How Formal Methods Impels Discovery: A Short History of an Air Traffic Management Project

Ricky W. Butler, George Hagen, Jeffrey M. Maddalon, César A. Muñoz, Anthony Narkawicz
NASA Langley Research Center
Hampton, VA 23681, USA
{r.w.butler,george.hagen,j.m.maddalon,c.a.munoz,anthony.narkawicz}@nasa.gov
Gilles Dowek
École polytechnique and INRIA
LIX, École polytechnique
91128 Palaiseau Cedex, France
gilles.dowek@polytechnique.fr

Abstract

In this paper we describe a process of algorithmic discovery that was driven by our goal of achieving complete, mechanically verified algorithms that compute conflict prevention bands for use in en route air traffic management. The algorithms were originally defined in the PVS specification language and subsequently have been implemented in Java and C++. We do not present the proofs in this paper: instead, we describe the process of discovery and the key ideas that enabled the final formal proof of correctness.

1 Introduction

The formal methods team at NASA Langley has developed air traffic separation algorithms for the last 10 years. Given the safety-critical nature of these algorithms, we have emphasized formal verification of the correct operation of these algorithms. In February of 2008, Ricky Butler and Jeffrey Maddalon started a project to develop and formally verify algorithms that compute conflict prevention bands for en route aircraft.

In air traffic management systems, a conflict prevention system senses nearby aircraft and provides ranges of maneuvers that avoid conflicts with these aircraft The maneuvers are typically constrained to ones where only one parameter is varied at a time: track angles, vertical speeds, or ground speeds. Such maneuvers are easy for pilots to fly and have the advantage that they can be presented in terms of prevention bands, i.e. ranges of parameter values that should be avoided. Prevention bands display the maneuvers that result in conflict within a specified lookahead time as a function of one parameter. Without conflict prevention information, a pilot might create another conflict while seeking to solve a primary conflict or otherwise changing the flight plan.

The National Aerospace Laboratory (NLR) in the Netherlands refers to their conflict prevention capability as Predictive Airborne Separation Assurance System or Predictive ASAS [3]. The NLR approach provides two sets of bands: near-term conflicts are shown in red, while intermediate-term conflicts are shown in amber as illustrated in Figure 1. We did not directly analyze the NLR system because the algorithms were not available to us; however, we did use some of their interface ideas.

When we began this project, we had no idea that this project would take almost two years to complete and that four additional formal methods researchers would join our effort before we were done. This project has been one of the most interesting and enjoyable projects that we have worked on in our careers. The reason for this is manifold: (1) the work resulted in a very elegant algorithm that is implemented in Java and C++, (2) the final algorithm was very different from our first ideas, (3) there were many, many discoveries that were surprising. (At some points in the project we were having daily insights that improved the algorithm or a proof), (4) on the surface the problem looks simple, but looks can

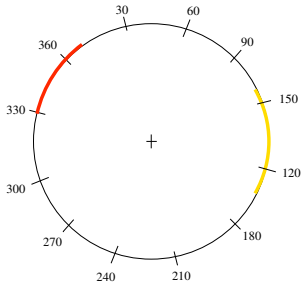

Figure 1: Compass Rose with Conflict Prevention Bands

be deceiving and the problem is actually very subtle with many special cases. After studying the problem for over a year, we developed an algorithm and rigorously documented this algorithm in a NASA Technical Memorandum [8]. We also formalized much of the mathematical development in the paper. We planned to follow up this paper with another paper that documented a complete formal proof of this algorithm. Much to our surprise this final formal proof step found some deficiencies in our algorithm. These deficiencies were repaired and a formal proof in the Prototype Verification System (PVS) [9] was finally completed in November 2009.

In this paper we will present a brief history of this project and highlight how the goal of formally verifying the algorithm in the PVS theorem prover pushed us to new discoveries. In addition to finding some subtle problems in the initial approach, we are confident that many of the discoveries would not have been made if we had taken a more traditional approach of constructing algorithms, testing, and revising them until they *worked*.

2 Notation

We consider two aircraft, the *ownship* and the *traffic* aircraft, that are potentially in conflict in a 3-dimensional airspace. The conflict prevention algorithm discussed here only uses state-based information, e.g,. initial position and velocity and straight line trajectories, i.e., constant velocity vectors in an Euclidean airspace. These approximations of real aircraft behavior are valid for short lookahead times (typically less than 5 minutes).

We use the following notations:

$\mathbf{s}_o$	3D vector	Initial position of the ownship aircraft
$\mathbf{v}_o$	3D vector	Initial velocity of the ownship aircraft
$\mathbf{s}_i$	3D vector	Initial position of the traffic aircraft
$\mathbf{v}_i$	3D vector	Initial velocity of the traffic aircraft

The components of each vector are scalar values, so they are represented without the bold-face font, for example $\mathbf{s}_o = (s_{ox}, s_{oy}, s_{oz})$. As a simplifying assumption, we regard the position and velocity vectors as accurate and without error. For notational convenience, we use $\mathbf{v}^2 = \mathbf{v} \cdot \mathbf{v}$ and we denote by $\mathrm{gs}(\mathbf{v})$ the ground speed of $\mathbf{v}$, i.e., the norm of the 2-dimensional projection of $\mathbf{v}$: $\mathrm{gs}(\mathbf{v}) = \sqrt{v_x^2 + v_y^2}$.

