An approach to formal verification of free-flight separation

Sebastian Eder
Institute of Computer Science
University of Augsburg
Germany
seb.eder@googlemail.com

Graeme Smith
School of Information Technology and Electrical Engineering
The University of Queensland
Australia
smith@itee.uq.edu.au

***Abstract*—This paper presents an approach to verifying complex free-flight algorithms. We give an abstract model that defines properties that a concrete implementation of a (distributed) free-flight algorithm has to maintain to guarantee conflict free movement of airplanes. We develop this model gradually by defining the emergent behavior of airplanes at a very abstract level and refine our definitions towards a more concrete model. In this process, we prove every refinement step to guarantee correctness of our approach.**

***Keywords*-free-flight separation, distributed systems, formal methods, refinement**

I. Introduction

In current air traffic control systems, airplanes are monitored by human controllers. To prevent collisions, the controllers advise new routes to the airplanes. The controllers rely on the spacing between established airways to maintain collision-free air traffic. But this system has its flaws: inefficient use of airspace, probability of human errors during this process and non-optimal routes [6]. To solve those problems, the concept of free-flight separation assurance was developed [1]. This approach allows airplanes or their pilots to choose their route on their own.

Algorithms for calculating routes for airplanes in a free-flight setting have been developed over the past years, for example in [6], [3], [5], [7]. In these papers the only tool that was used to verify correctness for an arbitrary number of airplanes is simulation. Like testing, while simulation can raise confidence in an algorithm, it cannot provide guarantees. As lives are at risk when using computer-aided collision avoidance for airplanes, much trust must be placed in such algorithms. Hence, we propose a formal approach to their verification.

We adopt the refinement-based approach of Smith and Sanders [9] for formally verifying that a self-organising system satisfies a global (emergent) behaviour. The idea is to start with an abstract specification in which the global behaviour is explicitly captured by a single event, and then to progressively refine that event towards a sequence of interactions of the system components. By definition, each step in the refinement process preserves the behaviour of the original abstract specification [2]. Hence, any implementation satisfying the interaction patterns of the final specification is guaranteed to exhibit the global behaviour.

The specification language we use in this paper is Z [10]. We introduce the proof obligations for refinement in Z in Section II. In Section III, we introduce a well established algorithm for free-flight separation by Eby [3]. In Section IV, we provide a series of refinements of an abstract model of free-flight separation. We show that Eby's algorithm does not refine the abstract model and cannot be guaranteed to maintain a minimum separation distance between airplanes. We also provide a modification of Eby's approach which can be proven to maintain separation. While we don't provide the implementation details of this approach, any implementation satisfying the specification will result in the desired global behaviour.

Other work in this area has looked at formally verifying free-flight separation algorithms at the implementation level [8], [4]. This has been limited, however, to scenarios involving only two airplanes. Our approach provides verification of a more abstract model but does so for an arbitrary number of airplanes.

II. Refinement

We use the Z notation [10] for our specifications in this paper. In the Z notation a concrete operation C is a refinement of an abstract operation A (denoted by $A \sqsubseteq C$) if the following conditions apply:

1) The precondition of C is equal to or more liberal than the precondition of A: $\mathrm{pre}\, A \Rightarrow \mathrm{pre}\, C$
2) The postcondition of C is equal to or stronger than the postcondition of A: $\mathrm{pre}\, A \wedge C \Rightarrow A$

This results in C being applicable whenever A is. Furthermore, C leads a system into a state of the state space that a system can take after an execution of A. Since nondeterminism may be reduced, the resulting state space of C may be smaller than the resulting state space of A.

III. Implementations of free-flight algorithms

Eby [3] presents a free-flight separation approach based on treating airplanes as if they were charged particles. An airplane wields a repulsive force on other airplanes proportional to its distance from them. Applying this force in all cases has unwanted effects, so Eby only applies the force to airplanes which have detected a future conflict.

This approach has been used by other researchers in the area including Hoekstra [6]. It consists of four steps which are continually repeated.

1) Detect conflicts
2) Calculate routes to avoid future conflicts
3) Fly along new routes for a short time
4) Restore routes to a straight line to the destination

The routes calculated in the second step are based on summing up the repulsive forces of conflicting airplanes. For a conflict involving only two airplanes, this force causes the airplanes to fly routes which are tangential to the desired safety radius around each airplane. This minimises the deviation from the original route. While this approach has been proven to work for two airplanes [4], it has only been simulated for more than two. The simulations show that the separation of airplanes by at least the safety radius is not always maintained [3].

