Automatic Certification of the Active Corner Method for Collision Avoidance

Nishant Kheterpal[1], J. Tanner Slagel[2], Elanor Tang[3], Serra Dane[1], and Jean-Baptiste Jeannin[1]

[1] University of Michigan, Ann Arbor, Michigan, USA
{nskh, sdane, jeannin}@umich.edu
[2] NASA Langley Research Center, Hampton, Virginia, USA
joseph.t.slagel@nasa.gov
[3] Carnegie Mellon University, Pittsburgh, Pennsylvania, USA
elanor@cmu.edu

Abstract. Formal methods are particularly useful to guarantee safety before deployment in safety-critical tasks. Ensuring absence of collision for vehicles such as airplanes, cars, or robots is one common application of formal verification. However, many methods for verifying collision avoidance model a vehicle as a moving point mass, though real vehicles have non-zero area. Our prior work proposed a novel algorithm (the *active corner method*) for verifying collision avoidance for polygonal objects moving in the plane, along with a Python implementation. That implementation was not verified or certified, relying instead on a pen-and-paper proof of correctness. In this work, we describe an extension of the active corner method to fully automatically generate machine-checkable proof certificates of correctness for specific instances within the Prototype Verification System (PVS). This work briefly discusses the original active corner method proof approach, presents a novel algebraic approach that generalizes to any convex polygon, details the certification process with examples in PVS, and provides a case study using the differential dynamic logic integration within PVS. Our implementation, certificate examples, and proof scripts are available on GitHub.

1 Introduction

In safety-critical hybrid systems such as medical devices, power systems, automated vehicles, and airplanes, formal verification helps assure correct, safe operation and prevents harm or loss of life. This paper concerns collision avoidance verification, a key task in hybrid systems such as navigation for ground and aeronautical autonomous robotics. In particular, we consider verification for collision avoidance when the object in question has non-zero area. Many approaches for collision avoidance or hybrid system safety reason about point objects, which is unrealistic for robotics applications where hardware systems (e.g., drones, wheeled robots, and grasping arms) have volume and mass [28,4,22].

Our prior work [25] introduced the *active corner method* for verifying collision avoidance for polygons moving along known planar trajectories. That work

also provided a geometric paper proof of correctness and completeness, and an open-source implementation in Python. The active corner method produces a quantifier-free geometric representation of the *safe region*: the set of locations where a point obstacle will not collide with a (convex) polygonal object translating along a planar trajectory. However, there is currently a gap between the paper proof and the implementation; the Python code may contain bugs or incorrectly implement the method. Thus, whether errors arise due to existing bugs in libraries, programming errors, or mistakes in the specification, we need further guarantees for deployment in safety-critical settings.

Formal methods can bridge the *trust gap* between the Python implementation and the pen-and-paper proof. Approaches include full-scale reimplementation of algorithms in formally-verified languages or platforms, such as Rocq, the Prototype Verification System (PVS) [38], Isabelle [36], and others; or formally certifying the correctness of a tool or its outputs. This work presents a **certification** of the active corner method for collision avoidance in PVS, which couples a formal specification language with an interactive theorem prover [38,39].

We automatically generate machine-checkable PVS proof certificates demonstrating correctness for instances of the active corner method. Correspondingly, we have extended the open-source Python implementation of the active corner method to return not only a computationally efficient way to test for collision avoidance, but also a machine-checkable proof certificate of correctness in PVS. These certificates formally demonstrate the correctness of the Python output and bridge the trust gap between the original pen-and-paper proof and the Python code. Note that our certification code automatically generates both the theorem and a corresponding proof script, ensuring fully automatic certification.

The original proof in [25] leveraged significant geometric intuition, which is easily understood but not well suited for formalization in a theorem prover; this work presents a more algebraic approach that is more easily expressed using a proof assistant. We present a novel proof method and proofs of correctness in PVS for rectangles moving on continuous, piecewise, differentiable trajectories $y = f(x)$. An informal definition of correctness is as follows: when the active corner method states a point obstacle will be safely avoided, it must indeed be avoided using a more intuitive definition of collision avoidance. Our work can fully automatically generate PVS proof commands showing the correctness of outputs over regions of the domain in which the trajectory has a bounded derivative, as in the original proof of correctness for the active corner method [25]. We include a case study that leverages our certificates to show safety for strategic planning and trajectory optimization in air traffic control scenarios.

In brief, the contributions of this work are as follows:

- A novel algebraic proof approach demonstrating correctness for the active corner method.
- A fully automatic certification in PVS implementing that proof approach, currently limited to rectangular polygons.
- An extension of the open-source Python implementation of the active corner method to automatically generate PVS-checkable proof certificates.

– Several examples and a case study implemented in Python demonstrating the certificates.

2 Overview

Consider a polygonal object, representing a bounding volume v moving along some planar trajectory $\mathcal{T}$. The bounding volume v at a point (x, y) in the plane can be denoted $v(x, y)$. Given a single point obstacle in the plane at fixed location (x_O, y_O), a natural question is whether the object will collide with that obstacle as it moves along its trajectory. An intuitive approach is to simply check all the points along trajectory $\mathcal{T}$, in this work represented as a continuous, differentiable, potentially piecewise function $f(x)$ in the plane.

Mathematically, that intuitive approach to checking collision avoidance along a trajectory is a *quantified* one: *for all* points $(x_\mathcal{T}, y_\mathcal{T}) \in \mathbb{R}^2$ along a trajectory $\mathcal{T}$, ensure that an obstacle at coordinate (x_O, y_O) is not inside of a volume v centered at $(x_\mathcal{T}, y_\mathcal{T}) \in \mathbb{R}^2$. That is,

$$\forall (x_\mathcal{T}, y_\mathcal{T}) \in \mathcal{T}, (x_O, y_O) \notin v(x_\mathcal{T}, y_\mathcal{T}). \tag{1}$$

However, this quantified representation has one issue: the $\forall$ quantifier cannot be easily modeled in settings where computational efficiency or complete correctness matters. Simulation-based approaches may rely on imperfect discretization, and the Cylindrical Algebraic Decomposition (CAD) algorithm for eliminating quantifiers is doubly exponential in the number of total variables (not just the quantified variables) [19][18]. Correspondingly, the runtime of quantifier elimination makes CAD intractable even for simple symbolic trajectories that involve only a few parameters. For checking collision avoidance either on-the-fly (at run time) or at design time, when, say designing an air traffic control system, we need quantifier-free methods of checking collision avoidance.

This work focuses on the certification in PVS of a different approach to finding a collision avoidance statement, one free of quantifiers and equivalent to Equation (1). Our prior work introduced a fully symbolic method of generating a quantifier-free equivalent to a quantified statement of collision avoidance [25]. The quantifier-free formulation of the "safe region" in that work represented the set of obstacle locations where, given a polygon's path, a collision will not occur.

Consider a running example of the following polygon translating with its center following some trajectory function, displayed in Figure 1:

– An axis-aligned rectangle with width 4 and height 2, so its four corners are at offsets $(\pm 2, \pm 1)$ from the center. In Figure 1, this is the gray rectangle.
– A parabolic trajectory $y = x^2$. In Figure 1, this is the dashed purple line.

