UAV it encounters to the shared boundary of their perimeter segments. Note that UAVs do not automatically turn around if they reach a segment boundary; they only turn around if they encounter a *perimeter* endpoint or start or stop a neighbor escort. Meeting and escorting neighbors to segment boundaries is key for both obtaining correct coordination variables and achieving the synchronization required for the optimum configuration. Also note that each UAV $i \in 1, \ldots, N$ continuously updates P_{R_i} and P_{L_i} based on the distance it travels, and UAVs are able to tell when they reach a perimeter endpoint.

Theorem 1. *Let the perimeter length P and the N UAVs in the system be fixed. If every UAV's coordination variables are correct, then Algorithm A achieves the optimal configuration within $2T$.*

Proof. UAVs can initially be positioned anywhere along the perimeter and can be traveling either left or right at constant speed V. Each UAV has correct coordination variables and so can accurately calculate its perimeter segment. Also, each UAV is guaranteed to meet its neighbors since each UAV only reverses direction at a *perimeter* (not segment) endpoint or when starting or stopping a neighbor escort.

For N UAVs monitoring a perimeter of length P, order segments of size P/N from the left edge of the perimeter as $1, \ldots, N$, so that UAV i is responsible for segment i. Label the left endpoint of segment 1 as p_1, the right endpoint of segment N as p_N, and the shared endpoint of segments i and j as $p_{i,j}$. Let T_s be the time for a UAV to travel once across a segment, which is the same for every UAV.

Consider first the actions of UAV 1. Once UAV 1 has escorted UAV 2 to $p_{1,2}$, then no UAV to the right of UAV 1 will ever enter segment 1 by crossing $p_{1,2}$ again. This is because UAV 1 will make the round trip from $p_{1,2}$ to p_1 back to $p_{1,2}$ in $2T_s$, whereas UAV 2 would require at least $2T_s$ to return to $p_{1,2}$ after reaching or passing $p_{2,3}$. This is because there are two cases for UAV 2, as depicted in Figure 5. UAV 2 will either meet UAV 3 on segment 2, escort it to $p_{2,3}$, and turn around and reach $p_{1,2}$ in exactly $2T_s$, or UAV 2 will meet UAV 3 to the right of $p_{2,3}$, turn around to escort it to $p_{2,3}$, then continue on toward $p_{1,2}$, which would require more time than $2T_s$. UAV 2 will therefore either meet UAV 1 at $p_{1,2}$, or it will meet UAV 1 to the right of $p_{1,2}$, where UAV 1 will escort it to $p_{1,2}$ and then UAV 2 will reverse direction back towad $p_{2,3}$. In either case, $p_{1,2}$ can now be regarded as a fixed perimeter endpoint for UAV 2, and the same argument can be repeated (once UAV 1 has completed escorting UAV 2) if we consider UAV 2 as the leftmost UAV in a set of $N-1$ UAVs. Therefore, there is a time τ after which all UAVs are constrained to their respective segments, and since UAVs only change direction when meeting their neighbors at their shared segment boundaries or a perimeter endpoint, the optimal configuration in Definition 1 is reached.

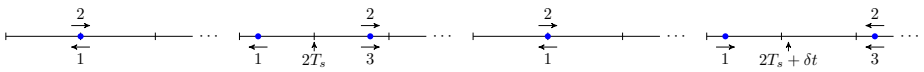

Fig. 5: Possible cases for the rendezvous between UAV 1 and UAV 2. They either meet in $2T_s$ at their shared segment boundary or to the right of it at $2T_s + \delta t$.

