Analyzing the Next Generation Airborne Collision Avoidance System

Christian von Essen[1,⋆] and Dimitra Giannakopoulou[2,⋆]

[1] Verimag, Grenoble, France
christian.vonessen@imag.fr
[2] NASA Ames Research Center
Moffett Field, CA, USA
dimitra.giannakopoulou@nasa.gov

Abstract. The next generation airborne collision avoidance system, ACAS X, departs from the traditional deterministic model on which the current system, TCAS, is based. To increase robustness, ACAS X relies on probabilistic models to represent the various sources of uncertainty. The work reported in this paper identifies verification challenges for ACAS X, and studies the applicability of probabilistic verification and synthesis techniques in addressing these challenges. Due to shortcomings of off-the-shelf probabilistic analysis tools, we developed a framework that is designed to handle systems with similar characteristics as ACAS X. We describe the application of our framework to ACAS X, and the results and recommendations that our analysis produced.

Keywords: Markov decision processes, probabilistic verification, probabilistic synthesis, aircraft collision avoidance.

1 Introduction

The current onboard collision avoidance standard, TCAS [7], has been successful in preventing mid-air collisions. However, its deterministic logic limits robustness in the presence of unanticipated pilot responses, as exposed by the collision of two aircraft in 2002 over Überlingen, Germany [4]. To increase robustness, Lincoln Laboratory has been developing a new system, ACAS X, which uses probabilistic models to represent uncertainty. Simulation studies with recorded radar data have confirmed that this novel approach leads to a significant improvement in safety and operational performance. The Federal Aviation Administration (FAA) has formed a team of organizations to mature the system, aiming to make ACAS X the next international standard for collision avoidance.

The adoption of a completely new algorithmic approach to a safety-critical system naturally poses a significant challenge for verification and certification.

⋆ The first author performed this work while employed by SGT Inc. as an intern at the NASA Ames Research Center. This work was funded under the System-wide Safety Analysis Technologies Project of the Aviation Safety Program, NASA ARMD.

E. Ábrahám and K. Havelund (Eds.): TACAS 2014, LNCS 8413, pp. 620–635, 2014.

Our goal in this work is to study the applicability of formal probabilistic verification and synthesis techniques, which go beyond simulation studies [8,5]. Our study was driven by tasks defined in collaboration with the ACAS X team to be complementary to their verification efforts. During the course of our work, we identified shortcomings of existing tools, which lead us to develop a framework customized for ACAS X (or similar systems). In our framework, models are expressed in a traditional programming language for increased expressiveness, and verification and synthesis algorithms are designed for scalability and efficiency.

The contributions of this work can be summarized as follows: 1) Development of a faithful model for synthesis of the ACAS X controller, based on the Lincoln Laboratory publications [6]; 2) Development of customized verification and synthesis algorithms for efficient handling of ACAS X (and like) systems; 3) Identification of design and verification challenges for ACAS X as related to probabilistic verification and synthesis; 4) Results obtained from the application of our framework to ACAS X and recommendations for the ACAS X effort.

The results of our work will serve as input for the certification of ACAS X. Due to access restrictions, we analyze a previous version of the system [6], but are currently working with the ACAS X team to extend our work to the current version. We believe that ACAS X presents researchers in probabilistic verification and synthesis with a unique opportunity to focus on a relevant, safety-critical case study. For this reason, we are preparing a public release of our models and framework, to encourage other members of the community to build on our work.

The remainder of this paper is organized as follows. Section 2 describes the ACAS X system as designed and deployed by the ACAS X team. In addition to these techniques, our work implements and applies formal verification and synthesis approaches, described in Sections 3 and 4. We discuss implementation details in Section 5, with Section 6 concluding the paper.

2 The ACAS X System

Model Description. Similarly to the current standard TCAS, ACAS X [6] uses several sources to estimate the current state of the plane on which it is deployed, and the planes in its vicinity. If it detects the possibility of an imminent collision (less than 40 seconds away), it produces vertical maneuver advisories (to climb or descend) in order to avoid the collision. Both TCAS and ACAS X operate at a frequency of one state update and advisory per second.

The ACAS X model consists of two airplanes on collision course. Loss of Horizontal Separation, from now on denoted as `LHS`, describes the situation where two airplanes are in the exact same location when their height difference is ignored. A Near Mid-Air Collision (NMAC) occurs when the two airplanes are within 100 ft of each other when `LHS` occurs. We refer to the plane equipped with ACAS X as *our* plane (often referred to as ownship in the literature), and the other plane as *intruder* (similarly to [6]).

The model has 5 parameters: (1) $h \in [-1000, 1000]\,\mathsf{ft}$, the height difference between the two planes, (2) $dh_0, dh_1 \in [-2500, 2500]\,\mathsf{ft}\,/\,\mathsf{min}$, our and the intruder's climbing rates (3) adv the advisory produced by ACAS X one second

ago (4) ps the pilot state. Pilot state and advisories can take the following values — note that the pilot can either follow the advisory (i.e., ps = adv) or perform random maneuvers (i.e., ps = COC), since studies have shown that pilots may not react immediately or at all to an advisory:

- COC stands for "clear of conflict" — the pilot is free to choose how to control the plane.
- CLI1500 / DES1500 stand for "climb / descend with 1500 ft / min", respectively; they advise the pilot to change the climbing rate with $\frac{1}{4}g$ until reaching a climbing rate of 1500 ft / min / −1500 ft / min, respectively.
- Advisories SCLI1500 / SDES1500 and SCLI2500 / SDES2500 are similar but employ an acceleration of $\frac{1}{3}g$. Moreover, SCLI2500 / SDES2500 target a final climbing rate of 2500 ft / min / −2500 ft / min, respectively.