The key idea of the active corner method is that, to construct verifiably correct "safe regions" (or reachable sets) for polygonal objects moving in the plane, we must consider the shape of the object **and** the paths of its corners [25]. At any point along the trajectory, two vertices of a convex polygon represent

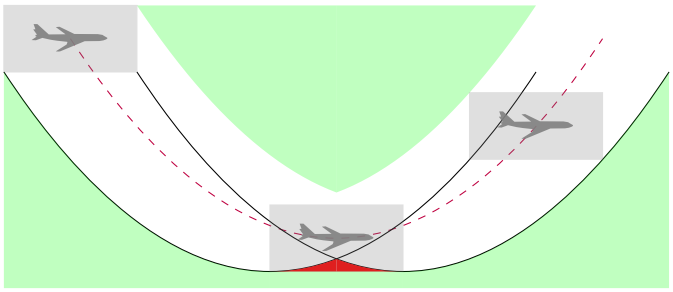


Fig. 1: A rectangular well-clear volume, representing an airplane, moving along a planar trajectory. At the *transition point* at the parabola's vertex, the "notch" is visible and shaded in red; part of the polygon lies outside of the corner-trajectories at this point.

the outermost extent of the polygon as it moves along the trajectory. These vertices are called *active corners*. For the rectangle under consideration, the bottom-left and top-right corners are *active* for $x \leq 0$, when the trajectory moves downward. For $x \geq 0$, the top-left and bottom-right corners are active. However, only following the paths of active corners could miss the "notch," an unsound underestimate that occurs when the set of active corners changes; that notch is depicted in Figure 1 in red. When the trajectory changes direction, the active corners may change as well, as in Figure 1 at the vertex of the parabola. At those instances, called *transition points*, the boundaries of the safe region follow a side of the polygon; for Figure 1, the boundary follows the bottom edge of the rectangle, thus accounting for the notch. Formally, transition points occur at any point where the tangent line to the trajectory function $f(x)$ is parallel to the slope of an edge of the polygon. In our running example, there is a single transition point at $x = 0$, the vertex of the parabolic trajectory.

In the active corner method, objects are assumed to move without rotating. Because of that, one can shift a (known) trajectory for the object center to get equations of motion for its corners. For the example of a rectangle moving along a parabola given above, the equation of motion for its top-left corner would be $y = (x + 2)^2 + 1$, as the function is shifted left by 2 and up by 1.

The active corner method can be used to evaluate whether a polygon translating along a continuous, differentiable, (potentially) piecewise trajectory $f(x)$ with its center at $(x, f(x))$ will intersect a point obstacle at (x_O, y_O). If that intersection occurs, the obstacle is deemed unsafe, or equivalently, to lie outside the "safe region" described earlier. To test if an obstacle at (x_O, y_O) is unsafe, the active corner method first checks if the obstacle lies in between the trajectories of the active corners of the polygon, which are opposite vertices for symmetric polygons. That is, an unsafe obstacle must be below the path of one active corner and above the path of its opposing corner. Any obstacle either above or below both corner-trajectories would be avoided, unless it lies in the notch (the solution for this is described below). Since the trajectory of a particular active

corner at offset $(\Delta x, \Delta y)$ is $y = f(x - \Delta x) + \Delta y$, one can substitute (x_O, y_O) for (x, y) and check whether y_O is greater than or less than $f(x_O - \Delta x) + \Delta y$ to see which "side" of the corner-trajectory on which the obstacle lies. Equivalently, one can check the sign of $y_O - f(x_O + \Delta x) - \Delta y$. To check this for both active corners and their trajectories, the active corner method checks that the two signs above are **different**; it does not matter which is negative and which is positive.

We can express this check as a product: Equation (2) is a Boolean test for an obstacle at (x_O, y_O) being *unsafe*. Note that $\Delta x_i, \Delta x_j, \Delta y_i, \Delta y_j$ are the offsets from the polygon center to the two active corners v_i and v_j, and the check is ≤ 0 because obstacles that lie exactly on a corner-trajectory are indeed unsafe.

$$(y_O - f(x_O + \Delta x_i) - \Delta y_i)(y_O - f(x_O - \Delta x_j) + \Delta y_j) \leq 0 \tag{2}$$

In our running example, the instantiation of Equation (2) yields the following test, which checks if an obstacle (x_O, y_O) lies between **either** pair of active corners (top-left with bottom-right, and bottom-left with top-right):

$$\begin{aligned} &\big((y_O - (x_O + 2)^2 - 1) \cdot (y_O - (x_O - 2)^2 + 1) \leq 0\big) \vee \\ &\big((y_O - (x_O - 2)^2 - 1) \cdot (y_O - (x_O + 2)^2 + 1) \leq 0\big) \end{aligned} \tag{3}$$

Using Equation (2) is not sufficient; at points along the trajectory where the active corners change, one must account for the *notch* (like the bottom of the parabola in Figure 1). At those discrete instances, called *transition points*, the active corner method additionally checks if an obstacle is unsafe by seeing if it lies within the polygon at that *transition point*. For this task, several point-in-polygon methods suffice; our proof technique represents an n-sided polygon as the intersection of n inequalities, each of which represents an edge. By checking the disjunction of multiple instances of Equation (2) for every pair of active corners and checking for point-in-polygon at every transition point, the active corner method constructs a test for whether an obstacle at (x_O, y_O) is unsafe.

In our running example, we extend Equation (3) with point-in-polygon checks for an instance of the rectangle centered at $x = 0$, our sole transition point. Note that this representation, the output of the active corner method, includes no trajectory variable x and can be used directly to check whether collision will occur with any point obstacle at (x_O, y_O). Only simple algebra is required.

$$\begin{aligned} &\big((y_O - (x_O + 2)^2 - 1) \cdot (y_O - (x_O - 2)^2 + 1) \leq 0\big) \vee \\ &\big((y_O - (x_O - 2)^2 - 1) \cdot (y_O - (x_O + 2)^2 + 1) \leq 0\big) \vee \\ &\big((x_O \geq -2) \ \wedge (x_O \leq 2) \wedge (y_O \geq -1) \ \wedge (y_O \leq 1)\big) \end{aligned} \tag{4}$$

To contrast, the quantified test for collision (the opposite of safety or collision avoidance, hence the use of $\exists$ instead of $\forall$) checks for the existence of some point $(x, y = x^2)$ along the trajectory for which a point obstacle (x_O, y_O) lies inside the boundaries of the rectangle, here represented by the conjunction of inequalities.

$$\begin{aligned} \exists x : \big((x_O - x \geq -2) \wedge (x_O - x \leq 2) \wedge \\ (y_O - x^2 \geq -1) \wedge (y_O - x^2 \leq 1)\big) \end{aligned} \tag{5}$$

We extend our prior work [25] to automatically certify in PVS the equivalence of Equations (4) and (5) for specific cases (described in Section 4.4).

3 A Mechanized Proof in PVS

3.1 Objective

Our soundness goal is to show that any point that the active corner method claims to be safe is indeed safe under the quantified definition in Equation (1).