The worst case occurs when all UAVs are stacked at one end of the perimeter and are traveling toward the other end. Once T has passed, all UAVs are at the opposite end of the perimeter where they meet both of their neighbors. Each pair will travel to their shared borders, which for the farthest pair will require a travel time less than T. Therefore, the steady-state behavior will be achieved before time $2T$. □

1: **if** agent i (left) rendezvous with neighbor j (right) **then**
2: Update perimeter length and team size:
3: $P_{R_i} = P_{R_j}$
4: $N_{R_i} = N_{R_j} + 1$
5: Calculate team size $N = N_{R_i} + N_{L_i} + 1$.
6: Calculate perimeter length $P = P_{R_i} + P_{L_i}$.
7: Calculate relative index $n = N_{L_i} + 1$.
8: Calculate segment endpoints:
9: $\mathcal{S}_i = \left\{\lfloor n - \frac{1}{2}(-1)^n \rfloor P/N, \lfloor n + \frac{1}{2}(-1)^n \rfloor P/N\right\}$.*
10: Communicate $\mathcal{S}_i$ to neighbor j and receive $\mathcal{S}_j$.
11: Calculate shared border position $p_{i,j} = \mathcal{S}_i \cap \mathcal{S}_j$.
12: Travel with neighbor j to shared border $p_{i,j}$.
13: Set direction to monitor own segment.
14: **else if** reached left perimeter endpoint **then**
15: Reset perimeter length to the left $P_{L_i} = 0$.
16: Reset team size to the left $N_{L_i} = 0$.
17: Reverse direction.
18: **else if** reached right perimeter endpoint **then**
19: Reset perimeter length to the right $P_{R_i} = 0$.
20: Reset team size to the right $N_{R_i} = 0$.
21: Reverse direction.
22: **else**
23: Continue in current direction keeping track of traversed perimeter length.
24: **end if**

Alg. B: Neighbor escort with coordination variable update from the perspective of UAV i. UAV j follows the same protocol but updates P_{L_j} and N_{L_j} in lines 3-4 instead. *This equation is incorrect. It requires an extra term to account for the fact the the left perimeter endpoint may not be at 0 or its estimate may not be at 0. For instance, it could be rewritten as $\mathcal{S}_i = \left\{\lfloor n - \frac{1}{2}(-1)^n \rfloor P/N - \text{left\_endp\_est}, \lfloor n + \frac{1}{2}(-1)^n \rfloor P/N - \text{left\_endp\_est}\right\}$

Algorithm B Now consider Algorithm B, which is essentially Algorithm A with the additional steps of communicating and updating the coordination variables. We show that this protocol achieves the optimal configuration within $5T$ for arbitrary initial conditions of position, direction, and coordination variables for each UAV in the system. We first show that every UAV obtains correct coordination variables within $3T$, then by Theorem 1, the system takes an additional $2T$ to converge to the optimal configuration.

Lemma 1. *Using Algorithm B, every UAV will obtain correct coordination variables within* $3T$.

Proof. We first prove that all UAVs converge to correct coordination variables in finite time. Note that since a UAV only changes direction at perimeter endpoints or when starting or stopping a neighbor escort, all UAVs are guaranteed to meet their neighbors.

Label UAVs, segments, and endpoints as in Algorithm A and consider the actions of UAV 1. UAV 1 is guaranteed to visit p_1 either after an escort from UAV 2 or before having met any other UAVs, depending on initial conditions. Once UAV 1 has visited p_1, both N_{L_1} and P_{L_1} are correct. At the next meeting of UAV 1 and 2, UAV 2 updates its "left" coordination variables N_{L_2} and P_{L_2} through communication with UAV 1, thereby obtaining correct values for them. Note that repeated meetings between UAV 1 and 2 will not change the value of these coordination variables since N and P are fixed. Now consider UAV 2 as the leftmost UAV in a team of $N-1$ UAVs, and note that UAV 3 is similarly ensured to obtain correct left coordination variables. Since only one neighbor meeting is required after the leftmost UAV has obtained correct values for its left coordination variables, the number of UAVs needing correct "left" coordination variables is reduced at each stage, and since meetings are guaranteed to occur in finite time, every UAV in the system obtains correct values for its left coordination variables in finite time. Clearly, the same argument holds for the right end of the perimeter and right coordination variables.

During the transient period when the UAVs are learning the correct coordination variables, the calculation of shared segment boundaries is incorrect relative to the optimal configuration, but it is consistent between UAVs involved in the rendezvous and escort. This can be seen by noting that after both UAVs have communicated and updated their coordination variables with each other, they each have the same understanding of P and N and so calculate the same shared segment boundary position. So while they escort each other to the (ultimately) wrong position, they are still guaranteed to continue in the correct direction afterward to ensure that each UAV meets both its neighbors.