In the airspace system, the separation criteria are specified as a minimum horizontal separation D and a minimum vertical separation H (typically, D is 5 nautical miles and H is 1000 feet). It is convenient to develop the theory using a translated coordinate system. The relative position $\mathbf{s}$ is defined to be $\mathbf{s} = \mathbf{s}_o - \mathbf{s}_i$ and relative velocity of the ownship with respect to the traffic aircraft is denoted by $\mathbf{v} = \mathbf{v}_o - \mathbf{v}_i$. With

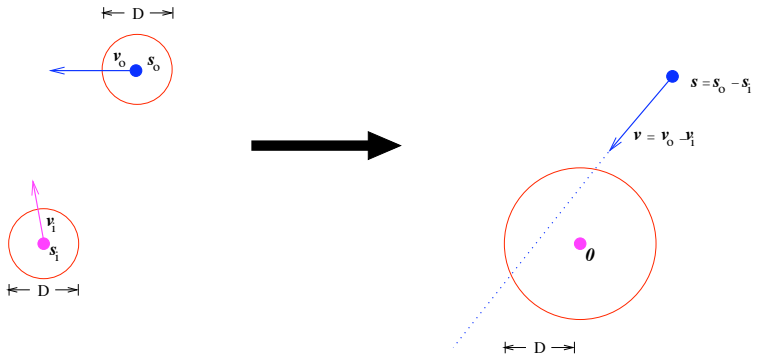

Figure 2: Translated Coordinate System

these vectors the traffic aircraft is at the center of the coordinate system and does not move. For example in the left side of Figure 2, the blue (upper) point represents the ownship with its velocity vector and its avoidance area (circle of diameter D around the aircraft). The magenta (lower) point represents the traffic aircraft. The right side represents the same information in the translated coordinate system. In a 3-dimensional airspace, the separation criteria defines a cylinder of radius D and half-height H around the traffic aircraft. This cylinder is called the *protected zone*.

In Figure 2, the two aircraft are potentially in conflict because the half-line defined by the relative velocity vector $\mathbf{v}$ intersects the protected area, meaning that in some future time the ownship will enter the protected zone around the traffic. If this future time is within the lookahead time T, the two aircraft are said to be in *conflict*.

3 The Start of the Project

We began our project by first surveying the literature for previous solutions. Hoekstra [4] describes some algorithms developed at NLR with some diagrams [3], but unfortunately he did not provide much detail about how the algorithms actually worked. We decided to develop our own track angle, ground speed, and vertical speed algorithms. In this paper, we will only present the track band algorithm. These are the most challenging and interesting of the three and the other bands are computed and verified using analogous methods. We adopted the NLR idea of introducing two parameters, T_{red} (typically three minutes) and T_{amber} (typically five minutes), which divide the set of conflicts based on their nearness (in time) to a loss of separation (see figure 1). If a loss of separation will occur within T_{red}, then the region is colored red. On the other hand, if a loss of separation will occur after T_{red}, but before T_{amber}, then the region is colored amber, otherwise it is painted green.

We first recognized that each aircraft's contribution to the prevention band is independent of all other aircraft; thus, the problem neatly decomposes into two steps:

1. Solve the bands problem for the ownship relative to each other aircraft separately.

2. Merge all of the pairwise regions.

We also quickly realized that an iterative solution was possible for the first step. We already had a formally proven, efficient algorithm available to us named `CD3D` that decides if a conflict occurs for specific values of $\mathbf{s}_o$, $\mathbf{v}_o$, $\mathbf{s}_i$, and $\mathbf{v}_i$, and parameters D, H, and T. More formally, `CD3D` determines whether there exists a future time t where the aircraft positions $\mathbf{s}_o + t\mathbf{v}_o$ and $\mathbf{s}_i + t\mathbf{v}_i$ are within a horizontal distance D of each other *and* where the aircraft are within vertical distance H of each other. In other words there

is a predicted loss of separation within the lookahead time. Therefore, one need only to execute `CD3D` iteratively, varying the track angle from 0 to 360° per traffic aircraft. By evaluating different scenarios, we determined that a step size of 0.1° would be adequate for ranges of up to 200 nautical miles. An iterative approach would consume more computational resources than an analytical one where the edges of the bands are computed directly. An iterative approach may not scale well where such separation assurance algorithms must be executed for many different traffic aircraft every second. Despite these disadvantages, the existence of an iterative approach provided a fall-back position: if we were not able to discover a formally verifiable analytical solution, then we knew we could use the interative approach.

4 Search for an Analytical Solution

To solve the prevention bands problem in an analytical way, it is useful to define separate horizontal and vertical notions of conflict. In the relative coordinate system, we define that a *horizontal conflict* occurs if there exists a future time t within the lookahead time T where the aircraft are within horizontal distance D of each other, i.e., $(s_x + tv_x)^2 + (s_y + tv_y)^2 < D^2$. where **s** and **v** are, respectively, the relative position and the relative velocity of the ownship with respect to the traffic aircraft. A *vertical conflict* occurs if there exists a future time t within the lookahead time T where the aircraft are within horizontal distance H of each other, i.e., $|s_z + tv_z| < H$. We say that two aircraft are in *conflict* if there is a time t where they are in horizontal and vertical conflict. Formally, we define the predicate `conflict?` as follows

$$\texttt{conflict?}(\mathbf{s},\mathbf{v}) \quad \equiv \quad \exists\, 0 \le t \le T : (s_x + tv_x)^2 + (s_y + tv_y)^2 < D^2 \text{ and } |s_z + tv_z| < H. \tag{1}$$