To achieve this goal, we instead prove the *contrapositive*: that all points $(x_{\text{int}}, y_{\text{int}})$ on the interior of a polygon centered at $(x_{\mathcal{T}}, y_{\mathcal{T}})$ for $y_{\mathcal{T}} = f(x_{\mathcal{T}})$ will evaluate to *unsafe* when substituted in the quantifier-free equivalent test for safety (Equation (4), constructed by shifting the trajectory by active-corner coordinates $(\Delta x_i, \Delta y_i)$ and $(\Delta x_j, \Delta y_j)$).

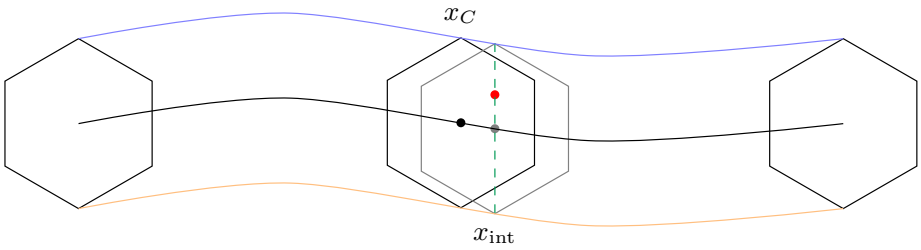


Fig. 2: Shifted polygon illustration. The red point is in the interior of the center black polygon but also lies along the dashed green centerline of the grey polygon.

3.2 Previous Geometric Approach

The original proof of correctness in [25] reasons about segments of a trajectory function in which the active corners do not change; that is, segments where the derivative of the function stays bounded within a certain interval. One key term used to bound the derivative is the slope of a side of the polygon. That proof shows soundness by reasoning about interior points of a polygon and showing that they lie along the center line of a different, shifted polygon (Figure 2). By bounding two quantities: 1) the coordinates of an interior point, and 2) the center coordinates of that shifted polygon, we can show that the interior point lies on the segment between active corners of the shifted polygon, and so Equation (2) evaluates to True and any interior point will test *unsafe*, as desired.

3.3 Algebraic Approach

We provide a novel proof approach that reasons algebraically rather than geometrically and is more straightforward to prove in PVS, using only analytic and

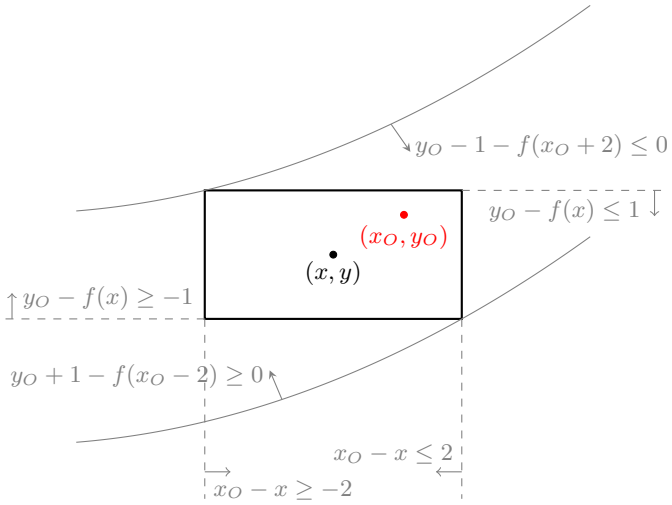


Fig. 3: Rectangle moving along trajectory $f(x)$. An obstacle (x_O, y_O) is labeled, as are inequalities defining the sides of the rectangle and the active-corner-trajectories.

algebraic manipulations. For example, it is a challenge to represent the slope of a side of an arbitrary polygon in a proof assistant. We encode the polygon representing the object moving along trajectory $f(x)$ as a conjunction of linear inequalities corresponding to each edge of the polygon. Our proof approach leverages the active corners by combining the inequalities corresponding to the edges adjacent to each corner, in order to show unsafeness according to Equation (2). Figure 3 displays inequalities defining both a rectangle and its active-corner trajectories, for some arbitrary trajectory $f(x)$. For the prior example of a rectangle moving along parabola x^2, all instances of $f(x)$ would be instantiated with x^2.

We automatically generate inequalities defining the edges of the polygon (assuming the center is at (0,0)) by:

- using the point-slope equation for each pair of adjacent points to get the edge;
- substituting (0,0) to determine the correct direction of the inequality;
- in order to get inequalities describing if (x_O, y_O) is contained in the general polygon centered at $(x, y := f(x))$, substituting $(x_O - x, y_O - f(x))$ for (x, y).

This last step combines two transformations: we use (x_O, y_O) instead of (x, y) since we are interested in the location of the obstacle, and we subtract $(x, f(x))$ from (x_O, y_O) so inequalities are relative to the polygon center along trajectory $(x, f(x))$ rather than $(0, 0)$.

In our running example and Figure 3, the inequalities defining the edges of a rectangle are

$$y_O - x^2 \geq -1, \quad y_O - x^2 \leq 1, \quad x_O - x \geq -2, \quad x_O - x \leq 2 \tag{6}$$

This method has allowed us to automatically generate our proof goals: PVS lemmas of soundness given a polygon, trajectory, and list of active-corner pairs. That process is described later in Section 4, and this running example is provided in our artifact as `example_0.py`. In our certification, unlike the original fully symbolic active corner method, we require that the vertices and trajectories be instantiated with real numbers, since the computation is significantly more difficult with symbolic vertices. This limits our certification compared to the fully symbolic nature of the original active corner method, but still permits certification for real-world examples.

3.4 Proof Structure

The proof of the active corner method structures its arguments segment-by-segment along a trajectory, reasoning over domains in which the active corners (defining the outermost extent of the polygon) do not change.

For example, for our running example of a rectangle moving on function $y = x^2$ over the reals, the active corners change (or there is a "transition point") at $x = 0$. Thus there are two "sections" to our proof, generated automatically: one over the domain $(-\infty, 0]$, and another over the domain $[0, \infty)$.

Our PVS certification proceeds similarly. We extend the prior Python implementation to compute details required to structure our proofs, such as the domain under consideration and any bounds of the derivative in that domain. This was one reason to choose certification of outputs rather than a full formalization of the method in PVS; much of the work done to compute transition points, derivative bounds, and more is straightforward using Python and SymPy [27]. In this way, Python and SymPy are used almost as untrusted oracles, with computation checked in PVS. As the original active corner method implementation was written using SymPy, any limitations of SymPy would affect generation of the quantifier-free result. Therefore, any certificate generated remains valid for showing soundness, as SymPy errors would at worst prevent generating any output and, since certificates are generated using SymPy output, a certificate.

At the end of our proof certificate, we provide a "unifying lemma," uniting the piece-by-piece cases into a single theorem proving soundness and safety throughout the domain under consideration.

3.5 Segment Proofs

Consider a segment of the trajectory $f(x)$ over which the active corners and piecewise case do not change. Over this segment, our goal is to show that the test in Equation (2) holds for every obstacle point (x_O, y_O) that is inside the polygon when it is centered at $(x, f(x))$. Because the active corners do not change, the slope of the function $f(x)$ is limited; active corners only change when the tangent line to the trajectory is parallel to an edge of the polygon.