IV. Formal models

A. Abstract view and emergence

We begin by introducing the abstract types *AIRPLANE* and *POSITION*.

$$[AIRPLANE, POSITION]$$

The airplanes in a particular airspace are denoted by the set *airplanes* and we define partial functions that return the origin (*orig*) and the destination (*dest*) of airplanes. The function d calculates the distance between two positions. r_d is the radius around the destination of an airplane where we consider the airplane to be at its destination. This radius would generally be several kilometers around an airport, and corresponds to where an air-traffic controller would take over the separation assurance.

In current requirements for safe air traffic, all airplanes must be separated by a certain distance. In our system, we define this safety zone as a circle around an airplane of radius r_s, which is defined to be greater than zero.

$$\begin{array}{l} orig : AIRPLANE \pfun POSITION \\ dest : AIRPLANE \pfun POSITION \\ airplanes : \mathbb{P}\, AIRPLANE \\ r_s, r_d : \mathbb{N} \\ d : POSITION \times POSITION \to \mathbb{N} \\ \hline r_d > 0 \land r_s > 0 \\ \mathrm{dom}\, orig = \mathrm{dom}\, dest = airplanes \end{array}$$

On this level of abstraction only positions of airplanes change. To describe the current position of an airplane, we define the function *pos*, which returns this position. The domain of *pos* equals the domain of *orig* and *dest*, which is the set *airplanes*. This function is defined in the state schema *Airspace* so that its value can be changed by operations. In Z, the values of axiomatically defined variables (such as those above) cannot change.

Airspace

$$pos : airplanes \to POSITION$$

Initially, all airplanes are located at their origins. So we define the schema *INIT*1 that models this condition. *INIT*1 states that the function *pos* returns exactly the values that *orig* returns.

*INIT*1

$$\begin{array}{l} Airspace \\ \hline pos = orig \end{array}$$

The global (emergent) property of an airspace with independently maneuvering airplanes is that, given all airplanes do not start within the safety zone of another airplane, every airplane reaches its destination. This is defined by *Event*1, which will be refined throughout this paper. After *Event*1 is executed, each airplane is at its destination, or at maximum r_d away from it. More precisely, the position of an airplane a after the execution of *Event*1 must be less than or equal to r_d from its destination. Note that the value of variable *pos* after the execution of *Event*1 is denoted by pos'. It is introduced into the operation schema along with *pos* by the notation $\Delta Airspace$.

*Event*1

$$\begin{array}{l} \Delta Airspace \\ \hline \forall\, a, a_c : airplanes \bullet a \neq a_c \Rightarrow d(pos(a), pos(a_c)) \geq r_s \\ \forall\, a : airplanes \bullet d(pos'(a), dest(a)) \leq r_d \end{array}$$

B. Stepwise movement

Now we refine our abstract schema *Airspace* to consider conflicts of airplanes by introducing a relation *in_conflict*. A conflict arises, if the distance between two airplanes is smaller than the safety radius. There is no conflict if both airplanes are within r_d of their destinations, because then an air traffic controller assures conflict-free movement.

AirspaceSteps

$$\begin{array}{l} Airspace \\ in\_conflict : airplanes \leftrightarrow airplanes \\ \hline \forall\, a, a_c : airplanes \bullet (a, a_c) \in in\_conflict \Leftrightarrow \\ \qquad a \neq a_c \land d(pos(a), pos(a_c)) < r_s \land \\ \qquad (d(pos(a), dest(a)) > r_d \lor d(pos(a_c), dest(a_c)) > r_d) \end{array}$$

This airspace, and that in all subsequent refinements, is initialised as in *INIT*1.

We define a new schema *Event*2 that guarantees that there are no conflicts while moving airplanes stepwise. Furthermore, *Event*2 assures that the distance to the destination of every airplane after every execution is smaller than the distance before. This assures progress of the system by bringing each airplane closer to its destination.