4.1 Track Only Geometric Solution

We begin with a non-translated perspective shown in Figure 3. The track angle is denoted by α[1]. For

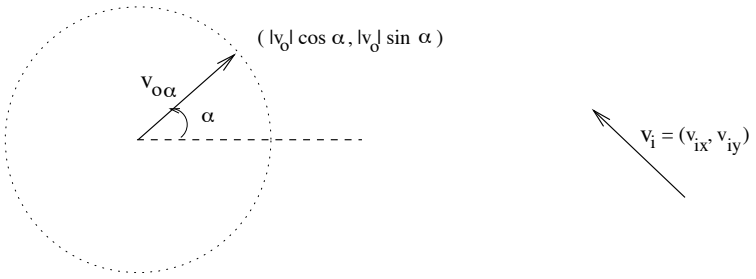

Figure 3: Two Dimensional Version of Track Bands Problem

given vectors $\mathbf{v}_o$ and $\mathbf{v}_i$, we need to find the track angles α such that the relative vector $\mathbf{v}_\alpha = \mathbf{v}_{o\alpha} - \mathbf{v}_i$:

$$\mathbf{v}_\alpha = (|\mathbf{v}_o| \cos\alpha - v_{ix},\ |\mathbf{v}_o| \sin\alpha - v_{iy}), \tag{2}$$

is not in conflict.

Our initial approach was to divide the conflict prevention problem into simplifying cases. We discovered later that this division was unnecessary and more elegant formulations provided the necessary leverage to formally verify the algorithm. In any case, we decided to first solve the track bands problem in two dimensions without consideration of the lookahead time. We also decided to ignore vertical speed

[1] In air traffic management, the track angle is calculated from true north in a clockwise direction, but here we have followed the traditional mathematical definition.

considerations and look at the horizontal plane only. The problem thus reduced to finding the tangent lines to the horizontal protection zone (in the relative frame of reference) as a function of α. We begin with the observation that in order for a vector to be tangent it must intersect the circle of the protection zone. In other words, we need solutions of $|\mathbf{s}+t\mathbf{v}_\alpha| = D$ or equivalently

$$(\mathbf{s}+t\mathbf{v}_\alpha)^2 = D^2 \tag{3}$$

where the vectors are two-dimensional. Expanding we obtain a quadratic equation $at^2+bt+c=0$ with ${\mathbf{v}_\alpha}^2$, $b = 2(\mathbf{s}\cdot\mathbf{v}_\alpha)$, and $c = \mathbf{s}^2 - D^2$. The tangent lines are precisely those where the discriminant of this equation is zero. In other words, where $b^2-4ac=0$. But, expanding the dot products yield:

$$\begin{aligned} b^2 &= 4[s_x(\omega\cos\alpha - v_{ix}) + s_y(\ \omega\sin\alpha - v_{iy})]^2 \\ 4ac &= 4(\omega^2 - 2\omega(v_{ix}\cos\alpha + v_{iy}\sin\alpha) + v^2)(\mathbf{s}\cdot\mathbf{s} - D^2) \end{aligned}$$

The discriminant finally expands into a complex second-order polynomial in $\sin\alpha$ and $\cos\alpha$. But to solve for α, we need to eliminate the $\cos\alpha$ using the equation

$$\cos\alpha = \sqrt{1-\sin^2\alpha}$$

The net result is an unbelievably complex fourth order polynomial in $\sin\alpha$. Solving for α analytically would require the use of the quartic formulas. Although these formulas are complicated, such a program could probably be written in a day or two. But, how would we verify these solutions? After all, the quartic equations involve the use of complex analysis. Therefore, we began to look for simplifications.

We found a simplification of the discriminant that had been used in the verification of the KB3D algorithm [6]:

$$b^2-4ac = 0 \text{ if and only if } (\mathbf{s}\cdot\mathbf{v}) = R\varepsilon\det(\mathbf{s},\mathbf{v}), \tag{4}$$

where $\varepsilon \in \{-1,+1\}$, $\det(\mathbf{s},\mathbf{v}) \equiv \mathbf{s}^\perp\cdot\mathbf{v}$, $\mathbf{s}^\perp = (-s_y, s_x)$, and $R = \frac{\sqrt{\mathbf{s}^2-D^2}}{D}$. The beauty of the final form is that the equation is linear on $\mathbf{v}$. The two solutions are captured in the two values of ε. When we instantiate $\mathbf{v}_\alpha$ in this formula, we end up with a quadratic equation in $\sin\alpha$.

Using this approach, we were able to derive the following solutions for α. If $\frac{|G|}{\sqrt{E^2+F^2}} \le 1$ then in some 2π range, we have

$$\begin{aligned} \alpha_1 &= \operatorname{asin}\left(\frac{G}{\sqrt{E^2+F^2}}\right) - \operatorname{atan}(E,F), \\ \alpha_2 &= \pi - \operatorname{asin}\left(\frac{G}{\sqrt{E^2+F^2}}\right) - \operatorname{atan}(E,F), \end{aligned}$$

where

$$E = \omega(R\varepsilon s_x - s_y),\ \ F = -\omega(R\varepsilon s_y + s_x),\ \ G = \mathbf{v}_i\cdot(R\varepsilon\,\mathbf{s}^\perp - \mathbf{s}),$$

Since E, F, and G are all functions of ε, we have two pairs of α_1 and α_2 or a total of four total angles. These angles are the places where the track prevention band changes color, assuming no lookahead time. This result was formalized in the PVS theorem prover and implemented in Java.