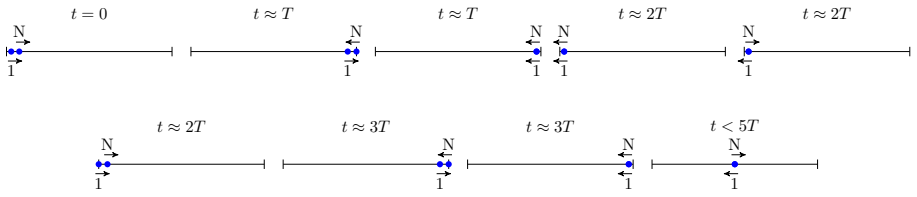

Fig. 6: Worst case convergence for Algorithm B demonstrated with 2 UAVs.

The worst case for obtaining correct coordination variables occurs when the overlap of the UAVs is the greatest, that is, when UAVs travel together rather than space out along the perimeter. Consider N UAVs stacked near $x = 0$ and headed right (Figure 6a). After time T has passed, UAV N will reach the right perimeter endpoint and update P_{R_N} and L_{R_N} to correct values (Figure 6b). Once UAV N has visited p_N, it will rendezvous with the rest of the UAVs in the system (Figure 6c). At this instant, the UAVs have arbitrary values for their "left" coordination variables. Suppose the estimated shared border position for

UAVs N and $N-1$ is at $x = \epsilon$ for $\epsilon << P$ and is less than that for all other UAVs. In this case, the entire team will again nearly travel the length of the perimeter to $x = \epsilon$ which requires less than another T units of time (Figure 6d). UAV N then separates from the team and begins heading toward the right end of the perimeter without learning about the left perimeter endpoint (Figure 6e). UAVs $1 \ldots N-1$ encounter the left endpoint of the perimeter and update to a completely correct set of coordination variables (Figure 6f). With correct coordination variables, UAVs begin to space out equally along the perimeter. Note that UAV $N-1$ will chase UAV N the entire length of the perimeter since it will escort UAV $N-2$ to their shared border position and then continue to the right. UAV $N-1$ then meets UAV N near the right end of the perimeter in T units of time since reaching the left endpoint (Figure 6g). Once they meet, UAV N will update to correct coordination variables (Figure 6h). At this point, $3T$ has passed. □

Theorem 2. *Let the perimeter length P and the N UAVs in the system be fixed. Then an upper bound for convergence of Algorithm B to the optimal configuration is $5T$.*

Proof. By Lemma 1, an upper bound on the amount of time for all UAVs to obtain correct coordination variables is $3T$. Then by Theorem 1, up to an additional $2T$ is needed for the UAVs to reach the optimal configuration. Therefore, an upper bound for the convergence of Algorithm B is $5T$. □

Appendix B

Here, we provide a more thorough overview of the AGREE/AADL models for Algorithm B, including guarantees that correspond to Lemma 1 and Theorem 2. However, the models are still too large to fully cover in this appendix. The full models can be found on GitHub [1][6].

The model consists of a top-level system model and a component-level UAV model that is instantiated multiple times. We henceforth refer to these simply as the "system" and the "UAV(s)." The system sets valid ranges for the initial conditions for each UAV through a set of top-level assumptions. It also essentially runs a discrete event simulation of the UAVs as they follow the DPSS protocol. Events include a UAV reaching a perimeter endpoint or two UAVs starting or stopping an escort. At initialization and after each event, the system uses the globally known constant UAV speed `V` and direction information from each UAV to determine the amount of time `deltaT` until the next event, and it uses this to update the positions of the UAVs. The UAV essentially uses a component-level assume/guarantee contract to model the behavior of a UAV, which includes updating the values of coordination variables when neighbors meet and changing direction at a perimeter endpoint or when starting or stopping an escort. The UAVs assist the system by computing each UAV's next "goal" location, i.e. estimated perimeter endpoint or shared segment boundary, and the system determines `deltaT` by finding the minimum time until the next event across all UAVs. Note that the system handles the case that a UAV's goal is an estimated perimeter endpoint that falls short of the actual perimeter endpoint.

```
const V : real = 1.0; --UAV velocity
const LEFT : int = -1;
const RIGHT : int = 1;
```

Listing 1.1: Constants file for Algorithm B for 3 vehicles.