*Event*2

Δ*AirspaceSteps*

$in\_conflict = \varnothing$

$\forall a : airplanes \bullet d(pos'(a), dest(a)) < d(pos(a), dest(a)) \vee$
$\quad d(pos(a), dest(a)) \leq r_d \wedge d(pos'(a), dest(a)) \leq r_d$

$in\_conflict' = \varnothing$

If *Event*2 repeatedly occurs every airplane will reach its destination. Therefore, while *Event*2 does not refine *Event*1, a finite number of repetitions of *Event*2 does. That is, for some $n : \mathbb{N}$, $Event1 \sqsubseteq Event2^n$.

To prove that we have a valid refinement, we need to prove the two conditions of Section II.

Theorem 1: $\exists n : \mathbb{N} \bullet Event1 \sqsubseteq Event2^n$

Proof: (1) pre *Event*1 $\Rightarrow$ pre *Event*2

pre *Event*1
$\Leftrightarrow \forall a, a_c : airplanes \bullet a \neq a_c \Rightarrow d(pos(a), pos(a_c)) \geq r_s$
$\Leftrightarrow \nexists a, a_c : airplanes \bullet a \neq a_c \wedge d(pos(a), pos(a_c)) < r_s$
$\Rightarrow \nexists a, a_c : airplanes \bullet a \neq a_c \wedge d(pos(a), pos(a_c)) < r_s \wedge$
$\quad (d(pos(a), dest(a)) > r_d \vee d(pos(a_c), dest(a_c)) > r_d)$
$\Leftrightarrow in\_conflict = \varnothing$
$\Leftrightarrow$ pre *Event*2

(2) For all $a : airplanes$ such that $d(pos(a), dest(a)) > r_d$,

$Event2 \Rightarrow d(pos'(a), dest(a)) < d(pos(a), dest(a))$
$\quad \Rightarrow d(pos'(a), dest(a)) + 1 \leq d(pos(a), dest(a))$

Therefore,

$Event2^n \Rightarrow d(pos'(a), dest(a)) + n \leq d(pos(a), dest(a))$

Hence, after $n = d(pos(a), dest(a)) - r_d$ executions of *Event*2, $d(pos'(a), dest(a)) \leq r_d$. Therefore for some n, $Event2^n$ will terminate with $d(pos(a), dest(a)) \leq r_d$ for all a in airplanes. ■

C. Routes

To reduce the level of abstraction further, we define routes for airplanes.

$ROUTE == \text{seq}\, POSITION$

We define a maximum velocity v_{max} for airplanes and assume planes move for one time unit each step. Hence, v_{max} is the maximum distance an airplane can travel in one step. To avoid airplanes 'jumping over' each other, r_s has to be (much) greater than v_{max}.

$v_{max} : \mathbb{N}$

$0 < v_{max} < r_s$

Now we can give a further refinement of the airspace by extending *AirspaceSteps*. We constrain the position of an airplane a to be at the beginning of its route ($route(a)(1)$) while the end of the route of the airplane ($route(a)(\#route(a))$) is at its destination. Furthermore, a route must lead an airplane towards its destination by reducing the distance to it after every step. Finally, routes must not require airplanes to fly faster than the maximum speed.

AirspaceRoutes

AirspaceSteps
$route : airplanes \rightarrow ROUTE$

$\forall a : airplanes \bullet pos(a) = route(a)(1) \wedge$
$\quad dest(a) = route(a)(\#route(a)) \wedge$
$\quad \forall i : \text{dom}(route(a)) \setminus \{\#(route(a))\} \bullet$
$\quad\quad d(route(a)(i+1), dest(a)) <$
$\quad\quad\quad d(route(a)(i), dest(a)) \wedge$
$\quad\quad d(route(a)(i), route(a)(i+1)) < v_{max}$

No additional constraints are required to initialise routes. The invariant of *AirspaceRoutes* already assures that a route extends from the current position of an airplane to its destination.

To avoid conflicts, airplanes have to change their routes. The operation *ChangeRoutes* achieves this. Here we only guarantee that the next waypoint an airplane flies to is not in conflict with another airplane, since the resolution of a conflict may require many steps. The invariant that the distance to the destination decreases at each step rules out inappropriate algorithms where an infinite number of steps can be necessary to resolve a conflict [5]. *ChangeRoutes* does not need to consider airplanes that are already in conflict because our operations do not allow them to get into conflict as proved below.

