Formal Verification for Safe AI-based Flight Planning for UAVs*

R. Bouchekir, M. Guzman
fortiss GmbH
Munich, Germany
{bouchekir, guzman}@fortiss.org

A. Cook, J. Haindl, R. Woolnough
Third party
Munich, Germany
Alasdaircook2014@gmail.com
johannes.haindl@gmx.de
Riqaq.Woolnough@ferchau.com

***Abstract*—Planning and decision-making, especially the planning of collision-free paths, are an integral part of the operation of Unmanned Aerial Vehicles (UAVs). Formalisms like Temporal Plan Networks (TPN) can be used to provide optimal flight plans for UAVs. However, ensuring that the generated flight plans are safe can be a complex task, depending on the method used to generate the plans. Safe in this context means that the next planned action for the UAV does not violate safety constraints, for example, no-fly zones (NFZ). In this paper, we investigate the application of formal methods to generate metrics that could be used as assurance evidence in the argumentation of the safety of the planning component. In particular, we make use of Satisfiability Modulo Theories (SMT)-based verification to verify low-level requirements, together with Program Verification System (PVS) to check the design requirements of the AI-based planning component.**

***Index Terms*—Formal verification, theorem proving, SMT-based verification, Temporal plan network, Remotely Piloted Aircraft (RPA), Flight planning component.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have been employed, with great effectiveness, in a diverse range of applications from cargo transportation, to search-and-rescue operations or environmental monitoring [16]. However, the main challenge of using UAVs is related to how we can gain sufficient confidence to ensure that UAVs will perform the tasks safely, i.e. the risk of human harm is as low as reasonably practicable. Industrial safety standards in safety-critical domains require developers of critical systems to provide safety assurance and safety evidence based on guidance and standards objectives. Therefore, following certification standards for safe UAVs planning components is crucial.

The flight planning component used is provided by an industrial third party and it is based on the Temporal Plan Network (TPN) [2]. TPN solution finder makes use of traditional AI-based search algorithms to find an optimal path from a specific *start waypoint* to a *goal waypoint*. Certification of this AI-based flight planning is a complex task. In planning, we need to be sure that all the outputs coming from the planning component are safe. Safe in this context means that the next planned action for the UAV does not violate safety constraints, e.g. no-fly zones (NFZ). Other safety concerns in avionics include resource consumption (e.g. safety constraints checking fuel consumption), uncertainty due to malfunctioning components, and adversarial weather conditions.

In this paper, we propose the use of formal methods to generate metrics that could be used as assurance evidence in the argumentation of the safety of the planning component to avoid geo-fence constraints. We consider two types of geo-fence, *static* and *dynamic* NFZ. We based our safety argumentation on the common standards used in avionic DO-178C and DO-333 [7], where we demonstrate how to use formal methods to generate assurance evidence that aligns with the objectives outlined in these relevant standards. The assurance evidence we provide is based on the use of two different approaches based on formal methods. The first one makes use of Satisfiability Modulo Theories (SMT) [12] to verify low-level requirements (i.e. requirements related to the algorithms in the planner implementing the logic avoiding NFZ). Additionally, we make use of the Program Verification System (PVS) theorem prover [14] to complement the verification of low-level requirements checking with design requirements (e.g. type checking tasks, non-zero divisions, etc). Notice that in PVS we can also check properties regarding the behavior of the algorithms implemented, and the evidence is presented as mathematical proofs. The evidence generated through formal methods helps us to identify errors that could require changes at the design level in the generation of the code. In addition, we illustrated via GSN (Goal Structuring Notation) [15], how the generated assurance evidence can be used to argument about the safety of the planning component.

In the following sections, we structure our work as follows. Section II describes UAV planning component and its concept of operation. Section III highlights our safety case for a safe flight planning component. Section IV gives a detailed explanation of the use of SMT-based verification and PVS theorem proving to generate assurance evidence. Section V represents our safety argument. We conclude the paper in Section VI with a summary of contributions and the scope of future work.

* This work was supported by the German Federal Ministry for Economic Affairs and Energy (BMWi) under the LuFo VI-1 program, project KIEZ4-0.

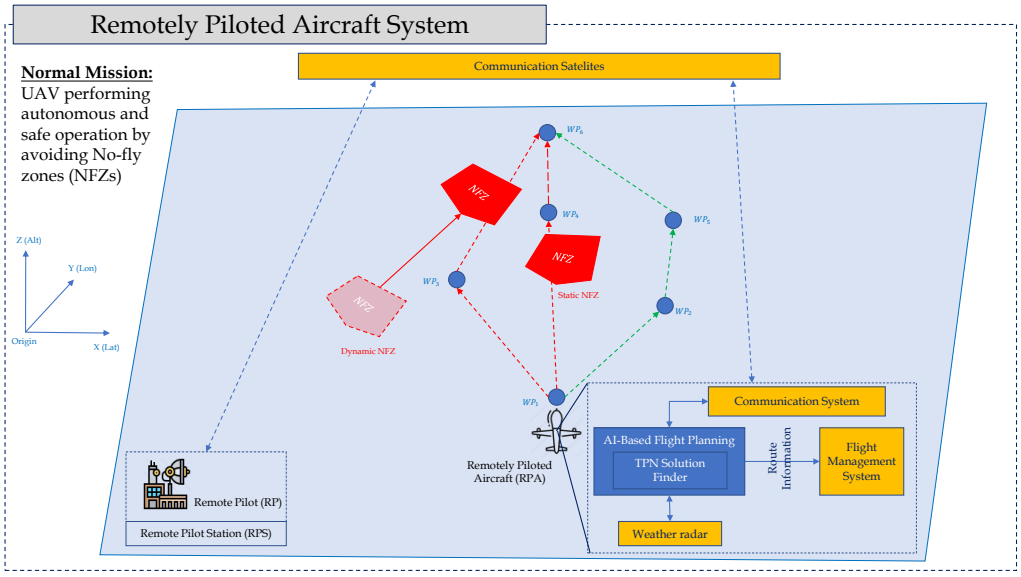

Fig. 1. Concept of Operations (ConOps): Mission Operation with geo-fence constraints.