Global constants accessible to both the system and UAVs are shown in Listing 1.1. These define the UAV speed `V` as a real value of `1.0` and enumerate directions `LEFT` and `RIGHT` as integer values of `-1` and `1`. Note that all AGREE variables have types. Ports have AADL `Base_Types` of `Integer`, `Float`, and `Boolean`, whose values are converted to `int`, `real`, and `bool` for reasoning in AGREE.

UAV Model Start by considering the UAV model. Its inputs and outputs are shown in Listing 1.2. Values for UAV inputs are passed in by the system and

[6] In particular, the AADL projects for this work are located under AADL_sandbox_projects and have names starting with DPSS. The Algorithm A models for three vehicles are in the DPSS-3-AlgA-for-paper project. The Algorithm B models for three vehicles are in the DPSS-3-AlgB-for-paper project.

```
--Initial direction and position, set by system and
--essentially constrained by system-level assumptions
initial_direction : in data port Base_Types::Integer;
initial_position : in data port Base_Types::Float;
--This UAV's position, updated by system after an event
pos : in data port Base_Types::Float;
--Set by system when right/left neighbor is co-located
meet_RN : in data port Base_Types::Boolean;
meet_LN : in data port Base_Types::Boolean;
--PR and NR of right neighbor, PL and NL of left neighbor, passed in by system
PR_RN : in data port Base_Types::Float;
PL_LN : in data port Base_Types::Float;
NR_RN : in data port Base_Types::Integer;
NL_LN : in data port Base_Types::Integer;
--Actual right and left perimeter endpoints
right_endp_truth : in data port Base_Types::Float;
left_endp_truth : in data port Base_Types::Float;

-- This UAV's current direction and goal
direction: out data port Base_Types::Integer;
goal : out data port Base_Types::Float;
-- This UAV's coordination variables
PR : out data port Base_Types::Float;
PL : out data port Base_Types::Float;
NR : out data port Base_Types::Integer;
NL : out data port Base_Types::Integer;
```

Listing 1.2: Inputs and outputs for the Algorithm B UAV model.

include the UAV's initial direction, initial position, and updated position after every event. As discussed in Section 5, system assumptions provide bounds on UAV initial positions as a function of perimeter length and the initial position of other UAVs. They also allow the initial direction to be either left or right. Input values from the system also include information on whether this UAV is co-located with its right or left neighbor and the values of the coordination variables of neighbors (unused if there is no left or right neighbor). Finally, they also include the true values of the left and right perimeter endpoints, which the UAV only uses to check if it is at a perimeter endpoint. UAV outputs include the UAV's direction, its next goal location, and its current coordination variables. These are sent back to the system so it can determine the time of the next event and pass the values of coordination variables between neighbors.

UAV behavior is essentially captured by the UAV's assume/guarantee contract. UAV assumptions include that the value of the right perimeter endpoint is greater than the left and that perimeter endpoints are fixed. Assumptions for the right perimeter endpoint are shown in Listing 1.3. Assumptions for the left perimeter endpoint are analogous. Note that a statement of the form `x -> y` is not a logical implication; rather, it means use `x` on the initial timestep, and `y` on all subsequent timesteps. This is often used in conjunction with the `pre` operator, where `pre(x)` returns the value of variable `x` on the previous timestep, which is not defined on the initial timestep. On the initial timestep, `pre(x)` can return any valid value given `x`'s type.