In describing the dynamics of the system, we use $X \sim P$ to denote that X is sampled according to probability distribution P. Moreover, $N(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ. Lastly, we denote by $\{p_1 : e_1, p_2 : e_2, \ldots\}$ the distribution in which e_i has probability p_i. Given a state $(dh_0, dh_1, h, \text{adv}, \text{ps})$ and an advisory a, the dynamics of the system are given by the following equations, which together describe a continuous probability distribution $\delta_c(dh_0', dh_1', h', \text{adv}', \text{ps}' \mid dh_0, dh_1, h, \text{adv}, \text{ps}, a)$, where the primed versions of variables (e.g., dh_0') characterize the next state. In these equations, function f returns the appropriate acceleration in ft/s^2 if the desired climbing rate has not been reached yet, and 0 otherwise.

$$\text{adv}' = a; \quad dh_1' \sim dh_1 + 60N(0,3); \quad h' = h + ((dh_0 + dh_0')/2 - (dh_1 + dh_1')/2)/60$$

$$\text{ps}' \sim \begin{cases} \{1 : a\} & \text{if } a = \texttt{COC} \vee a = \text{ps} \\ \{\frac{1}{6} : a, \frac{5}{6} : \texttt{COC}\} & \text{if } a \in \{\texttt{CLI1500}, \texttt{DES1500}\} \wedge a \neq \text{ps} \\ \{\frac{1}{5} : a, \frac{4}{5} : \texttt{COC}\} & \text{if } a \in \{\texttt{SCLI*}, \texttt{SDES*}\} \wedge a \neq \text{ps} \end{cases}$$

$$dh_0' \sim dh_0 + \begin{cases} 60N(0,3) & \text{if } \text{ps}' = \texttt{COC} \\ \{1 : \text{f}(dh_0, \text{ps}')\} & \text{otherwise} \end{cases}$$

Model Discretization. Similarly to [6], we generate an ACAS X controller by analyzing a Markov Decision Process (MDP) obtained through discretization of the above model. In our implementation, the number of discrete values that replace each continuous parameter is configurable by a *resolution* vector $(r_{dh_0}, r_{dh_1}, r_h)$, where r_{dh_0}, r_{dh_1}, r_h define the number of points below and above 0 used to discretise dh_0, dh_1, h, respectively. Formally, the set of discretization points is defined as $\text{D}_{r_{dh_0}, r_{dh_1}, r_h} = \{-2500, -2500 + 2500/r_{dh_0}, \ldots, 2500\} \times \{-2500, -2500 + 2500/r_{dh_1}, \ldots, 2500\} \times \{-1000, -1000 + 1000/r_h, \ldots, 1000\}$. The resolution of the controller defined in [6] is $(10, 10, 10)$.

The following two techniques are then employed in [6] to calculate the transition distribution over $\text{D}_{r_{dh_0}, r_{dh_1}, r_h}$. Instead of sampling from the continuous normal distribution $(N(0,3), N(0,3))$ for equations dh_0' and dh_1', we sample from the distribution $\{\frac{1}{6} : (0, \sigma), \frac{1}{6} : (0, -\sigma), \frac{1}{3} : (0, 0), \frac{1}{6} : (\sigma, 0), \frac{1}{6} : (-\sigma, 0)\}$, where

$\sigma = 3\sqrt{3}$. This is called sigma point sampling. After having modified the equations with sigma point sampling, we obtain a discrete probability distribution $\delta'(dh_0', dh_1', h, \mathrm{adv}', \mathrm{ps}' \mid dh_0, dh_1, h, \mathrm{adv}, \mathrm{ps}, a)$.

Secondly, linear interpolation matches the points of δ' to the discretization points in $\mathrm{D}_{r_{dh_0}, r_{dh_1}, r_h}$. Let Δ_{dh_0} be the distance between two discretization points of the climbing rate of our plane, and let Δ_{dh_1} and Δ_h be defined analogously. We define function ι to capture how "close" a point (dh_0, dh_1, h) is to a discretization point (dh_0', dh_1', h') immediately surrounding it as

$$\iota((dh_0, dh_1, h), (dh_0', dh_1', h')) = (1 - \frac{|dh_0 - dh_0'|}{\Delta_{dh_0}})(1 - \frac{|dh_1 - dh_1'|}{\Delta_{dh_1}})(1 - \frac{|h - h'|}{\Delta_h}),$$

and 0 for all other points. Based on these, we define the transition relation as $\delta(s^d \mid s, a) = \sum_{s'} \delta'(s' \mid s, a)\, \iota(s', s^d)$.

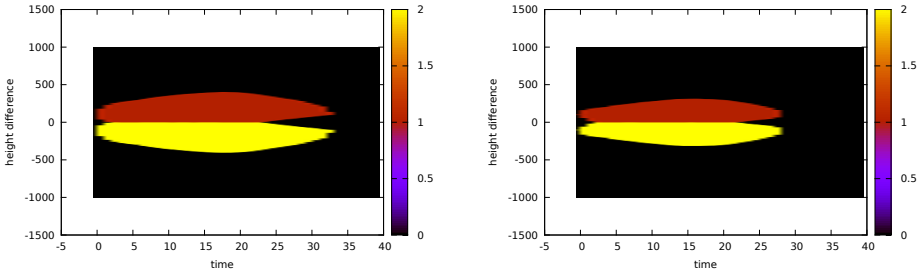
Controller Generation. In order to generate a controller, each ACAS X advisory receives a cost/reward, where costs are rewards with negative values. Reward `COC` is associated with switching from any alerting state to `COC`; `Alert` is a cost associated with switching from `COC` to either `CLI1500` or `DES1500`; `Reversal` is a cost associated with switching from any climbing to any descending advisory, or vice versa; `Strengthening` is a cost associated with switching from any climb/descent advisory with goal 1500 ft / min to `SCLI2500`/`SDES2500`, respectively; `NMAC` is a cost associated with the occurrence of an NMAC.