For example, for an axis-aligned rectangle, the active corners change when the slope of the trajectory equals zero, which is the slope of the bottom and top edges of the rectangle. Thus, we consider domains over which $f'(x) \geq 0$ and

$f'(x) \leq 0$. For an equilateral diamond, the active corners change when the slope of the trajectory equals -1 or 1. Thus, we would consider domains over which $f'(x) \geq 1$, $f'(x) \in [-1, 1]$, and $f'(x) \leq -1$.

The polygon interior is described by a small set of edge inequalities, as seen in Figure 3. For a certain x, the horizontal offset of any *unsafe* interior point is bounded (the obstacle cannot be arbitrarily far left or right of the center).

The inequalities defining any polygon involve $y_O - f(x)$, as seen in Figure 3, while the test in Equation (2) includes no x, only terms like y_O, x_O, and $f(x_O)$. Our proof technique rests on using the *mean value theorem*: over a region where the derivative $f'(x)$ is bounded, shifting the evaluation point from x to x_O (or to $x_O \pm n$ for some small n) changes f by an amount we can bound using exactly the bound we have on the derivative $f'(x)$ over domain D. For example, if $|f'(x)| \leq M$ for all x in a domain, then $|f(b) - f(a)| \leq M|b - a|$ for any a, b in that domain. As Equation (2) is a product of two inequalities, our proofs use the mean value theorem twice: once to show that one of the inequalities in Equation (2) is nonpositive and again to show that the other is nonnegative.

3.6 Unifying Lemma

The per-segment proofs described above hold over domains where the active corners do not change, and thus trajectory derivative $f'(x)$ is bounded. Our certificate links together the per-segment proofs into a "unifying lemma" that combines these per-segment proofs into a single theorem showing soundness. For our running example, this lemma would link together the proof cases for $x \leq 0$ (one set of active corners for $f'(x) \leq 0$) and $x \geq 0$ (the other set of corners corresponding to $f'(x) \geq 0$). Certificates can be automatically generated for trajectories with an arbitrary number of segment proofs, which come from both piecewise cases and transition points.

4 Automation in PVS

Our certification begins with the open-source Python implementation from the original active corner method paper [25], and extends the Python computation in SymPy to automatically generate proof scripts for PVS. Our certification requires PVS 8, which includes updated libraries and tactics used in our proofs. Given a trajectory, domain, and polygon, a Python script generates proof scripts which can be checked, as displayed in Figure 4.

To automatically construct PVS lemmas representing soundness, our certification constructs inequalities that represent each edge of the polygon. The process implements the automatic inequality construction described in Section 3.3.

In order to generate the goal, we run the original active corner method to generate the quantifier-free formulation, which is the disjunction of several products, each of two inequalities. In our proof goals, we restate Equation (2) as a conjunction of inequalities $\geq$ or ≤ 0, rather than multiplying two inequalities, because the conjunction is an easier proof goal to discharge in PVS. Note that

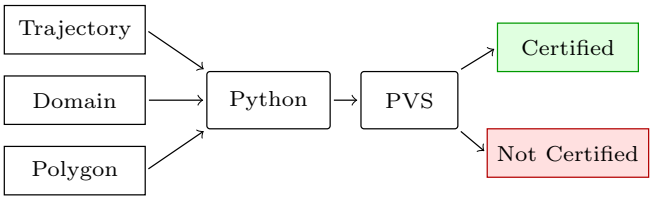


Fig. 4: Proof automation structure.

both possibilities of positive/negative sign are stated in this formulation, so the conjunction of inequalities is equivalent to the product representation.

Our resulting proof goal is to show that Equation (5) implies Equation (4), stated without multiplication:

$$\begin{aligned}
&\exists x : \big((x_O - x \geq -2) \wedge (x_O - x \leq 2) \wedge (y_O - x^2 \geq -1) \wedge (y_O - x^2 \leq 1)\big) \\
&\qquad\qquad \Longrightarrow \\
&\qquad \big((y_O - (x_O - 2)^2 - 1) \leq 0 \wedge (y_O - (x_O + 2)^2 + 1) \geq 0\big) \vee \\
&\qquad \big((y_O - (x_O - 2)^2 - 1) \geq 0 \wedge (y_O - (x_O + 2)^2 + 1) \leq 0\big) \vee \\
&\qquad \big((y_O - (x_O + 2)^2 - 1) \leq 0 \wedge (y_O - (x_O - 2)^2 + 1) \geq 0\big) \vee \\
&\qquad \big((y_O - (x_O + 2)^2 - 1) \geq 0 \wedge (y_O - (x_O - 2)^2 + 1) \leq 0\big) \vee \\
&\qquad\quad \big((x_O \geq -2) \wedge (x_O \leq 2) \wedge (y_O \geq -1) \wedge (y_O \leq 1)\big)
\end{aligned} \tag{7}$$

4.1 Lightweight Proof Automation with Prooflite

As a fully automatic method of certifiably checking collision avoidance, we aim for our certificate generator to be accessible to non-formal methods practitioners such as engineers. This influenced our decision to use SymPy and Python. Directly integrating the proof certificate generation with the existing active corner implementation makes it easy for any Python user to additionally generate and check certificates. Furthermore, because of the segment-by-segment structure of our proof (Section 3.4), any certificate generation would require some amount of computation in Python (or another platform supporting symbolic mathematics), regardless of how the proof steps were specified in PVS. For example, identifying piecewise cases, transition points, and bounds on the trajectory derivative all rely on SymPy symbolic differentiation and numeric evaluation, which would be unnecessarily difficult if done entirely in PVS. This factor was another reason to pursue a certification of the active corner method, rather than a full formalization in PVS.

As such, we chose to build proof certificates in Prooflite, which is a lightweight proof automation module within PVS [31]. Prooflite scripts allow users to specify sequences of tactics and include them alongside theorems or lemmas. Using Prooflite, our soundness proof certificates consisted of a single PVS file both stating properties of soundness and providing their proofs. Better yet, PVS proofs using Prooflite scripts can be proven straightforwardly from the command line:

using `proveit` will typecheck and prove all lemmas included in a PVS file, reporting which were unproved or not attempted (due to proofs not being specified).

When conducting this work, our proof workflow consisted of iteratively proving more complex theorems of soundness, by modifying domains, polygon dimensions, trajectory functions, and the number of proof cases. Once we had a fully-proved example soundness case, we attempted to streamline the proof as much as possible at the tactic level; for example, removing extraneous proof commands, hard-coding as few proof terms in the sequent as possible, etc. We then generated a Prooflite script for that theorem in PVS automatically, proceeding to parametrize it in Python to generate, say, a similar case but for a different domain or derivative bound. We constructed scripts by computing values for variables in the proof script and filled in a template using Python f-strings. The example in Figure 5 shows part of a Prooflite script *template* for one soundness lemma; note that terms in curly braces like {`max_right`} are replaced with terms like `xo - 2` in an actual proof script when instantiated using a Python f-string.

```
%|-  (SPREAD (CASE "{case_label1}")
%|-   ((THEN (FLATTEN) (LEMMA "{deriv_lemma}")
%|-     (SPREAD (INST -1 "f" "{domain_start}" "{domain_end}"
%|-                      "{deriv_bound}" "{max_right}" "x")
%|-      ((SPREAD (SPLIT -1)
%|-        ((THEN (ASSERT) (LEMMA "{deriv_lemma}")
%|-          (SPREAD (INST -1 "f" "{domain_start}" "{domain_end}"
%|-                           "{deriv_bound}" "x" "{min_left}")
%|-           ((ASSERT) (THEN (EXPAND "ci") (PROPAX))
%|-            (THEN (ASSERT) (EXPAND "ci") (PROPAX)))))
%|-         (ASSERT) (PROPAX) (PROPAX) (PROPAX)))
%|-       (THEN (EXPAND "ci") (ASSERT)) (THEN (EXPAND "ci") (ASSERT)))))
```

Fig. 5: Prooflite script template example for a closed interval (`"ci"`) subdomain.

