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][8].

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

[8] 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