ChangeRoutes

Δ*AirspaceRoutes*

$\forall a, a_c : airplanes \setminus \text{dom}\, in\_conflict \bullet (a \neq a_c \wedge$
$\quad (d(pos(a), dest(a)) > r_d \vee$
$\quad\ d(pos(a_c), dest(a_c)) > r_d)) \Rightarrow$
$\quad\quad d(route'(a)(2), route'(a_c)(2)) \geq r_s$

$pos' = pos$

MoveAlongRoutes moves airplanes to their next position in their route and sets the new route to the tail of the original route. If an airplane has already reached the zone around its destination defined by r_d, we only require that it remains within that zone.

MoveAlongRoutes

Δ*AirspaceRoutes*

$\forall a : airplanes \bullet$
$\quad pos'(a) = route(a)(2) \wedge route'(a) = tail(route(a)) \vee$
$\quad d(pos(a), dest(a)) \leq r_d \wedge d(pos'(a), dest(a)) \leq r_d$

The sequential composition of the operations *ChangeRoutes* and *MoveAlongRoutes* will move each airplane towards its destination without conflicts occurring. Hence, we have a refinement of *Event*2.

Theorem 2: $Event2 \sqsubseteq ChangeRoutes \fatsemi MoveAlongRoutes$

Proof: (1) The precondition has been weakened to *true*, so the proof for the preconditions is trivial.

(2) The invariant of the schema *AirspaceRoutes* assures that the route of each a : *airplanes* lead the airplane towards its destination:

$$
\begin{array}{l}
\mathit{AirspaceRoutes} \\
\Rightarrow d(\mathit{route}(a)(2), \mathit{dest}(a)) < d(\mathit{route}(a)(1), \mathit{dest}(a))
\end{array}
$$

Hence, *MoveAlongRoutes* moves airplanes towards their destinations:

$$
\begin{array}{l}
\mathit{MoveAlongRoutes} \\
\Rightarrow \mathit{pos}'(a) = \mathit{route}(a)(2) \lor \\
\quad d(\mathit{pos}(a), \mathit{dest}(a)) \leq r_d \land d(\mathit{pos}'(a), \mathit{dest}(a)) \leq r_d \\
\Rightarrow d(\mathit{pos}'(a), \mathit{dest}(a)) < d(\mathit{pos}(a), \mathit{dest}(a)) \lor \\
\quad d(\mathit{pos}(a), \mathit{dest}(a)) \leq r_d \land d(\mathit{pos}'(a), \mathit{dest}(a)) \leq r_d
\end{array}
$$

Also when there are no existing conflicts, *ChangeRoutes* ensures the next position in the route is conflict-free and does not change the position of any airplane:

$$
\begin{array}{l}
\mathrm{pre}\, \mathit{Event}2 \land \mathit{ChangeRoutes} \\
\Rightarrow \mathit{in\_conflict} = \varnothing \land \\
\quad (\forall a, a_c : \mathit{airplanes} \setminus \mathrm{dom}\, \mathit{in\_conflict} \bullet (a \neq a_c \land \\
\qquad (d(\mathit{pos}(a), \mathit{dest}(a)) > r_d \lor \\
\qquad\ d(\mathit{pos}(a_c), \mathit{dest}(a_c)) > r_d)) \Rightarrow \\
\qquad\qquad d(\mathit{route}'(a)(2), \mathit{route}'(a_c)(2)) \geq r_s) \land \\
\quad \mathit{pos}' = \mathit{pos} \\
\Rightarrow (\forall a, a_c : \mathit{airplanes} \bullet (a \neq a_c \land \\
\qquad (d(\mathit{pos}(a), \mathit{dest}(a)) > r_d \lor \\
\qquad\ d(\mathit{pos}(a_c), \mathit{dest}(a_c)) > r_d)) \Rightarrow \\
\qquad\qquad d(\mathit{route}'(a)(2), \mathit{route}'(a_c)(2)) \geq r_s) \land \\
\quad \mathit{pos}' = \mathit{pos}
\end{array}
$$

and

$$
\begin{array}{l}
\mathit{MoveAlongRoutes} \\
\Rightarrow \mathit{pos}'(a) = \mathit{route}(a)(2)
\end{array}
$$