We henceforth refer to the costs/rewards as "weights", thus describing the fact that they capture the relative importance of different quality criteria of the controller. Let $c(s, a)$ be the sum of costs and rewards ACAS X receives for selecting advisory a in state s (for example, if a in state s activates an alert, then $c(s, a) = \texttt{Alert}$). Moreover let $\mathbb{E}_{\delta(s' \mid s,a)}[\alpha(s')]$ describe the expected value of some function α under the probability distribution over the successor states s' of s when selecting action a. We then calculate a table equivalent to the family of functions $T_t(s, a) := c(s, a) + \mathbb{E}_{\delta(s' \mid s,a)}[\min_{a' \in A(s')} T_{t-1}(s', a')], 1 \leq t \leq 40$, where $A(s)$ stands for the set of advisories admissible in s. Further, $T_0(s, a) = \texttt{NMAC}$ if s models an NMAC, and 0 otherwise. Essentially, for each state and each advisory the table stores the expected accumulated cost.

Controller Deployment. The generated controller is deployed as look-up table $T_t(s, a)$ described previously. Linear interpolation is used to determine the advisory for a state s in the continuous world at time t until loss of horizontal separation by: $\arg\min_{a \in A(s)} \sum_{s' \in \mathrm{D}_{r_{dh_0}, r_{dh_1}, r_h}} \iota(s, s') T_t(s', a)$.

Figure 1(a) illustrates a part of the interpolated strategy generated according to [6]. In the figures, note that `LHS` occurs at time 0, located on the left hand side of the plots, so time in the plots flows from right to left. Thorough examination of such plots is part of the validation of ACAS X but goes beyond the scope of this paper. Our framework can easily generate such plots, though.

We would like to point out two features of the generated controller. Firstly, if the airplanes start out on the same height, then the controller waits for a long time until giving an advisory, as witnessed by the black space between the two "tails" on the right. This is because it is very unlikely that the two planes will



(a) Resolution $(10, 10, 10)$ (b) Resolution $(20, 20, 20)$

Fig. 1. Two controllers generated with the same weight in different resolutions. x-axis shows time until `LHS`, y-axis height difference. Parameters dh_0 and dh_1 are zero throughout, and adv = ps = `COC`. Color indicates selected advisory: black (0) for `COC`, red (1) for `CLI1500`, yellow (2) for `DES1500`.

remain on the same height for a long time (due to their random movement), and it is therefore better to wait until the intruder either starts climbing and or descending and go in the opposite direction. Secondly, notice the "mouth" shape close to time 0 and around height difference 0. In this collision situation, ACAS X is not giving any advisory, although one would intuitively expect that *some* advisory would be more informative to the pilot than `COC`, which may be misleading. This is an artifact of the costs used for synthesis, and we describe a technique that identifies situations like these in Section 3.

3 Verification

To complement the ACAS X work that primarily uses simulation, we apply formal analysis techniques to evaluate the ACAS X controller. Simulation-based techniques are studied and discussed in Section 4, where we explore the design-space of controllers and compare different generated controllers among themselves. In this section, we evaluate the ACAS X controller 1) in terms of the quality criteria used for its generation, and 2) through model checking of PCTL [3] properties, which are ideal for probabilistic models such as ACAS X's. For evaluation, we use models discretized at different resolutions, and could even use different model characteristics and parameters (although we do not do the latter in the experiments presented here).

The type of analysis that we perform provides a value $v(s)$ for each state of the discretized model. To easily compare results of analyses with each other and with simulations, we define a probability distribution $I(s)$ over the states of the model as follows (similarly to [6]). The only states we consider are those at 40 seconds from `LHS`, and in which ps = adv = `COC`. Over those states, we first define a continuous distribution over $(dh_0, dh_1, h) \in \mathbb{R}^3$ by sampling dh_0 and dh_1 uniformly from $[-1000, 1000]\,\mathsf{ft}\,/\,\mathsf{min}$, denoted as $dh_0 \sim U(-1000, 1000)$ and $dh_1 \sim U(-1000, 1000)$. To make a collision likely, and therefore to provoke the controller into action, h is sampled from $40((dh_1 - dh_0)/60) + N(0, 25)$.

To define an analogous distribution of $\mathrm{D}_{r_{dh_0}, r_{dh_1}, r_h}$, we assign probability masses to all three parameters so as to soak up the probability of the space around them. That is, the probability of picking sample point dh_0 is defined as: $\mathrm{P}(dh_0 - \Delta_{dh_0}/2 \leq H_0 \leq dh_0 + \Delta_{dh_0}/2)$, with $H_0 \sim U(-1000, 1000)$. Note that $\Delta_{dh_0}, \Delta_{dh_1}$ and Δ_h are defined as in Section 2. We define the discretized probability of dh_1 analogously. The discretized probability of h is defined as: $\mathrm{P}(h - \Delta_h/2 \leq H \leq h + \Delta_h/2)$, where $H \sim 40((dh_1 - dh_0)/60) + N(0, 25)$, i.e., the probability distribution of h depends on dh_0 and dh_1. We then use I to calculate the expected value $\mathbb{E}_{I(s)}[v(s)]$.

3.1 Influence of Resolution on Controller Evaluation

Our first step in evaluating the ACAS X controller involves calculating its performance in evaluation models of different resolutions for the two climbing rates and the height difference: $(10, 10, 20), \ldots, (10, 10, 50)$, $(20, 20, 10) \ldots (50, 50, 10)$ and $(20, 20, 20) \ldots (50, 50, 50)$.