4.2 Segment Proof Automation

To prove the soundness lemma for each segment of the proof certificate, in which active corners do not change (and thus the trajectory derivative can be bounded), we use two known terms: the edge-inequalities that define the polygon and the derivative bounds over that segment/domain. Next, we introduce the mean value theorem lemma corresponding to that segment. With these pieces in place, we can relate $f(x)$ and $f(x_O)$ to prove the consequent.

Using the `assert` command allows us to automatically simplify expressions and propositions for particular cases [43]. This command simplifies using decision procedures and applies a number of automatic rewrites; it can prove theorems that involve only linear inequalities and propositional logic, such as some of our proof cases. In our proofs, we use `assert` after stating and instantiating the

relevant mean value theorem lemma, letting us show part of or a full term in the consequent, in combination with terms like the derivative bounds (since the active corners do not change) and the edge-inequalities that define the polygon.

```
1: ge_ro_case_1: LEMMA
2:     FORALL(xo,yo:real, g:[real -> real]):
3:         LET f = LAMBDA(x:real):
4:         COND
5:             x < 0 -> g(0),
6:             x >= 0 -> g(x)
7:         ENDCOND
8:     IN
9:     derivable?[(right_open(0))](f) AND
10:     (FORALL(x:(right_open(0))):
11:         deriv[(right_open(0))](f)(x) >= 0)
12:     AND (EXISTS (x : real) :
13:     ((-x + xo + 2 >= 0) AND
14:      (-x + xo - 2 <= 0) AND
15:      (yo - f(x) + 1 >= 0) AND
16:      (yo - f(x) - 1 <= 0)) AND
17:     x >= 0 AND
18:     xo + 2 >= x AND xo + -2 <= x)
19:     IMPLIES
20:     ((yo - f(xo - 2) - 1 >= 0) AND
21:      (yo - f(xo + 2) + 1 <= 0)) OR
22:     ((yo - f(xo - 2) + 1 >= 0) AND
23:      (yo - f(xo + 2) - 1 <= 0)) OR
24:     ((yo - f(xo + 2) - 1 >= 0) AND
25:      (yo - f(xo - 2) + 1 <= 0)) OR
26:     ((yo - f(xo + 2) + 1 >= 0) AND
27:      (yo - f(xo - 2) - 1 <= 0)) OR
28:     (xo + 2 >= 0) AND
29:     (xo - 2 <= 0) AND
30:     (yo - f(0) + 1 >= 0) AND
31:     (yo - f(0) - 1 <= 0)
```

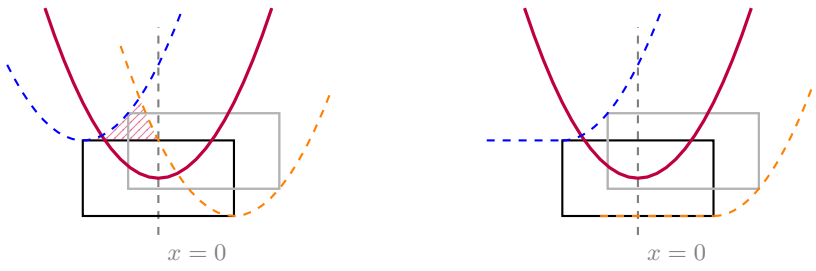
Fig. 6: Example single-segment lemma for a rectangle of width 4 and height 2. In the antecedent, lines 3-7 define the "clipped" trajectory, 9-11 bound the derivative, 12 states some unsafe x exists, 13-16 define the polygon edges, 17-18 restrict the domain. In the consequent, lines 20-27 state the quantifier-free tests from Equation 2, and 28-31 test for the notch.

Consider, for example, the ascending segment of our rectangle on parabola running example. In this case, the derivative bound is $f'(x) \geq 0$ for domain $x \geq 0$. Correspondingly, the proof script for this segment calls a mean-value theorem lemma corresponding to 1) a lower bound on the derivative over 2) a domain unbounded to the right. The proof script then instantiates that lemma

with our domain single-side bound 0, derivative bound 0, trajectory variable x, and the desired term from the proof goal $x_O - 2$. An example of the lemma showing correctness for this segment can be seen in Figure 6.

Our soundness lemmas do split on one or two cases, depending on the domain. This is because of a clipping rule in the original active corner method paper [25, Section III.D], which holds functions constant outside of the domain of a given piecewise function. The rule was originally intended to guard against arbitrary behavior of piecewise trajectories outside their defined subdomain. However, while proving soundness for certain cases, we discovered this "clipping" rule was necessary even for non-piecewise functions. Without clipping, the proof assistant generated cases corresponding to counterexamples in which trajectories curved upwards and valid obstacles were on the "same side" of both active corner trajectories. Figure 7 shows the counterexamples and how clipping the trajectory function to remain constant eliminates these counterexamples and enables certification. Our discovery of such counterexamples, as well as the more general need for clipping, further highlights the value of formal methods and certification.

One benefit of this approach is that relatively few proof script templates are required when automating proofs. We can simply instantiate the template with the correct arguments (the particular domain and derivative bounds, domain type, polygon dimensions, and so on) to construct a valid proof script.



(a) Without clipping (b) With clipping

Fig. 7: Proof counterexample necessitating clipping. a) The shaded region is clearly unsafe but is on the same side (below) both the orange and blue corner-trajectories. b) After clipping, the counterexample region no longer appears.

4.3 Unifying Lemma Automation

In order to prove the unifying lemma, the proof script divides the domain into multiple cases and invokes each of the per-segment lemmas. To show the bounds on the derivative hold for the trajectory in question, we use the `deriv` command in PVS, which differentiates the trajectory function; the limitations of that command are discussed in Section 4.4. The proof proceeds by stating that

the domains of the lemma and the case are equivalent and expanding the definitions of the "clipped" functions described above for each per-segment lemma. Various side conditions are generated but are straightforward to prove.

```
Processing certificate.pvs.
Writing output to file certificate.summary
Logging PVS information in certificate.log
Including proof traces in output file

Proof summary for theory
    active_corner_certificate
Theory totals:
  40 formulas, 40 attempted, 40 succeeded
  (0.01 s)
```

Fig. 8: `proveit` output from an example certificate.