Hence,

$$
\begin{array}{l}
\mathrm{pre}\, \mathit{Event}2 \land \mathit{ChangeRoutes} \fatsemi \mathit{MoveAlongRoutes} \\
\Rightarrow \forall a, a_c : \mathit{airplanes} \bullet (a \neq a_c \land \\
\qquad (d(\mathit{pos}(a), \mathit{dest}(a)) > r_d \lor \\
\qquad\qquad d(\mathit{pos}(a_c), \mathit{dest}(a_c)) > r_d)) \Rightarrow \\
\qquad\qquad d(\mathit{pos}'(a)(2), \mathit{pos}'(a_c)(2)) \geq r_s \\
\Rightarrow \nexists a, a_c : \mathit{airplanes} \bullet a \neq a_c \land \\
\qquad d(\mathit{pos}'(a)(2), \mathit{pos}'(a_c)(2)) < r_s \land \\
\qquad (d(\mathit{pos}(a), \mathit{dest}(a)) > r_d \lor \\
\qquad\ d(\mathit{pos}(a_c), \mathit{dest}(a_c)) > r_d) \\
\Rightarrow \mathit{in\_conflict}' = \varnothing
\end{array}
$$

That is,

$$
\begin{array}{l}
\mathrm{pre}\, \mathit{Event}2 \land \mathit{ChangeRoutes} \fatsemi \mathit{MoveAlongRoutes} \\
\Rightarrow \mathit{Event}2
\end{array}
$$

■

D. Eby's algorithm

A crucial part of Eby's approach is the detection of conflicts. We therefore define a relation between airplanes that captures whether they will enter into a conflict. We extend *AirspaceRoutes* by adding this relation.

AirspaceEby

$$
\begin{array}{l}
\mathit{AirspaceRoutes} \\
\mathit{conflict\_detected} : \mathit{airplanes} \leftrightarrow \mathit{airplanes}
\end{array}
$$

In order to detect conflicts, airplanes look ahead by a certain amount of time. An airplane must not move further in one step than the radius defined by its look ahead time.

$$
\begin{array}{|l}
t_l : \mathbb{N} \\
\hline
0 < t_l
\end{array}
$$

We can now specify an operation for detecting conflicts.

DetectConflicts

$$
\begin{array}{l}
\Delta \mathit{AirspaceEby} \\
\hline
\mathit{conflict\_detected}' = \{a, a_c : \mathit{airplanes} \mid a \neq a_c \land \\
\qquad (\exists t_c : \mathbb{N} \bullet 0 \leq t_c \leq t_l \land \\
\qquad\qquad d(\mathit{route}(a)(t_c), \mathit{route}(a_c)(t_c)) < r_s \land \\
\qquad (d(\mathit{pos}(a), \mathit{dest}(a)) > r_d \lor \\
\qquad\ d(\mathit{pos}(a_c), \mathit{dest}(a_c)) > r_d))\} \\
\mathit{pos}' = \mathit{pos} \land \mathit{route}' = \mathit{route}
\end{array}
$$

To model the algorithm of Eby [3], we define the function *restore* which abstractly models a function that calculates a route on a straight line between two positions.

$$
\begin{array}{|l}
\mathit{restore} : \mathit{POSITION} \times \mathit{POSITION} \rightarrow \mathit{ROUTE}
\end{array}
$$

We now define an operation that applies this function to the routes of all airplanes while not changing anything else.

Restore

$$
\begin{array}{l}
\Delta \mathit{AirspaceEby} \\
\hline
\forall a : \mathit{airplanes} \bullet \mathit{route}'(a) = \mathit{restore}(\mathit{pos}(a), \mathit{dest}(a)) \\
\mathit{pos}' = \mathit{pos} \land \mathit{conflict\_detected}' = \mathit{conflict\_detected}
\end{array}
$$

A further operation calculates avoidance routes for airplanes which have detected a conflict.