For each of these resolutions, Figure 2 presents the evolution of the probability of seeing an NMAC versus the resolution. The three lines represent the three groups of increasing resolutions. Line "Height" represents resolutions $(10, 10, n)$, while line "Climbing Rate" represents the resolutions $(n, n, 10)$ and line "All" represents the resolutions (n, n, n), for $n \in \{10, 20, 30, 40, 50\}$.

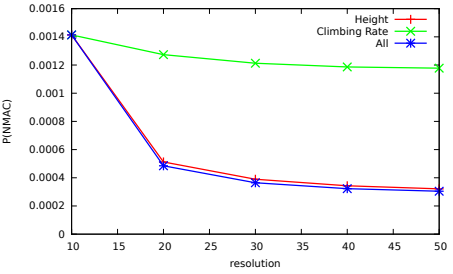


Fig. 2. P(`NMAC`) of baseline controller in various resolutions

These plots indicate that the probability of NMAC drops as we increase resolution. This in turn indicates (though does not guarantee) that a coarse resolution provides a conservative estimate for the quality criteria of the controller. Lines "Height" and "Climbing Rate" indicate that increasing the resolution of the height difference has a stronger influence on the quality of the analysis than the resolution of the climbing rate. This observation is reinforced by comparing lines "Height" and "All". The difference between these two lines is small, despite the fact that an n-fold increase in resolution of the climbing rate leads to an n^2-fold increase in state space.

3.2 PCTL Model Checking

The PCTL model checking engine that we have developed enables users to: (1) vary the resolution of the model to get more precise results, and (2) analyse non-trivial properties expressed in the PCTL formal property language. In contrast to simulation, PCTL model checking allows an exhaustive search of the state space and can thus uncover scenarios that simulations might easily miss. This is important given the low probability of some of the properties we want to check.
Property 1: Near Mid-Air Collision. Studies the probability of a near

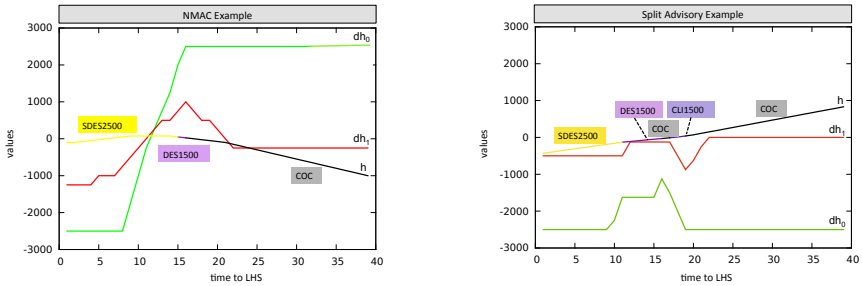


Fig. 3. Trace plots for properties 1 and 3. x-axis displays time to LHS, y-axis displays values of (dh_0, dh_1, h). The color of line h depicts the advisory, tagged above the line.

mid-air collision, formally $P_{=?}[\mathsf{F}\,\mathtt{NMAC}]$. During analysis, we observed that the most likely cases of this undesirable scenario stem from late reactions from the pilot. We therefore decided to instead concentrate on NMACs that occur despite immediate reactions to advisories by the pilot. We formulate this as $P_{=?}(\mathsf{F}\,\mathtt{NMAC} \mid \mathsf{G}\,\mathrm{adv} = \mathrm{ps})$, i.e., what is the probability of reaching an NMAC state although the pilot always reacts immediately.

The highest probability over all initial states that we encounter with the conditional probability formula is $2.30 \cdot 10^{-8}$, as opposed to $6.92 \cdot 10^{-4}$ with the original formula. This confirms that the vast majority of NMACs happen because the pilot does not react fast enough or at all. To understand the NMACs that occur despite the fact that the pilot reacts to advisories, we analyzed some traces that are most likely to fulfill $P_{=?}(\mathsf{F}\,\mathtt{NMAC} \mid \mathsf{G}\,\mathrm{adv} = \mathrm{ps})$. Figure 3 depicts such a scenario: initially, our airplane is 1000ft below the intruder and we are climbing with $2500\,\mathsf{ft}\,/\,\mathsf{min}$. The intruder, on the other hand, starts out with a climbing rate of $-250\,\mathsf{ft}\,/\,\mathsf{min}$. Until 22 seconds to LHS, the two airplanes maintain their course, and therefore the height difference shrinks. If both planes were to continue to maintain their course, then our plane would be well above the intruder at time 0 to LHS, so ACAS X does not alert.

At this point, climbing rate of the intruder starts increasing, and the vertical distance becomes $-150\,\mathsf{ft}$. The height difference levels off as a result of the intruder's increase in climbing rate from now on. ACAS X signals the DES1500 advisory seven seconds later, and SDES2500 one second after that. As a result, our airplane starts descending steeply until it reaches $-2500\,\mathsf{ft}\,/\,\mathsf{min}$. At the point of the first alarm, the vertical distance is $50\,\mathsf{ft}$, i.e., our plane is slightly above the intruder. Unfortunately, the climbing rate of the intruder starts decreasing at exactly the same point and from that point on, the two climbing rates are not different enough to carry our plane outside of the danger zone and we end up with a vertical distance of $100\,\mathsf{ft}$, and hence an NMAC.

Traces like these capture exactly the type of unforeseen behaviour that led to the Überlingen accident [4], and probabilistic model checking can detect cases like these easily. We consider it encouraging that the most likely case of collision requires relatively complex behaviour of the intruder (first increasing the climbing rate, then decreasing it, at exactly the right point in time).