UAV guarantees specify the values of the UAV's `goal`, `direction`, `PR`, `PL`, `NR`, and `NL` based on the inputs and assumptions. These are shown in Listing 1.4, with some omitted for brevity. Equations defined by the `eq` keyword are used to make

```
assume "right_endp_truth is greater than left_endp_truth":
  right_endp_truth > left_endp_truth;

assume "right_endp_truth is fixed":
  true -> (right_endp_truth = pre(right_endp_truth));
```

Listing 1.3: Assumptions for the Algorithm B UAV model.

it easier to compute the UAV's estimated segment boundaries. Equations with the `prev` operator are used to store the UAV's previous direction and position, where `prev(x,y)` means use the value of `y` on the initial timestep and the value of `pre(x)` on all other timesteps.

System Model Now consider the top-level system model. It computes the time `deltaT` until the next event using an equation. It also updates the positions of the UAVs, flags when UAVs are co-located, and passes the values of coordination variables between neighboring UAVs. It sends this information to the UAVs through their input data ports. It should be noted that assumptions of the system define valid ranges for initial UAV positions and directions. An assumption is also used to update the cumulative running `time`. Assumptions for the system are shown in Listing 1.5.

The system also receives information from the UAVs through their output data ports, including the values of their coordination variables. It stores these and the current and previous values of the some of the inputs it sends to the UAVs, including UAV positions. It uses these to define an equation that checks whether all the UAVs' coordination variables are correct. It also defines an equation that checks whether the optimal configuration has been reached. These equations are shown in Listing 1.6 and are used in the system guarantees. The system guarantees correspond to the theorems of Section 2 and are discussed in the next section.

System Guarantees The AGREE guarantees corresponding to Theorem 2 and Lemma 1 are shown in Listing 1.7. Note that our formal definition of `optimal` in Listing 1.6 actually includes one extra synchronous UAV cycle of time $\frac{1}{N}$, so we subtract `1.0/N_TRUTH_REAL` from `5.0*N_TRUTH_REAL` in the guarantee corresponding to Theorem 2. This is because the UAVs need to have been at their shared boundary segments on the previous time step as well as the current time step, the length of which should be $\frac{1}{N}$. We therefore also include a sanity check that `deltaT = T/N_TRUTH_REAL` in the guarantee. Note that a user can experiment with the bounds, and a counterexample will be returned if a bound is too low. By experimenting with the bound in the last lemma, we are able to find a counterexample up to $(3 + \frac{1}{2}T)$, but not for $(3 + \frac{2}{3}T)$.

```
--UAV's estimate of id n_id, N, P, and perimeter endpoints
eq n_id : int = NL + 1;
eq N : int = NL + NR + 1;
eq P : real = PR + PL;
eq left_endp_est : real = pos - PL;
eq right_endp_est : real = pos + PR;

--Shared border positions where n_id_real and N_real
--are N and n_id as real types returned by a lookup table
eq S_L : real = (n_id_real -1.0)*P/N_real + left_endp_est;
eq S_R : real = n_id_real*P/N_real + left_endp_est;

--Previous direction
eq pre_direction : int =
        prev(direction, initial_direction);

--Previous PR; initially can be any real value
eq pre_PR : real = pre(PR)

--Booleans to check whether an endpoint has been reached
eq reach_right_endp_truth : bool=(pos>=right_endp_truth);
eq reach_left_endp_truth : bool=(pos<=left_endp_truth);

--max_real returns the max of its arguments
guarantee "PR formula":
  PR = if meet_RN then PR_RN
       else if reach_right_endp_truth then 0.0
       else if pre_direction = DPSS_Constants.RIGHT then
            DPSS_Node_Lib.max_real(0.0, pre_PR - (pos - pre_pos))
       else DPSS_Node_Lib.max_real(0.0, pre_PR + (pos - pre_pos));

guarantee "NR formula":
  NR = if meet_RN then NR_RN + 1
       else if reach_right_endp_truth then 0
       else pre_NR;

guarantee "PL formula":
guarantee "NL formula":
  --Omitted for brevity; analogous to above

guarantee "Direction formula":
  direction = (
    --Turn around at the boundaries
    if reach_left_endp_truth then DPSS_Constants.RIGHT
    else if reach_right_endp_truth then DPSS_Constants.LEFT
    --If meeting a neighbor, travel to shared border
    else if meet_LN then
      if pos <= S_L then DPSS_Constants.RIGHT
      else DPSS_Constants.LEFT
    else if meet_RN then
      if pos < S_R then DPSS_Constants.RIGHT
      else DPSS_Constants.LEFT
    --In all other cases, proceed in the same direction
    else
      pre_direction
  );

guarantee "Goal formula":
  --Omitted for brevity. Sets goal to either S_R, S_L,
  --right_endp_est, or left_endp_est
```