Avoid

$$
\begin{array}{l}
\Delta \mathit{AirspaceEby} \\
\hline
\forall a, a_c : \mathit{airplanes} \bullet \\
\qquad (a, a_c) \in \mathit{conflict\_detected} \Rightarrow \\
\qquad\qquad d(\mathit{route}'(a)(2), \mathit{route}'(a_c)(2)) \geq r_s \\
\forall a : \mathit{airplanes} \setminus \mathrm{dom}\, \mathit{conflict\_detected} \bullet \\
\qquad \mathit{route}'(a) = \mathit{route}(a) \\
\mathit{pos}' = \mathit{pos} \land \mathit{conflict\_detected}' = \mathit{conflict\_detected}
\end{array}
$$

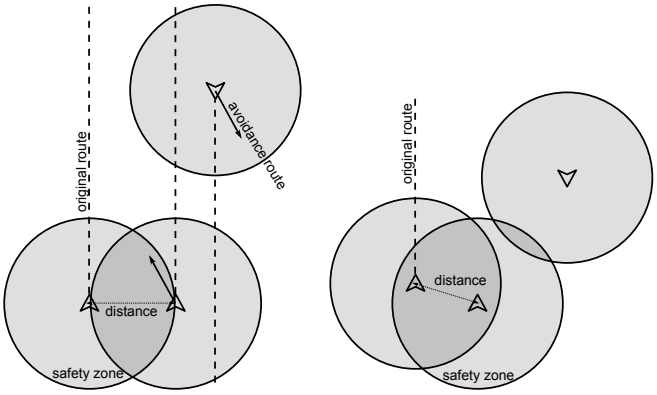
Although Eby's algorithm does not define what happens if two airplanes are already in conflict, we have assumed an

avoidance route will be calculated as in Hoekstra's algorithm [6].

Now we can model Eby's algorithm for conflict resolution by providing a replacement for *ChangeRoutes*. *ChangeRoutesEby* composes the required operations sequentially.

$$ChangeRoutesEby \equiv Restore \fatsemi DetectConflicts \fatsemi Avoid$$

It turns out that *ChangeRoutesEby* is not a refinement of *ChangeRoutes*. This is because airplanes that are rerouted are only guaranteed not to conflict with airplanes with which they detected a conflict. This leads to a problem, as shown in the Figures 1(a) and 1(b).



(a) An example configuration where airplanes will get into conflict

(b) Resulting configuration after moving one step

Fig. 1. Valid example configuration with resulting conflict

The two rerouted airplanes are rerouted because they detected a conflict with each other. The middle airplane changed its route and will now move on without considering the airplane to its left. This behaviour can lead to a conflict whenever the distance between two airplanes is smaller than $r_s + 2 * v_{max}$. Hence, the high-level specification is not met: an airplane *can* violate the safety radius of another airplane.

This does not contradict the results of Eby whose simulations show such violations are possible. He argues that this only occurs in dense airspaces and that the degradation in separation between aircraft is gradual. However, he cannot guarantee that collisions cannot occur.

E. An improved approach

To solve the problems mentioned in the previous section, we define an extended safety radius r_e such that airplanes cannot get into conflict in one step when outside this extended safety zone. When both airplanes are travelling towards each other at maximum speed, they must not enter the safety zone of each other. Hence, r_e extends the safety radius by $2 * v_{max}$.

$$r_e : \mathbb{N}$$
$$r_e = r_s + 2 * v_{max}$$

To use the extended safety radius, we refine *AirspaceRoutes* introducing the relation *at_risk* between airplanes. Airplanes are in that relation, if they are not in conflict, but in the extended safety radius of each other.

AirspaceExtended

$$AirspaceEby$$
$$at\_risk : airplanes \leftrightarrow airplanes$$

We adopt the operations *MoveAlongRoutes* and *Restore* from the previous sections with the addition that those operations do not change the set *at_risk*. When detecting conflicts, we use the extended safety radius. Furthermore, we change the relation *at_risk* so that airplanes are in that relation if the distance between their current positions is lower than the extended safety radius and greater or equal to the safety radius.

DetectConflictsExtended

$$\Delta AirspaceExtended$$
$$conflict\_detected' = \{a, a_c : airplanes \mid a \neq a_c \land (\exists t_c : \mathbb{N} \bullet 0 \leq t_c \leq t_l \land d(route(a)(t_c), route(a_c)(t_c)) < r_e) \land d(pos(a), dest(a)) > r_d \lor d(pos(a_c), dest(a_c)) > r_d\}$$
$$at\_risk' = \{a, a_c : airplanes \mid a \neq a_c \land r_s \leq d(pos(a), pos(a_c)) < r_e \land d(pos(a), dest(a)) > r_d \lor d(pos(a_c), dest(a_c)) > r_d\}$$
$$pos' = pos \land route' = route$$

To avoid conflicts, airplanes now calculate avoidance routes outside the extended safety radius of airplanes with which they have detected a conflict. The first part of the operation *AvoidAndEvade* below guarantees this.