Property 2: No advisory despite collision. Studies the probability of issuing no advisory although a future NMAC is likely, formally $\mathrm{P}_{=?}[\mathsf{F}(\mathrm{P}_{=1}[\mathsf{X}\,\mathtt{COC}] \wedge \mathrm{P}_{>0.1}[\mathsf{F}\,\mathtt{NMAC}])]$. This formula was motivated by our previous observation of Figure 1(a) in Section 2, according to which there is an area where ACAS X issues no advisory although an NMAC is imminent. Figure 4 shows the probability of the formula for all states in which $dh_0 = dh_1 = 0\,\mathsf{ft}\,/\,\mathsf{min}$ and adv = ps = COC. This probability is 1 until about 12 seconds away if the height difference between the planes is less than a 100 ft. Model checking the formula, however, reveals that among all initial states, the highest probability is 0.3%, so getting into such a situation is improbable.

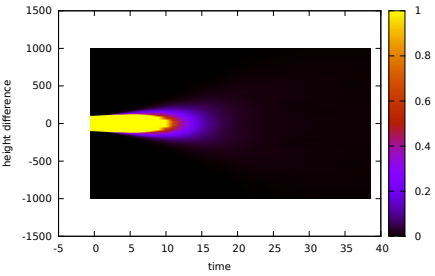
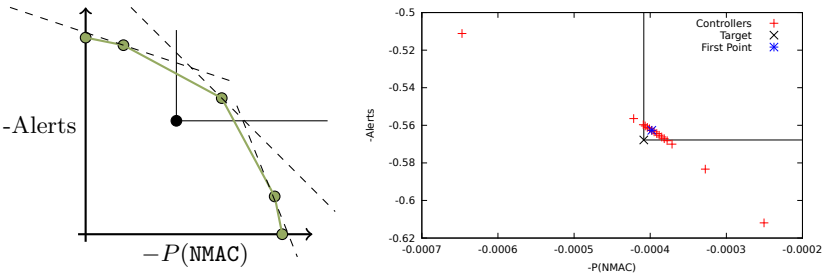


Fig. 4. Probability of fulfilling property 2. Plot parameters as in Figure 1(a); color depicts probability

Property 3: Split Advisory. Studies the probability of issuing an alert, switching it off, and then switching an alert on again (a *split advisory*), formally $\mathrm{P}_{=?}[\mathsf{F}(\neg\,\mathtt{COC} \wedge P_{=1}[\mathsf{X}\,\mathtt{COC}] \wedge P_{>0}[\mathsf{F}\,\neg\,\mathtt{COC}])]$. Even though during controller generation ACAS X penalizes reversals, these costs only reflect immediate changes in controller advisories. Split advisories are also undesirable, but are harder to capture during controller generation. The PCTL property described above can however be used to study how likely such situations are. Analysis of the model checking results revealed that a main cause for such situations is the pilot not following the advisory. We therefore refined the property similarly to Property 1, by checking cases where split advisories occur under the condition that the pilot always reacts immediately to advisories.

Figure 3 depicts a split advisory scenario under the refined property. Initially (at 40 seconds to LHS), our plane is 830 ft above the intruder and descending with 2500 ft / min, while the intruder is in level flight. The vertical distance is therefore decreasing. Around 19 seconds into the scenario, the intruder starts descending, and soon after, ACAS X advises CLI1500 and maintains this advisory for 2 seconds, before switching it off again. Accordingly, the rate of descent of our plane gradually reduces to 1500 ft / min. The advisory is then switched off, as the intruder stops descending, effectively moving out of the way of our plane. ACAS X switches to COC but, a second later, gives advisories DES1500, followed by SDES2500, as the intruder's rate of descent increases again.

Let us further analyze this generated scenario. The first climb advisory aimed at avoiding a collision that would be likely if our plane continued to descend at the same rate. It could not force the pilot to increase the rate of descend further, since 2500 ft / min already is the maximum. Therefore, climbing was the only possibility. Then the intruder stopped descending, which reduced the probability of colliding with our current climbing rate. This may have caused ACAS X to shut the advisory off. Shortly before ACAS X switched the advisory back on, the



(a) Simplified Pareto curve (b) Points generated for two objectives

Fig. 5. Fictional and actual Pareto fronts

difference in climbing rates was 1000 ft / min, and the height difference was -30 ft. Since we were about 15 seconds away from LHS, this amounted to a decreased vertical distance of about 260 ft. ACAS X decided to increase the vertical distance by increasing the rate of descent.

It would be interesting to study whether the cost function of ACAS X may encourage such cases of split advisories. Given that (Alert + COC < Reversal), it is possible that ACAS X decided to gain a small reward for selecting COC after the first advisory, and additionally avoid the cost of a reversal that would be incurred if the advisory was switched directly from a climb to a descend.

4 ACAS X Design Challenges

The generation of the ACAS X controller depends on two major design issues that have so far been unexplored: the selection of weights, and the discretization resolution. As reported in [6], the weights were selected based on an intuition of the relative importance of the different quality criteria. In this section, we study more systematic techniques for selecting controller weights, and investigate how discretization resolution influences the generated controller.

4.1 Generating Controller Weights

Our goal is to systematically explore deterministic controllers whose performances exceed requirements on NMAC, Alert, etc., provided by domain or certification experts. We refer to these requirements as "targeted performance", or simply "target". Central to achieving this goal is an existing result, which states that the performance of all controllers that can be generated by weights form a convex Pareto front [2]. The Pareto front is n-dimensional, where n is the number of costs/rewards. The performances of all possible controllers (even controllers using randomization and memory) lie on the inside of the Pareto front.