Listing 1.4: Guarantees for the Algorithm B UAV model.

```
assume "time update": time = (0.0 -> pre(time) + deltaT);

assume "UAVs are numbered according to their position from
        left to right and do no start co-located":
(initial_pos_UAV1 < initial_pos_UAV2 and
 initial_pos_UAV2 < initial_pos_UAV3) -> true;

assume "Initial positions are between perimeter endpoints":
(initial_pos_UAV1 >= LEFT_ENDP_TRUTH and
 initial_pos_UAV2 >= LEFT_ENDP_TRUTH and
  ...
 initial_pos_UAV2 <= RIGHT_ENDP_TRUTH and
 initial_pos_UAV3 <= RIGHT_ENDP_TRUTH) -> true;

assume "Initial directions are LEFT or RIGHT":
((initial_direction_UAV1 = DPSS_Constants.LEFT or
  ...
  initial_direction_UAV3 = DPSS_Constants.RIGHT)) -> true;
```

Listing 1.5: Assumptions for the Algorithm B system model.

```
--Shared border positions
eq P_12 : real =
  1.0*P_TRUTH/N_TRUTH_REAL + LEFT_ENDP_TRUTH;
eq P_23 : real =
  2.0*P_TRUTH/N_TRUTH_REAL + LEFT_ENDP_TRUTH;

--Coordination variables are correct
--Values from UAV models obtained through data ports
eq correct_coordination_variables : bool =
  NL_UAV1 = 0 and NL_UAV2 = 1 and NL_UAV3 = 2 and
  NR_UAV1 = 2 and NR_UAV2 = 1 and NR_UAV3 = 0 and
  PL_UAV1 = pos_UAV1 - LEFT_ENDP_TRUTH and
  ...
  PR_UAV3 = RIGHT_ENDP_TRUTH - pos_UAV3;

eq optimal : bool = correct_coordination_variables and
  --Either the UAVs are at LEFT_ENDP_TRUTH, P_23, and P_23
  --and were previously at P_12, P_12, and RIGHT_ENDP_TRUTH
  (
    (pos_UAV1 = LEFT_ENDP_TRUTH  and
     pos_UAV2 = P_23 and pos_UAV3 = P_23)
    and
    (pre_pos_UAV1 = P_12 and pre_pos_UAV2 = P_12 and
     pre_pos_UAV3 = RIGHT_ENDP_TRUTH)
  )
  --or the UAVs are at P_12, P_12, and RIGHT_ENDP_TRUTH
  --and were previously at LEFT_ENDP_TRUTH, P_23, and P_23
  or (
    (pos_UAV1 = P_12 and pos_UAV2 = P_12 and
     pos_UAV3 = RIGHT_ENDP_TRUTH)
    and
    (pre_pos_UAV1 = LEFT_ENDP_TRUTH and
     pre_pos_UAV2 = P_23 and
     pre_pos_UAV3 = P_23)
  )
);
```

Listing 1.6: Key equations for the Algorithm B system model.

```
--Main Theorem
guarantee "Theorem 2 for Algorithm B for 3 Vehicles":
  (time >= ((5.0*N_TRUTH_REAL - 1.0/N_TRUTH_REAL)*T) =>
    (optimal and deltaT = T/N_TRUTH_REAL);

lemma "Time to optimal configuration is less than 4T":
  (optimal and not (pre(optimal))) => (time < 4.0*T);

lemma "Time to correct coord. variables is < (3 + 1/2)T":
  (correct_coordination_variables and
    not (pre(correct_coordination_variables))) =>
  (time < (3.0 + 1.0/4.0)*T);
```

Listing 1.7: Main Theorem and Lemmas for Alg. B for 3 UAVs