Of course, an airplane can still enter the extended safety zone of another airplane. Hence, we specify an additional part of the operation for such airplanes. An implementation of the operation has to assure that once an airplane is in the extended safety radius – that is, when it is in the domain of *at_risk* – then it does not move closer to the other airplane, which is modeled in the second part of *AvoidAndEvade*.

The rest of the schema ensures that nothing else is changed.

AvoidAndEvade

$$\Delta AirspaceExtended$$
$$\forall a, a_c : airplanes \setminus at\_risk \bullet (a, a_c) \in conflict\_detected \Rightarrow d(route'(a)(2), route'(a_c)(2)) \geq r_e$$
$$\forall a, a_c : airplanes \bullet (a, a_c) \in at\_risk \Rightarrow d(route'(a_c)(2), route'(a)(2)) \geq d(pos(a_c), pos(a))$$
$$\forall a : airplanes \setminus \mathrm{dom}(at\_risk \cup conflict\_detected) \bullet route'(a) = route(a)$$
$$pos' = pos \land conflict\_detected' = conflict\_detected \land at\_risk' = at\_risk$$

We compose the operations as follows.

$ChangeRoutesSolution \equiv$
$Restore \fatsemi DetectConflictsExtended \fatsemi AvoidAndEvade$

Theorem 3: $ChangeRoutes \sqsubseteq ChangeRoutesSolution$

Proof: (1) The preconditions of both of the operations are true and so the proof for the preconditions is trivial.

(2) Since $v_{max} > 0$ and hence $r_s < r_e$,

$$
\begin{array}{l}
AvoidAndEvade \\
\Rightarrow \forall a, a_c : airplanes \setminus at\_risk \bullet \\
\quad (a, a_c) \in conflict\_detected \Rightarrow \\
\quad\quad d(route'(a)(2), route'(a_c)(2)) \geq r_e \\
\Rightarrow \forall a, a_c : airplanes \setminus at\_risk \bullet \\
\quad (a, a_c) \in conflict\_detected \Rightarrow \\
\quad\quad d(route'(a)(2), route'(a_c)(2)) \geq r_s
\end{array}
$$

Also,

$$
\begin{array}{l}
AvoidAndEvade \\
\Rightarrow \forall a, a_c : airplanes \bullet \\
\quad (a, a_c) \in at\_risk \Rightarrow \\
\quad\quad d(route'(a_c)(2), route'(a)(2)) \geq \\
\quad\quad\quad d(pos(a_c), pos(a))
\end{array}
$$

and

$$
\begin{array}{l}
DetectConflictsExtended \\
\Rightarrow \forall a, a_c : airplanes \bullet \\
\quad (a, a_c) \in at\_risk \Rightarrow \\
\quad\quad r_s \leq d(pos(a), pos(a_c)) < r_e
\end{array}
$$

Hence

$$
\begin{array}{l}
DetectConflictsExtended \fatsemi AvoidAndEvade \\
\Rightarrow \forall a, a_c : airplanes \bullet \\
\quad (a, a_c) \in at\_risk \Rightarrow \\
\quad\quad d(route'(a_c)(2), route'(a)(2)) \geq r_s
\end{array}
$$