4.2 Solution with Lookahead Time

The solution presented so far only considers the 2-dimensional case with no lookahead time. Figure 4 illustrates three distinct cases that appear when the lookahead time is considered: (a) the protection zone is totally within lookahead time, (b) the zone is partially within, and (c) the zone is totally beyond the

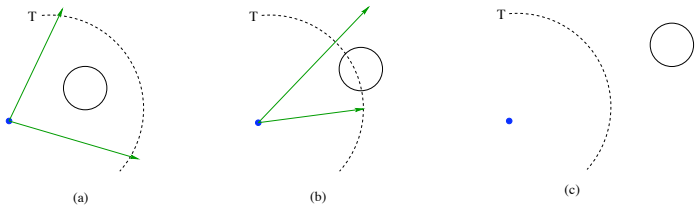

Figure 4: Relationship of Encounter Geometry and Lookahead Time

lookahead time. Cases (a) and (c) were easy to handle, but we realized that case (b) was going to take some additional analysis. We were quite pleased with our initial geometric result and decided to present the result to our branch head and research director. During the presentation, a member of the original KB3D team announced, "I think you can solve this problem without trigonometry," and he urged us to defer the use of trigonometry until the last possible moment. In other words, he suggested that we solve for $(\mathbf{v}_\alpha)$ without expanding its components. Only after the appropriate abstract solution vector is found, should the conversion to a track angle, α, be made. This was a key idea that had been used in the development of the KB3D algorithms, which resulted in very efficient and elegant algebraic solutions [1]. Indeed, we realized that the geometric problem was solvable by a particular kind of KB3D resolutions called *track lines*, computed by the function `track_line`:

`track_line(`$\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, \varepsilon, \iota$`): Vect2`

The function `track_line` returns the vector $\mathbf{0}$ when all track angles for the ownship yield a potential conflict. Otherwise, the vector returned by this function is a velocity vector for the ownship that is tangent to the 2-dimensional protected zone. Since ε and ι are ± 1, there are four possible track line solutions for given $\mathbf{s}$, $\mathbf{v}_o$, and $\mathbf{v}_i$.

The key to solving track bands with a lookahead time is to find where the *projected* lookahead time intersects the protected zone. That is, plot where the relative position of the aircraft will be after T units of time in every possible direction given an unchanged ground speed and find the intersection points with the protection zone. The function `track_circle`, also available in KB3D, provides these solutions, which are called *track circle* solutions.

The function `track_circle`

`track_circle(`$\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, T, \iota$`): Vect2`

returns the vector $\mathbf{0}$ when there are no track circle solutions, i.e., when the lookahead time boundary T and the protected zone do not intersect, or when there are an infinite number of solutions. Otherwise, the vector returned by this function is a velocity vector for the ownship that intersects the 2-dimensional protected zone at a time later than T. Since ι is ± 1, for given $\mathbf{s}$, $\mathbf{v}_o$, and $\mathbf{v}_i$ there are two possible track circle solutions. The `track_line` and `track_circle` functions are derived and discussed in [8].

We believe that the lookahead problem would not have been analytically tractable using the trigonometric approach pursued at first. This switch to a pure algebraic approach was fundamental to achieving the final proof of the 3-dimensional bands algorithm.

4.3 The Track Bands Algorithm

The two functions, `track_line` and `track_circle`, form the basis of the track bands algorithm. We define a *critical vector* as a relative velocity vector where the color of the bands *may* change. These

critical vectors are

$$\begin{aligned}
\mathbf{R}_{mm} &= \texttt{track_line}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, -1, -1),\\
\mathbf{R}_{mp} &= \texttt{track_line}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, -1, +1),\\
\mathbf{R}_{pm} &= \texttt{track_line}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, +1, -1),\\
\mathbf{R}_{pp} &= \texttt{track_line}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, +1, +1),\\
\mathbf{C}_{rm} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, T_{red}, -1),\\
\mathbf{C}_{rp} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, T_{red}, +1),\\
\mathbf{C}_{am} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, T_{amber}, -1),\\
\mathbf{C}_{ap} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, T_{amber}, +1),\\
\mathbf{C}_{em} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, t_{entry}, -1),\\
\mathbf{C}_{ep} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, t_{entry}, +1),\\
\mathbf{C}_{xm} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, t_{exit}, -1),\\
\mathbf{C}_{xp} &= \texttt{track_circle}(\mathbf{s}, \mathbf{v}_o, \mathbf{v}_i, t_{exit}, +1).
\end{aligned}$$

Some of these vectors may be zero vectors in which case they are ignored. The times t_{entry} and t_{exit} are the calculated entry and exit times into the protection zone.

In the track bands algorithm, these vectors are calculated, the corresponding track angles (using `atan`) are computed and sorted into a list of angles. Next, the angles 0 and 2π are added to the list to provide appropriate bounding. Then, a conflict probe (such as CD3D) us applied to an angle between each of the critical angles to characterize the whole region (i.e., determine which color the region should be painted: green, amber, or red). This procedure is iterated between the ownship and all traffic aircraft. Finally, the resulting bands are merged to get the display as in Figure 1.