For example, Figure 5(a) illustrates a two-dimensional Pareto front. The performance of all deterministic controllers (green dots in the plot), lie on the vertices of the Pareto front. The targeted performance is depicted as a black dot in Figure 5(a). The box with a lower left corner at this target and extending to

infinity in all dimensions, defines the section of the Pareto front in which we are interested. To find this section, we modified an algorithm presented in [9].

While the details of the approach are beyond the scope of this paper, the idea can be summarized as follows. Initially, the optimal controller for each dimension is generated, i.e., the controller with the lowest P(`NMAC`), the controller with the lowest expected number of `Alert`s (i.e., zero), etc. We add the performance of these controllers to the approximation of the Pareto front. These points, illustrated as the two green dots on the axes in Figure 5(a), reflect the performance of the corresponding controller in terms of the selected quality attributes.

We then keep adding points to the Pareto front in the following way. We calculate the convex hull of the points generated so far. This hull defines a set of n-dimensional faces (lines, in our picture), that connect the points. Further, the hull defines a lower bound for new points (the Pareto front is convex, so missing points must lie on or above the hull). In the picture, the lines connecting the green dots form the hull. The generated points also define an upper bound on the space of controller performances, illustrated by the dashed lines in the figure. The direction (normal) of the dashed line (separating hyperplane) is given by the weights we used to generate the point. If there are any more points we can generate, then these points exist between the hull and the upper bound.

Since we want to find new points in the box defined by the target, we pick new weights so as to refine the face (by lowering the upper bound or breaking up the face) above which there is a point that 1) lies inside the upper bound 2) lies above the target 3) is maximally far away from the face (as defined by the Euclidean distance). We continue until we either prove that the target lies outside the upper point (which means that no controller fulfilling the minimal requirement exists) or until we have found enough points above the target.

Figure 5(b) presents a subset of the points generated by this approach on `Alert` and `NMAC` exclusively. The target point and the box it defines are plotted in black, and the points generated are plotted in red. The algorithm first generated 8 points outside the box. The first point generated within the target box (the 9th overall) is depicted in blue. We generated 10 more points after we found it. We note that all subsequent 10 points that are generated also lie within the box. The same effect has been observed for three dimensions. We conclude that this algorithm is good at approximating the interesting part of the Pareto front (that inside the box) once it finds the first point that meets the target specifications.

We have checked this algorithm against various targets, and it always either finds a controller meeting the requirement, or proves that no such controller exists. Note that finding a controller in the box is an NP-complete problem (easy adaptation of proof from [1]). In the worst case, the algorithm has to generate all points of the Pareto front of the model, of which there are exponentially many. However, as the next section shows, little more than 100 points suffice to find a controller meeting the requirement for various resolutions.

We believe that this technique can be very helpful as the controller model ACAS X evolves. Each evolution (be it a change in discretization or a change in parameters), necessitates tuning weights anew (as witnessed by the

first experiment in the next section). Our approach allows to semi-automatically select these weights by presenting domain experts with the trade-offs. They can then select a controller they deem sufficient, or select an area for further refinement.

4.2 Discretization Resolution

To study the effects of discretization resolution on the quality of the obtained controller, we designed a number of experiments described in this section. We will from now on refer to the controller presented in [6] as the "baseline" controller.

Experiment 1. This experiment aims to analyze the performance of controllers generated at resolutions $(20, 20, 20)$, $(30, 30, 30)$, $(40, 40, 40)$ and $(50, 50, 50)$, using the weights of the baseline controller. Our expectation was that a higher resolution would lead to a better performance, at least in terms of P(`NMAC`). However, the experiments showed that the controllers we generate by this method do not necessarily perform better in all the quality attributes. Instead, higher resolution controllers have a significantly higher P(`NMAC`) and significantly fewer alerts than the baseline controller in the same resolutions.

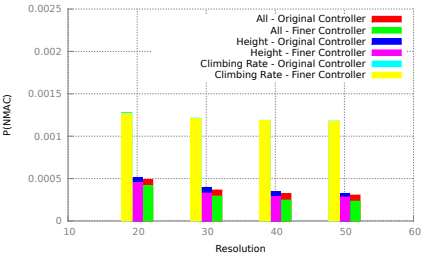
The reason becomes clear when we consider the controller plots in Figure 1(a) and Figure 1(b). The area in which an alert is signalled by the controller is significantly smaller in Figure 1(b) when compared to Figure 1(a). To understand the reason for this effect, we analyzed the controllers using the techniques from Section 3. It turns out that controllers in higher resolutions indeed perform better in the sense of having a higher expected reward than the baseline controller. Intuitively, the controllers use the additional information they receive from a higher resolution to improve the score they receive. To this end, the controllers improve their score by reducing the expected number of alerts, at the cost of a higher P(`NMAC`).

This experiment made it clear to us that weights may balance out the quality attributes of a controller differently, when different resolutions are considered. As a consequence, we believe that it is more meaningful to systematically explore the design space of controllers based on specific target quality attributes, as presented in Section 4.1. One could then compute weights based on these target values, and within the resolution where the generation will occur.

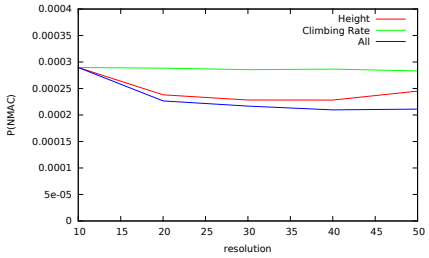
Experiment 2. Given the first experiment, we decided to study whether it is possible to generate controllers that are better than the baseline controller in all quality attributes, in higher resolutions. To generate a controller that performs better than the baseline controller in a given resolution $R = (r_h, r_{dh_0}, r_{dh_1})$, we first evaluate the performance of the baseline controller in resolution R. The result is a vector $v = (\texttt{NMAC}, \texttt{Alert}, \texttt{Strengthening}, \texttt{Reversal}, \texttt{NMAC})$, which summarizes the performance of the baseline controller when model checked in resolution R (see Section 3 for more details). We then use the technique described above to approximate the Pareto front above v. From the generated controllers that meet the specification, we then pick the one with the lowest P(`NMAC`).