In practice, given: 1) a polygon, 2) a trajectory, and 3) a domain containing n intervals over which the active corners do not change, a total of n per-segment lemmas and one unifying lemma are generated. Despite this, Figure 8 shows many more than a handful of lemmas are attempted and proved, because:

1. The Prooflite certificate file is self-contained, so it includes several theorems defining mathematical properties we need (derived from the mean value theorem for a variety of cases). These lemmas have been fully proved and will be added to NASALib. After that, their inclusion in the certificate will no longer be necessary, so the number of lemmas per example will decrease.
2. PVS automatically generates type correctness conditions, called TCCs, while type-checking. Many of these proofs are easy or even trivial to discharge. In our proof automation, we used approaches including calling lemmas to show differentiability or that a domain is not a singleton, running `assert` when trivial, or running PVS's automatic `grind` command which repeatedly applies rewrites, simplifications, and more.

4.4 Capabilities and Limitations

While our lemma generation code generalizes to convex polygons and various trajectories, the proof automation currently supports only rectangles—a common choice for bounding volumes in collision detection. Extending the proof automation to handle arbitrary convex polygons is future work. Our certification also inherits limitations from the original active corner method: namely, that trajectories must have finitely many transition points over the domain. The `deriv` command, used to verify derivative bounds, may not always succeed automatically; our certificates include a helper lemma that assumes derivative bounds hold, with a separate proof that the trajectory satisfies those bounds.

5 Examples

The GitHub repository[4] for this work contains our Python implementation to automatically generate correctness certificates for the active corner methods, a selection of example certificates, and our case study. All certificate examples begin with Python files describing the three inputs to our certification:

1. A (potentially-)piecewise trajectory function $f(x)$.
2. A domain, either unbounded or bounded over real numbers.
3. A polygon; as described in Section 4.4 our fully automatic proofscript generation supports only rectangles but lemmas can be generated for other convex polygons.

Example	Certificate Generation (s)	Certificate Checking (s)	Number of Goals
1	0.84	10.30	34
2	0.74	7.91	31
3	0.73	30.02	42
4	0.77	33.28	42
5	0.88	27.38	48
6	0.85	41.02	48
7	0.87	27.72	48
8	0.83	65.09	62
9	0.85	64.59	62
10	0.78	45.78	55

Table 1: Proof certificate generation and checking times.

Table 1 displays how long each of the following examples took to run on a MacBook Pro with 32GB of RAM and an M1 Pro processor. Generating the certificates took effectively the same time regardless of scenario complexity, with all certificates taking between 0.7 and 0.9 seconds to generate in Python. On the other hand, scenario complexity significantly impacted the time to check each certificate in PVS, roughly following along with the number of proof goals.

(E1) Linear ascent on a closed interval. Let $f(x) = x$ on $[-10, 10]$. There is a single segment with $f'(x) = 1 \geq 0$. The certificate comprises one segment lemma; the unifying lemma leverages the one segment lemma.

(E2) Linear descent on the reals. Let $f(x) = -x$ on $\mathbb{R}$. There is a single (unbounded) segment with $f'(x) = -1 \leq 0$. No clipping (as displayed in Figure 7)

[4] https://github.com/nskh/nfm26-artifact

is required; the unifying lemma discharges straightforwardly from the segment lemma without needing to split into cases.

(E3, E4) Parabolas on the reals. For $f(x) = x^2 - 4x + 3$ and $f(x) = -x^2 - 10x - 25$, each trajectory has one transition point at the parabola's vertex, yielding two segments. For the first parabola, this vertex is at $x = 2$, yielding segments $(-\infty, 2]$ and $[2, \infty)$; the second has a vertex at $x = -5$ and segments $(-\infty, -5]$ and $[-5, \infty)$. The certificates for each example lay out two segment lemmas plus a unifying lemma over $\mathbb{R}$ that bridges the vertex transition. The clipping rule illustrated in Figure 7 is used here and in the following examples.

(E5, E6, E7) Parabola–linear hybrids (single transition at the vertex). We include two piecewise, continuous, differentiable trajectories, representing shifted versions of each other:

$$f(x) = \begin{cases} x^2, & x \le 4, \\ 8x - 16, & x > 4, \end{cases}$$

$$f(x) = \begin{cases} x^2 + 2x + 1, & x \le 3, \\ 8x - 8, & x > 3. \end{cases}$$

Each has one transition point (at $x = 0$ and $x = -1$, respectively), so the proof consists of two segment lemmas (left/right of the vertex) and a unifying lemma stitches the cases together. **Note:** in PVS, the `deriv` tactic is more robust on expanded polynomials; expanding $(x + 1)^2$ as $x^2 + 2x + 1$ improves automation when discharging the derivative-bound obligations. Our examples include both the factored and expanded forms to illustrate this; the final unifying lemma fails to prove fully if $(x + 1)^2$ is used but succeeds for $x^2 + 2x + 1$.

(E8, E9) Parabola–linear–parabola–linear with interior notch. Two C^1 piecewise trajectories alternating between parabolas and linear segments:

$$f(x) = \begin{cases} x^2, & x \le -10, \\ -20x - 100, & -10 < x \le 0, \\ 2x^2 - 20x - 100, & 0 < x \le 10, \\ 20x - 300, & x > 10; \end{cases} \qquad f(x) = \begin{cases} x^2, & x \le -8, \\ -16x - 64, & -8 < x \le 0, \\ 2x^2 - 16x - 64, & 0 < x \le 8, \\ 16x - 192, & x > 8. \end{cases}$$

Each has four piecewise segments, but the middle parabola contains a vertex ($x = 5$ and $x = 4$, respectively) where $f' = 0$. This interior transition point splits that segment, yielding five segment lemmas from four piecewise pieces. The unifying lemma bridges both piecewise boundaries and the notch.

(E10) Linear–parabola–linear.

$$f(x) = \begin{cases} -6x - 72, & x \le -12, \\ \frac{1}{4}x^2 - 36, & -12 < x \le 12, \\ 6x - 72, & x > 12. \end{cases}$$

The parabola's vertex at $x = 0$ introduces a notch where $f'(0) = 0$, splitting the middle segment. As such, the proof requires four segment lemmas.

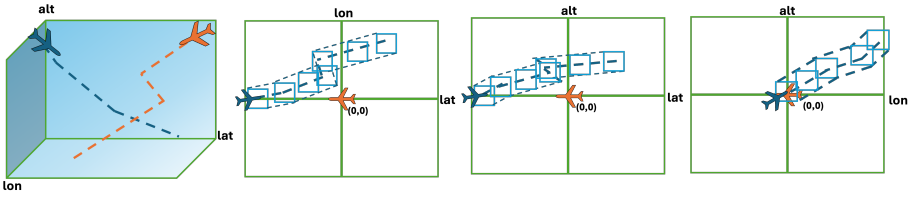


Fig. 9: This figure visualizes the computation of piecewise relative trajectories between two aircraft in four dimensions (latitude, longitude, altitude, and time). In order: a 3-D view of the two flight paths; horizontal separation in longitude-latitude; vertical separation in altitude–latitude; vertical separation in altitude–longitude.