5 Formal Verification of Pairwise Prevention Bands Algorithms

The functions `track_line` and `track_circle` discussed in Section 4.2 have been verified *correct* for conflict resolution, i.e., they compute vectors that yield conflict free trajectories for the ownship, and *complete* for track prevention bands, i.e., they compute all critical vectors where the track bands change colors. These functions are slightly different from the original ones presented in [8]. Indeed, the functions presented in that report, while correct for conflict resolution, failed to compute all the critical vectors. For conflict prevention bands this is a safety issue, because a region that should be colored red can be colored green instead. Interestingly, those functions, which had been tested on over 10,000 test cases without any error manifestations, were in fact incorrect. The deficiencies in these functions were only found during the formal verification process!

The general idea of the correctness proof of the prevention bands algorithms is as follows:

1. For a given parameter of the ownship, e.g., track angle, define a function $\Omega_{\text{trk}} : \mathbb{R} \to \mathbb{R}$, parametrized by $\mathbf{s}$, $\mathbf{v}_o$, and $\mathbf{v}_o$, that characterizes conflicts in the following way: $\Omega_{\text{trk}}(\alpha) < 0$ if and only if $\texttt{conflict?}(\mathbf{s}, \mathbf{v}_\alpha)$, where $\mathbf{v}_\alpha$ is defined as in Formula 2.
2. Prove that the critical vectors computed in Section 4.3 are *complete*, i.e., they compute all of the zeros of the function Ω_{trk},
3. Prove that the function Ω_{trk} is continuous.
4. Use the Intermediate Value theorem to deduce that any point in an open bands region, e.g., the mid-point, determines the color of the whole band. This last step requires the existence of a conflict probe algorithm that is proven correct, which we have already developed and verified.

This approach is illustrated in figure 5.

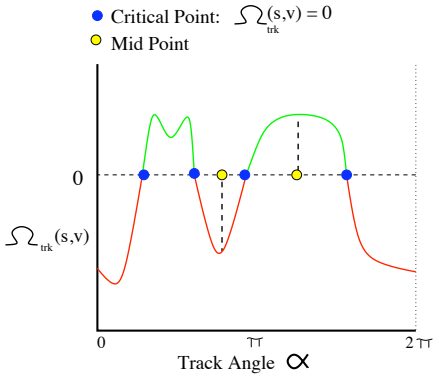

Figure 5: Ω_{trk} Proof Approach

The discovery of this Ω_{trk} proof approach also directly influenced the final bands algorithms. Originally we expected to compute the color of a region. This proof method lead us to the idea of using the `CD3D` conflict probe on the midpoint of the region in order to color the region.

We first tried this proof approach on a simplified version of the problem: the two-dimensional ground speed bands with infinite lookahead time. In this case, the function Ω_{gs} must characterize $\texttt{conflict?}(\mathbf{s},\mathbf{v}_k)$, where $\mathbf{v}_k = \frac{k}{\text{gs}(\mathbf{v})}\mathbf{v}$.

The following formula provided the needed relationship between horizontal conflict that does not include a quantification over time:

$$\texttt{horizontal\_conflict?}(\mathbf{s},\mathbf{v}) \iff \mathbf{s}\cdot\mathbf{v} < R\,\texttt{det}(\mathbf{v},\mathbf{s}) < -\mathbf{s}\cdot\mathbf{v}, \tag{5}$$

where, R is defined as in Formula 4.

The function Ω_{gs} was constructed based on this theorem. The resulting function required the use of if-then-else logic so the proof that it was continuous was tedious. After much effort, we were able to prove in PVS that the ground-speed functions `gs_line` and `gs_circle`, which are analogous to `track_line` and `track_circle`, were complete assuming no look-ahead time.

The lesson learned from this first attempt was that we needed a more abstract way of defining the functions Ω_{gs} and Ω_{trk} so that the complexity of the continuity proofs could be untangled from the subtleties of the track and ground speed resolutions. With this in mind, we defined a function $\Omega\colon \mathbb{R}^n \to \mathbb{R}$, parametrized by $\mathbf{s}$, $\mathbf{v}_o$, and $\mathbf{v}_o$, such that $\Omega_{\text{trk}} = \Omega \circ \mathbf{v}_\alpha$ and $\Omega_{\text{gs}} = \Omega \circ \mathbf{v}_k$. The continuity of Ω_{trk} and Ω_{gs} is a consequence of the continuity of Ω, which was proved once and for all for all kinds of bands, and the continuity of $\mathbf{v}_\alpha$ and $\mathbf{v}_k$. All this seems straightforward except that there are several technical difficulties.

The function Ω is closely related to the function that computes the minimum distance between two aircraft. That function is, in general, noncontinuous for an infinite lookahead time. Interestingly, it is continuous when a lookahead time is considered, but the general proof of this fact requires the use of vector variant of the Heine-Cantor Theorem, i.e., if M is a compact metric space, then every continuous function $f : M \to N$, where N is a metric space, is uniformly continuous. Furthermore, the minimum distance function may have flat areas. Therefore, special attention has to be paid to the definition of Ω to guarantee that the set of critical points is finite. Otherwise, it cannot be proven that the critical vector functions are complete.

The next sections discuss the formal verification of the prevention bands algorithms with lookahead time for both the 2-D and 3-D cases.