Figure 6(a) illustrates the obtained results. The bars show, for resolution factor n the performance of the baseline controller when checked against resolutions

$(n, n, 10)$ (Climbing Rate), $(10, 10, n)$ (Height) and (n, n, n) (All) respectively. It can be seen that we were almost unable to decrease P(NMAC) using the climbing rate alone. The relative performance of these controllers is consistently around 99.5%. When we increase the resolution of the height, then we get a relative performance of about 85%. Finally, when increasing the resolution of both we see a relative performance of about 83%. As witnessed in Section 3, the discretization of height seems to have the biggest influence on controller quality. Interestingly, the relative performance does not improve as we increase the resolution.



(a) Controller quality vs resolution



(b) Controller quality checked in (50,50,100)

Fig. 6. Plots for Experiment 3

To further judge the quality of the generated controllers, we checked them against resolution $(50, 50, 100)$ and present the results in Figure 6(b). On the x-axis, we have the controller resolution, while on the y-axis we have the probability of a Near Mid-Air Collision. As before, "Height" stands for the controllers of resolution $(10, 10, n)$, "Climbing Rate" for the controllers of resolution $(n, n, 10)$ and "All" for the controllers of resolution (n, n, n). This experiment confirms that increasing the resolution of the height difference between the two planes has the most impact up to and including $(10, 10, 30)$, after which we notice no further improvement. In contrast to this, we notice further improvements in category "All". Our experiments indicate that the best ratio of resolution for the three parameters is $(n, n, 3 \cdot n)$.

Experiment 3. Let $v_R(c)$ denote the quality vector of a controller c in resolution R (i.e., the vector of P(`NMAC`), P(`Alert`), *etc*). We organized this experiment to study if $\forall c_1, c_2, R_1, R_2 : v_{R_1}(c_1) \geq v_{R_1}(c_2) \wedge R_2 > R_1 \implies v_{R_2}(c_1) \geq v_{R_2}(c_2)$ holds. To this end, we compared the performance of the controller we generated in resolution $(20, 20, 20)$ to the baseline controller in resolutions $(20, 20, 20)$ and $(50, 50, 100)$, and present the results in the following table. Note that the higher resolution controller performs better than the baseline in all dimensions in resolution $(20, 20, 20)$; specifically, it is very close to the target performance in everything but `NMAC`, where it is notably better.

This attests to the efficacy of our Pareto front algorithm. When comparing this to the analysis results in resolution $(50, 50, 100)$, we observe that while the higher resolution controller and the baseline controller are still very close in all

	NMAC	Alert	Strengthening	Reversal	COC
$(10, 10, 10)$ in $(20, 20, 20)$	$-4.850 \cdot 10^{-4}$	-0.6310	-0.083	-0.019	0.629
$(20, 20, 20)$ in $(20, 20, 20)$	$-4.186 \cdot 10^{-4}$	-0.6306	-0.081	-0.019	0.631
$(10, 10, 10)$ in $(50, 50, 100)$	$-2.897 \cdot 10^{-4}$	-0.6245	-0.078	-0.020	0.622
$(20, 20, 20)$ in $(50, 50, 100)$	$-2.313 \cdot 10^{-4}$	-0.6308	-0.078	-0.019	0.630

characteristics except NMAC, the higher resolution controller is no longer strictly better in all dimensions. For example, it uses slightly more alerts and slightly more reversals. This is offset by the fact that the P(NMAC) of the higher resolution controller is still significantly better than that of the baseline controller. To summarize, the general tendencies of the relation of controllers when checked in higher resolutions are the same, but the exact relations are not preserved.

4.3 Bayesian Model Checking

In this section, we evaluate the generated controllers using simulation (where discretization is not required), and compare the results with model checking. To this aim, we implemented a parallel Bayesian model checking engine [10], which simulates the system based on the dynamic equations of Section 2. We used the same initial distribution as [6], described in Section 3. In [6], the authors also report on the use of a Bayesian network instead of the dynamic equations.

This approach allows us to run simulations, and state "given the traces observed, the probability that property φ holds lies in interval $[a, b]$ with confidence c. The level of confidence and the size of the interval are configurable. In the following, we use this framework to estimate the probability that an NMAC happens when using the baseline controller, and compare the results to Experiment 2. Our analysis reports that the probability of NMAC lies in range $[2.48 \cdot 10^{-4}, 2.58 \cdot 10^{-4}]$ with probability 95%. We needed to generate 38,796,000 samples to get this level of confidence for the given interval size.

We additionally applied this simulation technique to controllers of resolution $(10, 10, 10), \ldots, (10, 10, 50)$ generated previously. The following table presents the probability of seeing an NMAC for each of them.

Resolution	10	20	30	40	50
$\mathrm{P}(\texttt{NMAC}) \cdot 10^4$	[2.51, 2.61]	[2.17, 2.27]	[2.08, 2.18]	[2.12, 2.22]	[2.27, 2.37]

We conclude that the trend follows that depicted in Figure 6(a): improvements in performance are significant until we reach resolution $(10, 10, 30)$, at which point they taper off. We were unable to perform this analysis on controllers with resolution larger than $(20, 20, 20)$ because we could not fit the whole table into memory at once. For $(20, 20, 20)$, though, we receive $\mathrm{P}(\texttt{NMAC}) \in [2.06 \cdot 10^{-4}, 2.16 \cdot 10^{-4}]$, i.e., a number very close to that of the controller generated for $(10, 10, 30)$.