6 Case Study: Strategic Planning for Air Traffic Control

This case study concerns airspace strategic planning, where air traffic control plans the trajectories of aircraft entering an airport's airspace with the intention to land. This setting is particularly relevant for trajectory optimization, where an optimizer outputs discrete paths defined by waypoints. However, after the optimization is performed, it is necessary to verify that the resulting aircraft trajectories remain collision-free. In this case study, we formally validate one set of optimization results against a predefined safety specification using the certificate generation implementation in this paper.

Simple Two-Aircraft Planning: We study two optimized aircraft trajectories in four dimensions (lat, lon, alt, and time). The trajectories were generated by a Mixed-Integer Linear Programming (MILP) optimizer, which takes as input the historical pre-Terminal Radar Approach Control (TRACON) entry points of the two aircraft and computes flight paths that satisfy required safety separations while minimizing fuel consumption. The trajectories of the two aircraft, output by the optimization, are a series of timed waypoints.

To enable verification, we first generate the *relative* trajectory of aircraft A with respect to aircraft B, thus treating B as a (stationary) point obstacle: concretely, we consider the trajectory of aircraft A in the referential of aircraft B. We consider a well-clear volume around aircraft A of side $1,000\,\text{ft}$ in latitude, longitude and height ($\pm 500\,\text{ft}$). Figure 9 displays how the relative trajectories are generated. We then construct a piecewise linear function to approximate motion between consecutive waypoints using linear interpolation, assuming pilots follow them exactly. In this first-order approximation, each segment of the trajectory is represented by a straight line connecting successive waypoints. To illustrate, consider a two-dimensional relative trajectory defined by waypoints $(x_0, y_0), \ldots, (x_n, y_n)$, where n is the number of waypoints. Since this is a relative trajectory, we do not need timestamps anymore to define collision avoidance. Also note that given the cubic nature of the well-clear volume in this example, absence of collision in any two dimensions (lat-lon, lat-alt or lon-alt) implies absence of collision in three dimensions.

To avoid floating-point discrepancies in certificate generation and proof, we round the piecewise-linear segment coefficients (m_i, b_i) of the relative trajectory to integer-valued representations in degrees; because the enforced well-clear margins correspond to large angular separations, any perturbation induced by rounding is insignificant relative to the safety boundaries. Then, using the generated piecewise function and the required safety cube, we apply the automatic certificate generator implementation and prove that the quantifier-free safe region is equivalent to the relative trajectory. As a result, we are able to validate our safety condition against the verified quantifier-free boolean condition for the unsafe region. This case study is provided in our artifact as example file `strategic.py`.

7 Related Work

PVS has been used to verify properties of hybrid systems in the past [44,1] along with tools like Isabelle [20]. Barendregt and Barendsen [11] lay out various approaches for trusting algorithms: an *autarkic* approach, where an algorithm is fully reimplemented in a theorem prover, or a *skeptical* approach, where a "trace" (laying out the steps taken) of computation or proof is verified. Our certificate structure, and many others, follows this *skeptical* approach. Our work in generating checkable proof certificates is reminiscent of the Proof Module in CVC [24,9]. CVC5 uses a domain-specific language to generate these proofs in a modular, lazily evaluated fashion [37,10]. Other satisfiability (SAT) and satisfiability modulo theories (SMT) solvers include certificate capabilities, such as Z3 [14,13], VeriT [42] and zChaff (which uses Rocq) [23].

The PVS system has a long history of being applied to the formal verification of safety–critical systems, particularly in the domain of air traffic management and unmanned aircraft systems (UAS). In the context of collision avoidance, PVS has been used to model and verify early aircraft separation systems [15], and more recently to develop and analyze advanced UAS safety concepts such as PolySafe [16]. PVS played a central role in the formal development of DAIDALUS [30], the reference implementation of detect–and–avoid (DAA) algorithms, included in Appendix G of RTCA/FAA DO-365 [41], standards that represent a step toward trustworthy deployment of autonomous collision-avoidance technologies. The body of work on PVS in the context of UAS detect–and–avoid, collision avoidance, and certification demonstrates its suitability for real-world, safety–critical aerospace applications [32]. PVS includes the NASALib library, which contains formalizations of mathematical concepts in geometry, calculus, real analysis, and quantifier elimination that are useful building blocks for the proofs we present [34].

Due to the safety-critical nature of the task, many methods for collision avoidance checking exist. Zonotope reachability methods iteratively propagate a set-based abstraction to compute an estimate of safety [22], and have been used to verify safety for automated road vehicles [3] and quadrotor aircraft [26]. Barrier certificates are a common tool for verifying safety invariants for dynamical

and hybrid systems [40], as has Hamilton-Jacobi reachability [8]. Safe controllers can be synthesized using control barrier functions [5].

Beyond collision avoidance, PVS has supported a range of mathematical foundations necessary for certification of safety–critical systems. This includes libraries for real analysis, hybrid systems reasoning, and decision procedures based on Sturm's and Tarski's theorems for univariate quantifier elimination [33]. PVS itself can export externally-checkable proof certificates [21] to Dedukti [7] and MetiTarski [2]. Certification-inspired methods have also been explored in the domain of numerical software, most notably through PRECiSA, a tool for statically certifying bounds on floating-point roundoff error using PVS proofs [45,29]. Similar finite-precision error bounds can be formalized using in Rocq and HOL4 [12]. Armand et al. presented proof witnesses for SMT solving in Rocq [6]. Proof-carrying code also relies on a *certifier* to check correctness of compilation and computation, with early work done by Necula et al. [35,17].

8 Conclusion

In this paper, we have reviewed the active corner method; discussed two proof approaches: the original geometric proof from [25] and a novel, more algebraic approach; formalized this new proof in PVS; presented our automatic proof certificate generator; and discussed several fully automatic examples and a case study requiring some manual proof effort. Future work includes extending our certificates to other polygons, supporting symbolic trajectory parameters, generalizing the active corner method to more reachability problems, and generating proof certificates outside PVS for other applications, such as solutions of differential equations.

Acknowledgments

We would like to thank Aaron Dutle, César Muñoz, and Mariano Moscato for their assistance in developing the proof techniques and PVS proofs presented in this paper. This work was funded by NASA Fellowship #80NSSC20K1465, National Science Foundation grant CCF-2422028, and the U.S. Federal Aviation Administration through the FAST Grant Program, Grant Number NG-FAS-0016, Funding Opportunity Number 23-FAA-FAST-001 under the supervision of Joshua Glottmann. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the FAA.