5.1 Verification of 2D Prevention Bands

In the 2-dimensional case, a direct definition of Ω is possible by using τ, the time of minimal horizontal separation between two aircraft:

$$\tau(\mathbf{s},\mathbf{v}) \quad = \quad -\frac{\mathbf{s}\cdot\mathbf{v}}{\mathbf{v}^2}. \tag{6}$$

From τ, we can define Ω as follows:

$$\Omega(\mathbf{v}) \quad = \quad (\mathbf{s}+\min(\max(0,\tau(\mathbf{s},\mathbf{v})),T)\mathbf{v})^2 - D^2, \tag{7}$$

where $\mathbf{s}$ is the relative distance between the ownship and the traffic aircraft.

The use of square distances in Formula 7 avoids the use of the square root function. Since the minimum and maximum of a continuous function is continuous, the use of min and max is easier to handle than the if-then-logic used in our first attempt.

The function Ω is not defined when $\mathbf{v}$ is $\mathbf{0}$. Therefore, rather than using Ω directly, we used the function $\mathbf{v} \mapsto \mathbf{v}^2\Omega(\mathbf{v})$, which is defined everywhere, and proved that it is continuous and that it correctly characterizes conflicts, i.e., `conflict?`$(\mathbf{s},\mathbf{v})$ if and only if $\mathbf{v}^2\Omega(\mathbf{v}) < 0$.

The function $\mathbf{v}^2\Omega(\mathbf{v})$ has an infinite number of zeroes in some special cases, e.g., when $\mathbf{s}$ is at the border of the protected zone, i.e, when $\mathbf{s}^2 = D^2$. In those, special cases, we use an alternative characterization of conflicts that has the required properties. In August 2009, we completed the proof of the 2-dimensional track and ground speed bands with finite lookahead time. For additional technical details on this formal development, we refer the reader to [7].

5.2 Verification of 3D Prevention Bands

The verification of the 3D conflict prevention bands algorithm is similar to that of the 2D algorithm. Indeed, many of the geometrical concepts critical to the verification in the 2D case can be generalized to the 3D case. However, these generalizations are typically nontrivial because, geometrically, a circle (a 2D protected zone) is much easier to work with than a cylinder (a 3D protected zone). The Ω function used in the verification of the 2D algorithm uses the horizontal time of minimum separation τ, which is easy to compute analytically. In contrast, the fact that a cylinder is not a smooth surface indicates that a 3D generalization of the Ω function will not be as simply defined.

Despite these geometric challenges, a concept was discovered that can be used to simplify geometry problems involving distance on cylinders. This concept is the notion of a *normalized cylindrical length* [2]:

$$||\mathbf{u}||_{\text{cyl}} \quad = \quad \max(\frac{\sqrt{u_x^2+u_y^2}}{D}, \frac{|u_z|}{H}). \tag{8}$$

This metric nicely reduces horizontal and vertical loss of separation into a single value. Indeed, if $\mathbf{s}$ is the relative position vector of two aircraft, then $||\mathbf{s}||_{\text{cyl}} < 1$ if and only if the aircraft are in 3D loss of separation.

Using the cylindrical distance metric, the Ω function can be defined in the 3D case as follows.

$$\Omega_{3D}(\mathbf{v}) \quad = \quad \min_{t\in[0,T]} ||\mathbf{s}+t\cdot\mathbf{v}||_{cyl} - 1, \tag{9}$$

where S is the relative position vector between the ownship and traffic aircraft. An immediate consequence of this definition is that two aircraft are in conflict if and only if $\Omega_{3D}(\mathbf{v}) < 0$.

The correctness of the prevention bands algorithms relies on the fact that Ω_{3D} is a continuous function of $\mathbf{v}$, that the set of critical vectors, i.e., the zeroes of the function, is finite, and that the critical vector algorithms are complete.

For many functions, a proof of continuity follows immediately from definitions. In this case, the function Ω_{3D} is a minimum over the closed interval $[0,T]$. While standard methods from differentiable calculus are often employed in similar problems, this function is a minimum of a non-differentiable function, namely the cylindrical length. Its closed form involves several if-else statements and it would be difficult to use directly in a proof of continuity. Thus, somewhat more abstract results from real analysis were needed to be extended to vector analysis, e.g., the notion of limits, continuity, compactness, and finally the Heine-Cantor Theorem.

As in the 2-dimensional case, the function Ω_{3D} may have flat areas and, consequently, in some special cases, may have an infinite number of critical zeros. We carefully identified these special cases and then used an alternative definition of Ω_{3D}. These special cases are extremely rare, indeed all the missing critical vectors in the original algorithms presented in [8] were due to these special cases. Although they are rare, dealing with them is necessary for the correctness proof of the algorithms. If one critical vector is missing, the coloring of the bands will be potentially switch from red to green or vice-versa.

Finally, the PVS proof that the algorithms find all of the critical points is less abstract but more tedious than the proof of continuity. It required the development of several PVS theories about the Ω_{3D} function, which are general enough to be used in other state-based separation assurance algorithms. The proof of the correctness of the 3-dimensional algorithms for track, ground speed, and vertical speed with finite lookahead time was complete in December 2009.

6 Verification of the Merge Algorithm

Having developed methods to calculate pairwise solutions (see section 4.3), we turned to the problem of merging all of the pairwise solutions into a single set of bands for all aircraft (second problem listed in section 3). Our original approach relied on complex reasoning about overlapping regions coupled with precedence rules: an amber region take precedence over a green region but not a red region. We developed a Java version that "worked," but it was soon obvious that this solution was complex enough that it could not be implicitly trusted. We decided that a formal verification of the merge algorithm was necessary.