5 Implementation

We originally used existing probabilistic model checking tools for ACAS X but encountered several limitations. First, we could not express the linear interpolation needed in the controller evaluation. Second, we not only require capabilities

for the specification of a model, but also for loading generated controllers for subsequent verification. Last but not least, for our mupliple experiments involving increasing resolution, the state spaces we generate grow prohibitively large, and there is a considerable slow-down that could benefit from parallelization, which is unavailable in current releases of existing tools.

More specifically, the size of the controller has $40 \cdot ((2r_{dh_0} + 1) \cdot (2r_{dh_1} + 1) \cdot (2r_h + 1) \cdot 13)$ states in resolution $(r_{dh_0}, r_{dh_1}, r_h)$. So, for example, the model from [6] has $4,815,720$ states overall. A controller with resolution $(50, 50, 50)$ has $535,756,520$ states. We wrote a simplified version of the model in [6] for PRISM [8] (without linear interpolation, but with sigma point sampling). While PRISM succeeded in loading the model as a BDD model, analyzing it was not possible (we aborted conversion to the hybrid representation after 10 min).

These problems motivated us to create our own framework that takes advantage of two key insights into the ACAS X model. Firstly, if we want to calculate the values of any property in this model at time t, then we only need to keep the value of time $t - 1$ in memory. This alone leads to a reduction of memory consumption to 2.5%. Secondly, since we need to calculate value iteration steps only a relatively small number of times for each state, it is possible to avoid storing the transition matrix in memory and generate the values on-demand.

In addition, we parallelized value iteration, and the speed-up obtained in experiments using up to 12 cores was almost linear (1.94 for 2 cores, 3.37 for 4 cores, 4.67 for 6 cores, 6.47 for 8 cores, 7.54 for 10 cores, 8.93 for 12 cores). Parallelization proved essential for our experiments involving increasing discretization resolution; generating the Pareto fronts for all cases took about 2 days, as opposed to more than a month.

6 Conclusions and Future Work

ACAS X is a safety-critical system that the FAA plans on introducing as the new standard for collision avoidance. The system that will be deployed is the look-up table that is generated by the techniques described in [6]. It is therefore reasonable that a large number of the verification efforts would focus on the verification of the generated controller in operation. However, we believe that it is meaningful to take advantage of the existence of models for additional formal analysis both of the controller itself, and of the design choices.

Our experiments related to the effects of resolution on controller generation were particularly interesting. For example, we observed that height discretization is more effective that climbing rate alone, when exploring the space of controllers better than a particular target. We therefore recommend increasing height resolution first, when there is an upper bound in controller size that does not allow for uniform discretization of all variables. In the future, we intend to carry out more experiments in this domain in order to give more precise recommendations.

Some of the results that we obtained were also unexpected: the fact that a higher resolution may balance the weights of quality attributes differently and therefore result in a drop in performance of NMAC; or the fact that the relative performance of two controllers may change when moving to higher resolutions.

This cautions us, in exploring the space of controllers, to ultimately evaluate their relative performance in simulation. However, the Pareto-front-based techniques for controller generation provide a systematic way of generating and comparing controllers that can complement designer intuition.

PCTL model checking also proves valuable in studying properties of generated controllers. However, more useful than the model checking itself, is the capability to visualize its results and generate traces that help with understanding of the model checking results. We therefore found that latter aspect of our tools most helpful, together with a simulator that we built, which allows to interactively explore generated controllers. In the future, we plan to connect the simulator to the model checker, to allow replay of the generated traces.

The techniques and tools that we developed are general, and the customization for memory savings is applicable to problems that have a similar nature; for example, it could be used in the domain of car collision avoidance systems, which is important as we move towards self-driving cars. Our work on analysis of ACAS X will continue beyond this paper. Our plans for future work include the modeling of a reasonably adversarial pilot for the intruder plane, and alternative representations of the look-up table for verification and deployment. Moreover, we plan to study a version of ACAS X that is targeted to unmanned vehicles, as well as experiment with the evaluation of generated controllers in the context of hybrid verification tools, which the ACAS X team has expertise in.

Acknowledgement. We thank Guillaume Brat, and members of the ACAS X team Ryan Gardner, Mykel Kochenderfer and Yanni Kouskoulas, for valuable discussions and feedback.

References

1. Chatterjee, K.: Markov decision processes with multiple long-run average objectives. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 473–484. Springer, Heidelberg (2007)
2. Forejt, V., Kwiatkowska, M., Parker, D.: Pareto curves for probabilistic model checking. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 317–332. Springer, Heidelberg (2012)
3. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 102–111 (1994)
4. Johnson, C.: Final report: review of the BFU Überlingen accident report. Contract C/1.369/HQ/SS/04 to Eurocontrol (2004), `http://www.dcs.gla.ac.uk/~johnson/Eurocontrol/Ueberlingen/Ueberlingen_Final_Report.PDF`
5. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. 68(2) (2011)
6. Kochenderfer, M.J., Chryssanthacopoulos, J.P.: Robust airborne collision avoidance through dynamic programming. Project Report ATC-371, Massachusetts Institute of Technology, Lincoln Laboratory (2011)

7. Kuchar, J., Drumm, A.C.: The traffic alert and collision avoidance system. Lincoln Laboratory Journal 16(2), 277 (2007)
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
9. Rennen, G., van Dam, E.R., den Hertog, D.: Enhancement of sandwich algorithms for approximating higher-dimensional convex Pareto sets. INFORMS Journal on Computing 23(4), 493–517 (2011)
10. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Stateflow/Simulink verification. Formal Methods in System Design 43(2), 338–367 (2013)