Also

$$
\begin{array}{l}
DetectConflictsExtended \\
\Rightarrow \forall a : airplanes \setminus \mathrm{dom}(conflict\_detected \cup at\_risk) \bullet \\
\quad \forall a_c : airplanes \bullet a \neq a_c \wedge \\
\quad\quad (d(pos(a), dest(a)) > r_d \vee \\
\quad\quad d(pos(a_c), dest(a_c)) > r_d) \Rightarrow \\
\quad\quad\quad (d(pos(a), pos(a_c)) \geq r_e \vee \\
\quad\quad\quad d(pos(a), pos(a_c)) < r_s) \\
\Rightarrow \forall a : airplanes \setminus \mathrm{dom}(conflict\_detected \cup at\_risk) \bullet \\
\quad \forall a_c : airplanes \bullet a \neq a_c \wedge \\
\quad\quad (d(pos(a), dest(a)) > r_d \vee \\
\quad\quad d(pos(a_c), dest(a_c)) > r_d) \Rightarrow \\
\quad\quad\quad (d(pos(a), pos(a_c)) \geq r_e \vee \\
\quad\quad\quad (a, a_c) \in in\_conflict) \\
\Rightarrow \forall a : airplanes \setminus \mathrm{dom}(conflict\_detected \cup at\_risk) \bullet \\
\quad \forall a_c : airplanes \bullet a \neq a_c \wedge \\
\quad\quad (d(pos(a), dest(a)) > r_d \vee \\
\quad\quad d(pos(a_c), dest(a_c)) > r_d) \Rightarrow \\
\quad\quad\quad (d(route(a)(2), route(a_c)(2)) \geq r_s \vee \\
\quad\quad\quad (a, a_c) \in in\_conflict)
\end{array}
$$

since from the invariant of *AirspaceRoutes* we know that for all $a : airplanes$, $d(route(a)(1), route(a)(2)) < v_{max}$ and $r_e = r_s + 2 * v_{max}$.

Hence, since the routes of such airplanes are not changed by *DetectConflictExtended* or *AvoidAndEvade*, and none of the operations changes *pos* we have

$$
\begin{array}{l}
Restore \fatsemi DetectConflictsExtended \fatsemi AvoidAndEvade \\
\Rightarrow \forall a, a_c : airplanes \setminus \mathrm{dom}\, in\_conflict \bullet (a \neq a_c \wedge \\
\quad (d(pos(a), dest(a)) > r_d \vee \\
\quad d(pos(a_c), dest(a_c)) > r_d)) \Rightarrow \\
\quad\quad d(route'(a)(2), route'(a_c)(2)) \geq r_s \wedge \\
\quad pos' = pos
\end{array}
$$

That is,

$ChangeRoutesSolution \Rightarrow ChangeRoutes$

■

V. Conclusion

The models described in this paper provide an approach to developing a trustworthy distributed algorithm for free-flight separation. The conformity of the final model to the desired emergent behavior is proven and so all algorithms that conform to this model are guaranteed to achieve conflict-free movement while ensuring progression. The approach can be applied more generally to any distributed system which contains an arbitrary number of agents that have to move to certain destinations without colliding.

In contrast to other papers, we have developed and verified a specification for an algorithm that ensures conflict-free movement of an arbitrary number of agents. This is the main contribution of this work.

References

[1] RTCA Task Force 3. *RTCA Task Force 3 Final Report on Free Flight Implementation*. RTCA Inc., October 1995.

[2] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

[3] Martin S. Eby and Wallace E. Kelly III. Free flight separation assurance using distributed algorithms. In *Proc. of Aerospace Conf*, 1999.

[4] Andre L. Galdino, Cesar Munoz, and Mauricio Ayala-Rincon. Formal verification of an optimal air traffic conflict resolution and recovery algorithm. In *Logic, Language, Information and Computation*, volume 4576 of *Lecture Notes in Computer Science*, pages 177–188. Springer-Verlag, 2007.

[5] Dowek Gilles, Munoz Cesar, and Geser Alfons. Tactical conflict detection and resolution in a 3-d airspace. Technical report, Institute for Computer Applications in Science and Engineering, 2001.

[6] Jacob Marten Hoekstra. *Designing for Safety: the Free Flight Air Traffic Management Concept*. PhD thesis, Technische Universiteit Delft, 2001.

[7] Inseok Hwang and Claire J. Tomlin. Protocol-based conflict resolution for air traffic control. *Air Traffic Control Quarterly*, 15(1), 2007.

[8] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *FM 2009: Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 547–562. Springer-Verlag, 2009.

[9] Graeme Smith and J. W. Sanders. Formal development of self-organising systems. In *Autonomic and Trusted Computing*, volume 5586 of *Lecture Notes in Computer Science*, pages 90–104. Springer-Verlag, 2009.

[10] Jim Woodcock and Jim Davies. *Using Z: specification, refinement, and proof*. Prentice-Hall, Inc., 1996.