As we began considering how to formally specify and analyze this merge algorithm, we recognized two problems with our approach. First, our algorithm was specialized to the precise problem we were working; almost any change to the system would require a new algorithm and therefore a new verification. The other problem was that our algorithm was monolithic; there was no obvious decomposition into general pieces that could be verified once and used in different contexts. To resolve these problems, we soon realized that standard set operations (set union, set difference, etc.) could be used to implement not only the multiple-aircraft merge problem, but also the different colors of conflict prevention information.

Suppose we had a way of determining the set of track angles that have a loss of separation within time T, denoted $\mathscr{G}_{<T}$. Then since $T_{red} < T_{amber}$, we can define the colored bands of track angles in terms of this new set:

$$\begin{aligned} \mathscr{G}_{red} &= \mathscr{G}_{<T_{red}} \\ \mathscr{G}_{amber} &= \mathscr{G}_{<T_{amber}} - \mathscr{G}_{<T_{red}} \\ \mathscr{G}_{green} &= \{\alpha \,|\, 0^\circ \leq \alpha < 360^\circ\} - \mathscr{G}_{<T_{amber}} \end{aligned}$$

This observation simplified the analysis, because only one set, $\mathscr{G}_{<T}$ needed to be analyzed.

Next, we observed that each aircraft's contribution to the set $\mathscr{G}_{<T}$ is independent of all other traffic; thus, the problem neatly divides into a series of aircraft pairs: the ownship and each traffic aircraft. If we use $\mathscr{G}^{o,i}_{<T}$ to represent the set of track angles which cause a loss of separation within time T between traffic aircraft i and the ownship o, then the set of track angles for all traffic is then be formed by

$$\mathscr{G}_{<T} = \bigcup_{i \in traffic} \mathscr{G}^{o,i}_{<T}$$

This observation simplified the analysis again, because now we only needed to find the track angles which cause a conflict between two aircraft, denoted by the set $\mathscr{G}^{o,i}_{<T}$. The set $\mathscr{G}^{o,i}_{<T}$ corresponds to output of the algorithm presented in section 4.3 except the two lookahead times are replaced with one time, T.

By using set operations we had a well-defined specification of the key parts of our merge algorithm. However common implementations of sets in programming languages do not include efficient ways to deal with ranges of floating point numbers; therefore, we chose to implement our own. We then performed a code-level verification of the set union and set difference operations that were used in the merge algorithm.

Each band is represented by an interval describing its minimal and maximal values, with the set of all bands of one color being an interval set. These interval sets were internally represented by arrays of (ordered) intervals. Necessary properties for the implementation would be that the data structures representing the bands remained ordered and preserved the proper value ranges within a set of bands.

Set union combines overlapping bands as appropriate and set difference involves breaking larger bands into smaller ones. There were two complications in the verification. The first complication arose because the implementation used a subtle notion of ordering where a zero or a positive value represents an actual position in the array of intervals, but a negative value represents a point between (or beyond) the intervals currently in the set. Although tedious, this verification was completed without issues. The second complication resulted from the boundary conditions. The original Java implementation did not clearly indicate whether the endpoints of a band of green angles were part of that the green band, or if they were part of the next band. The formal verification brought this issue forward. It was not possible to exclusively use closed or open intervals for both union and difference operations: the use of one necessitates the use of the other. For instance, removing a closed interval, which includes the endpoints, leaves us with open intervals—everything up to, but not including, these end points. Also, removing two adjacent open intervals leads to a left over point between them.

At this point we had an implementation issue. Should we use fully general set operations with open and closed intervals, or should we use set operations with well-defined, but non-standard semantics. In the interest of having a consistent interface and eliminating redundant code, we decided to take the second route. The union operation would assume that inputs would be closed intervals and therefore, the result would be closed intervals. The difference operations would assume that the set to be subtracted would consist of only open intervals and therefore, the result would be only of closed intervals. As mentioned above, this lead to the possibility of introducing artifacts of *singleton* intervals, where both endpoints have the same value. After consideration, however, we realized these points could be safely eliminated, as they are equivalent to a critical point at a local minimum or maximum. Green singletons could be eliminated without introducing additional danger, and a red singleton would represent a brush against (but not a cross into) the traffic aircraft's protected zone.

The formal verification of merge algorithm required us to think deeply about what the merge algorithm was trying to accomplish. During this analysis process we were able to develop an elegant solution which can be presented in a paragraph of text, instead of a complicated 400 line Java program with many special cases. In addition the formal verification process required us to clearly specify how our algorithm would behave at the points where there is a transition from one color to another.