References

1. Ábrahám-Mumm, E., Hannemann, U., Steffen, M.: Verification of hybrid systems: Formalization and proof rules in PVS. In: Proceedings of the Seventh IEEE International Conference on Engineering of Complex Computer Systems. pp. 48–57. IEEE (2001)
2. Akbarpour, B., Paulson, L.C.: Metitarski: An automatic theorem prover for real-valued special functions. Journal of Automated Reasoning **44**(3), 175–205 (2010)
3. Althoff, M., Dolan, J.M.: Online verification of automated road vehicles using reachability analysis. IEEE Transactions on Robotics **30**(4), 903–918 (2014)
4. Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. Annual Review of Control, Robotics, and Autonomous Systems **4**, 369–395 (2021)
5. Ames, A.D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., Tabuada, P.: Control barrier functions: Theory and applications. In: 2019 18th European Control Conference (ECC). pp. 3420–3431. IEEE (2019)
6. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of SAT/SMT solvers to Coq through proof witnesses. In: International Conference on Certified Programs and Proofs. pp. 135–150. Springer (2011)
7. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory (2016)
8. Bansal, S., Chen, M., Herbert, S., Tomlin, C.J.: Hamilton-Jacobi reachability: A brief overview and recent advances. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC). pp. 2242–2253. IEEE (2017)
9. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: CVC5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022)
10. Barbosa, H., Reynolds, A., Kremer, G., Lachnitt, H., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Viswanathan, A., Viteri, S., Zohar, Y., Tinelli, C., Barrett, C.: Flexible proof production in an industrial-strength smt solver. In: 11th International Joint Conference on Automated Reasoning, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings. p. 15–35. Springer-Verlag, Berlin, Heidelberg (2022)
11. Barendregt, H., Barendsen, E.: Autarkic computations in formal proofs. Journal of Automated Reasoning **28**(3), 321–336 (2002)
12. Becker, H., Zyuzin, N., Monat, R., Darulova, E., Myreen, M.O., Fox, A.: A verified certificate checker for finite-precision error bounds in Coq and HOL4. In: Formal Methods in Computer Aided Design (FMCAD). IEEE (2018)
13. Böhme, S., Fox, A.C., Sewell, T., Weber, T.: Reconstruction of Z3's bit-vector proofs in HOL4 and Isabelle/HOL. In: International Conference on Certified Programs and Proofs. pp. 183–198. Springer (2011)
14. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: International Conference on Interactive Theorem Proving. pp. 179–194. Springer (2010)

15. Carreño, V., Muñoz, C.: Aircraft trajectory modeling and alerting algorithm verification. In: International Conference on Theorem Proving in Higher Order Logics. pp. 90–105. Springer (2000)
16. Colbert, B.K., Slagel, J.T., Crespo, L.G., Balachandran, S., Muñoz, C.: Polysafe: a formally verified algorithm for conflict detection on a polynomial airspace. IFAC-PapersOnLine **53**(2), 15615–15620 (2020)
17. Colby, C., Lee, P., Necula, G.C., Blau, F., Plesko, M., Cline, K.: A certifying compiler for Java. In: Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation. p. 95–107. PLDI '00, Association for Computing Machinery, New York, NY, USA (2000)
18. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. pp. 134–183. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
19. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. Journal of Symbolic Computation **5**(1), 29–35 (1988)
20. Foster, S., Huerta y Munive, J.J., Gleirscher, M., Struth, G.: Hybrid systems verification with Isabelle/HOL: Simpler syntax, better models, faster proofs. In: International Symposium on Formal Methods. pp. 367–386. Springer (2021)
21. Gilbert, F.: Proof certificates in PVS. In: International Conference on Interactive Theorem Proving. pp. 262–268. Springer (2017)
22. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: International Workshop on Hybrid Systems: Computation and Control. pp. 291–305. Springer (2005)
23. Jouannaud, J.P., Strub, P.Y., Zhang, L.: Certification of SAT solvers in Coq. In: Guangzhou Symposium on Satisfiability in Logic-Based Modeling (2010)
24. Katz, G., Barrett, C., Tinelli, C., Reynolds, A., Hadarean, L.: Lazy proofs for DPLL(T)-based SMT solvers. In: 2016 Formal Methods in Computer-Aided Design (FMCAD). pp. 93–100. IEEE (2016)
25. Kheterpal, N., Tang, E., Jeannin, J.B.: Automating geometric proofs of collision avoidance with active corners. In: Griggio, A., Rungta, N. (eds.) Proceedings of the 22nd Conference on Formal Methods in Computer-Aided Design – FMCAD 2022. vol. 3, pp. 359–368. TU Wien Academic Press (2022)
26. Kousik, S., Holmes, P., Vasudevan, R.: Safe, aggressive quadrotor flight via reachability-based trajectory design. In: Dynamic Systems and Control Conference. vol. 59162, p. V003T19A010. American Society of Mechanical Engineers (2019)
27. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, e103 (2017)
28. Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24-June 28, 2013 (2013)
29. Moscato, M., Titolo, L., Dutle, A., Munoz, C.A.: Automatic estimation of verified floating-point round-off errors via static analysis. In: International Conference on Computer Safety, Reliability, and Security. pp. 213–229. Springer (2017)
30. Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., Chamberlain, J.: Daidalus: detect and avoid alerting logic for unmanned systems. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC). pp. 5A1–1. IEEE (2015)

31. Muñoz, C.A.: Batch proving and proof scripting in pvs. Tech. rep. (2007)
32. Muñoz, C.A., Dutle, A., Narkawicz, A., Upchurch, J.: Unmanned aircraft systems in the national airspace system: a formal methods perspective. ACM SIGLOG News **3**(3), 67–76 (2016)
33. Narkawicz, A., Muñoz, C., Dutle, A.: Formally-verified decision procedures for univariate polynomial computation based on Sturm's and Tarski's theorems. Journal of Automated Reasoning **54**(4), 285–326 (2015)
34. NASA Formal Methods Team, LaRC and the PVS Community: NASALib: NASA PVS Library of Formal Developments. GitHub repository (2025), https://github.com/nasa/pvslib, version v7.1.1 (stable release) / current development pre-release
35. Necula, G.C., Lee, P.: The design and implementation of a certifying compiler. In: Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation. p. 333–344. PLDI '98, Association for Computing Machinery, New York, NY, USA (1998)
36. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: a proof assistant for higher-order logic. Springer (2002)
37. Nötzli, A., Barbosa, H., Niemetz, A., Preiner, M., Reynolds, A., Barrett, C., Tinelli, C.: Reconstructing fine-grained proofs of rewrites using a domain-specific language. In: Conference on Formal Methods in Computer-Aided Design (FMCAD 2022). p. 65 (2022)
38. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: International Conference on Automated Deduction. pp. 748–752. Springer (1992)
39. Owre, S., Shankar, N.: A brief overview of PVS. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) Theorem Proving in Higher Order Logics. pp. 22–27. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
40. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: International Workshop on Hybrid Systems: Computation and Control. pp. 477–492. Springer (2004)
41. RTCA Special Committee 228: DO-365: Minimum operational performance standards (mops) for detect and avoid (daa) systems. Tech. rep., RTCA, Inc. (2017), https://standards.globalspec.com/std/14562761/do-365c/
42. Schurr, H.J., Fleury, M., Desharnais, M.: Reliable reconstruction of fine-grained proofs in a proof assistant. In: CADE. vol. 28, pp. 450–467 (2021)
43. Shankar, N., Owre, S., Rushby, J.M., Stringer-Calvert, D.W.: PVS prover guide. Computer Science Laboratory, SRI International, Menlo Park, CA **1**, 11–12 (2001)
44. Slagel, J.T., White, L., Dutle, A.: Formal verification of semi-algebraic sets and real analytic functions. In: Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 278–290 (2021)
45. Titolo, L., Feliú, M.A., Moscato, M., Muñoz, C.A.: An abstract interpretation framework for the round-off error analysis of floating-point programs. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 516–537. Springer (2017)