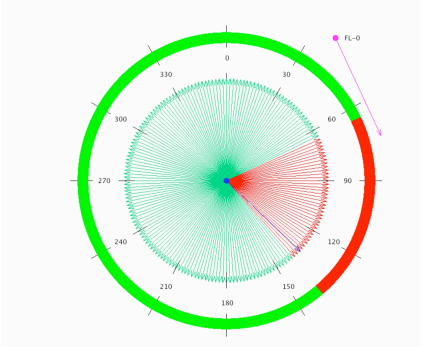

Figure 6: Screenshot of Track Prevention Band Display

7 Java and C++ Implementations

Ultimately these algorithms will be brought into large simulation environments where they will be evaluated for performance benefits (improvements to airspace capacity or aircraft efficiency). Some of these simulation environments are in Java and some are in C++. Therefore a requirement of this project was not only to develop an algorithm and verify it, but to also produce Java and C++ implementations of the algorithm. The initial Java version of the algorithm was available in December 2008 (see Figure 6). This version successfully passed the limited test suite we developed. By the summer of 2009 we had a C++ version and the testing apparatus to verify exact agreement between the Java and C++ versions, along with a regression suite of 100 test scenarios.

Since PVS is a specification language and contains non-computable functions, we deliberately restricted our use of PVS in the specification of our algorithms to only the computable elements. In this way there was a direct translation of PVS into Java or C++. We are currently developing a tool to automatically convert PVS specifications into Java [5], but the tool is not yet mature enough to handle the specification of these kinds of algorithms. However, we ran into problems even in our manual conversion of the algorithm. For instance, the PVS libraries contain all the appropriate vector operations (addition, dot-product, etc.) but libraries do not exist for these operations in standard Java or C++. We searched for a third-party library to offer these functions, but there were two downsides to these libraries. First, we wanted the same algorithm in both Java and C++, but most libraries were targeted to only one programming language. Second, we desired a library with identical semantics in both languages. But even when we found libraries that supported both languages, we inevitably discovered certain quirks in their implementation. For example, One vector library took full advantage of the imperative nature of the Java language, implementing functions on vectors which would change the components of the vector. While this results in efficient code, because object creation is not necessary, it does not closely relate to the functional style of PVS. Because of these incompatibilities, we chose to implement our own vector libraries. For similar reasons, we developed our own set operations (union and intersection).

Even with this hand translation, we still do not have an exact behavioral replica of the PVS in Java or C++. The most glaring difference is that Java and C++ use floating point numbers while PVS uses actual real numbers. All of our verifications in PVS are accomplished with vectors defined over the real numbers. This can be thought of as computation using infinite precision arithmetic. Clearly, our Java and C++ implementations execute on less powerful machines than this. There are several places where we must be especially careful:

- Calculation of quadratic discriminants. Since we are often computing tangents, the theoretical

value is zero, but the floating point answer can easily be a small negative number near zero. In this case, we would miss a critical point.

- The possibility of the mid-point of a region being very close to zero.

Finally, another aspect related to this issue is that the data input into the algorithm is not precise. A standard engineering assumption is that the error in the input data will overwhelm any error introduced by floating point computations. However, we would like to make a formal statement that includes both data and computational errors.

8 Conclusions

In this paper, we have presented a short history of the development and formal verification of prevention bands algorithms. The resulting track-angle, ground speed, and vertical speed bands algorithms are far more simple than our earlier versions. The goal of completing a formal proof forced us to search for simplifications in the algorithms and in the underlying mathematical theories. A key insight that enabled the completion of this work, is that trigonometric analysis should be deferred until the latest possible time. Although, the project took far longer than we expected, we are very pleased with the elegance and efficiencies of the discovered algorithms.

References

[1] G. Dowek, A. Geser, and C. Muñoz. Tactical conflict detection and resolution in a 3-D airspace. In *Proceedings of the 4th USA/Europe Air Traffic Management R&DSeminar, ATM 2001*, Santa Fe, New Mexico, 2001. A long version appears as report NASA/CR-2001-210853 ICASE Report No. 2001-7.

[2] Gilles Dowek and C. Muñoz. Conflict detection and resolution for 1,2,...,N aircraft. In *Proceedings of the 7th AIAA Aviation, Technology, Integration, and Operations Conference, AIAA-2007-7737*, Belfast, Northern Ireland, 2007.

[3] J. Hoekstra, R. Ruigrok, R. van Gent, J. Visser, B. Gijsbers, M. Valenti, W. Heesbeen, B. Hilburn, J. Groeneweg, and F. Bussink. Overview of NLR free flight project 1997-1999. Technical Report NLR-CR-2000-227, National Aerospace Laboratory (NLR), May 2000.

[4] J. M. Hoekstra. Designing for safety: The free flight air traffic management concept. Technical Report 90-806343-2-8, Technische Universiteir Delft, November 2001.

[5] Leonard Lensink, César Muñoz, and Alwyn Goodloe. From verified models to verifiable code. Technical Memorandum NASA/TM-2009-215943, NASA, Langley Research Center, Hampton VA 23681-2199, USA, June 2009.

[6] Jeffrey Maddalon, Ricky Butler, Alfons Geser, and César Muñoz. Formal verification of a conflict resolution and recovery algorithm. Technical Report NASA/TP-2004-213015, NASA/Langley Research Center, Hampton VA 23681-2199, USA, April 2004.

[7] Jeffrey Maddalon, Ricky Butler, César Muñoz, and Gilles Dowek. A mathematical analysis of conflict prevention information. In *Proceedings of the AIAA 9th Aviation, Technology, Integration, and Operations Conference, AIAA-2009-6907*, Hilton Head, South Carolina, USA, September 2009.

[8] Jeffrey Maddalon, Ricky Butler, César Muñoz, and Gilles Dowek. A mathematical basis for the safety analysis of conflict prevention algorithms. Technical Report TM-2009-215768, NASA Langley, June 2009.

[9] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer-Verlag, June 1992.