II. UAV Planning Component

A. Concept of Operations

Remotely Piloted Aircraft (RPA) is an UAV which is piloted from a remote pilot station. As shown in Fig. 1, we envision an RPA performing an autonomous mission during a normal operation where no contingency situations are encountered during operation and the operation is to be performed in a location with communication to Remotely Piloted Aircraft System (RPAS) that is expected to oversee the RPA operation. The mission involves following a sequence of waypoints, called route, generated using TPN solution finder, for example, WP_1 (Start), WP_2, WP_5, WP_6 (Goal). After reaching the goal position, the RPA will land.

The AI-based flight planning uses TPN solution finder to propose an optimal path, following a sequence of predefined waypoints, and prevent the RPA from traveling into unwanted areas called no-fly-zones (NFZ). NFZ is a term related to the zones in the airspace that are not available to fly through due to some temporary (e.g. because of weather conditions) or permanent restrictions (e.g. because of a military base in the area). Therefore, as part of the mission configuration, the TPN solution finder has NFZ pre-specified. These NFZ can be of two types:

- **Static NFZ:** correspond to restricted parts of the airspace that cannot be violated due to, typically, restrictions on the ground. For example, due to governmental buildings on the ground, military bases, airports, etc.
- **Dynamic NFZ:** correspond to restricted parts of the airspace that cannot be violated due to adversarial weather conditions, e.g. strong winds or an area affected by bad weather. As for Static NFZ, dynamic NFZ are going to be represented as geo-fences containing the coordinates of the restricted moving area, the azimuthal angle and the velocity.

B. TPN solution finder

TPN solution finder makes use of an $A^* - like$ algorithm to find the best route and suggests it as the flight plan the RPA should take from a given starting point to the goal. The TPN solution finder makes use of a formalism called *Temporal Plan Networks* (TPN) which is based on the work of reactive systems [2]. TPN are extensions of *Simple Temporal Networks* (STN), augmented with symbolic constraints and decision nodes. The nodes of a TPN represent temporal events, whereas the edges represent temporal relations. The TPN solution finder checks also for geo-fence constraints to be satisfied. Such constraints come from static NFZ and dynamic NFZ.

Our goal is to make use of formal verification methods to account for assurance evidence and metrics stating the safety assurance of TPN solution finder. In fact, a safe flight plan mainly relies on avoiding these NFZ. Based on this insight, we focus our attention on generating evidence ensuring that the routes generated by the TPN solution finder should always avoid NFZ (static & dynamic).

III. Safety Case for Safe Flight Planning Component

In this section, we describe the safety case that shall be used as the basis for the evidence to be generated. A safety case is a compelling, comprehensive, defensible, and valid justification of the system safety for a given application in a defined operating environment; it is therefore a means to provide the grounds for confidence and to assist decision-making in certification [1]. Early research in safety cases has mainly focused on their formulation in terms of claims, arguments, and evidence elements based on fundamental argumentation theories like

the Toulmin model [4]. In this paper, we use *Goal Structuring Notation* (GSN) to present our assurance arguments [3]. Our Safety argument is aligned with the objectives defined in DO-178C, the supplement DO-333 [7] and the guidance material provided by EASA in [5] to handle the regulations for the operations of RPA.

Fig. 2 illustrates the assurance workflow that we follow. Our safety case is related to keeping the flight plans generated by the TPN solution finder from violating geographical fences (NFZ). Violations of NFZ come with the risk of damaging the structure of the aircraft, which could cause victims on the ground. In autonomous systems, it is required to provide guarantees that the algorithms performing safety-critical tasks do satisfy *safety requirements*. TPN solution finder has the task of providing the autonomous vehicle with a set of waypoints that the vehicle has to follow to achieve a certain goal.

The high-level requirements are going to allow us to define the safety requirements that serve as the main goal of our safety cases. The high-level requirements are related directly to the safety of the system, in this case, the RPA. Thus, it shall stipulate what considerations need to take into account to keep the RPA safe. It is also used to define the *ConOps* that describes the functionality of the system and the assessment of the risk. In [6] the authors propose a series of regulations aiming at moderating the operation of RPA in the airspace. Such regulations consider as well geographic restrictions that can affect the flight plans of the aircraft. Article 15 in [6] defines the main features of the geographic zones. Thus, our safety case is related to keeping the TPN solution finder from violating geographical fences (NFZ). In the context of this paper, we are interested in two types of NFZ, static and dynamic NFZ. Therefore, our high-level requirement will be: *"**Safe_UAV_1:** The TPN solution finder should ensure that, for every generated route, this route should not violate geographical fences (NFZ)"*. In the next section, we describe how we apply formal verification methods to verify this requirement.

IV. Formal Verification of RPA Planning Component

DO-178C is a software standard for the development of safety-critical software used in airborne systems. DO-178C introduces three verification approaches, which are *Reviews, Analysis* and *Software Testing*, while DO-333 provides guidance on how formal methods may be used for the purpose of producing verification evidence suitable for use in certification. Our aim is to provide evidence for related objectives described in DO-333 by using formal methods. The DO-333 glossary defines formal methods as *"formal methods apply logic and discrete mathematics to model the behaviour of a system and to formally verify that the system design and implementation satisfy functional and safety properties"* [7].

- *Formal Model.* is an abstract representation of a given set of aspects of a system that is used for analysis, simulation, code generation, or any combination thereof [8].

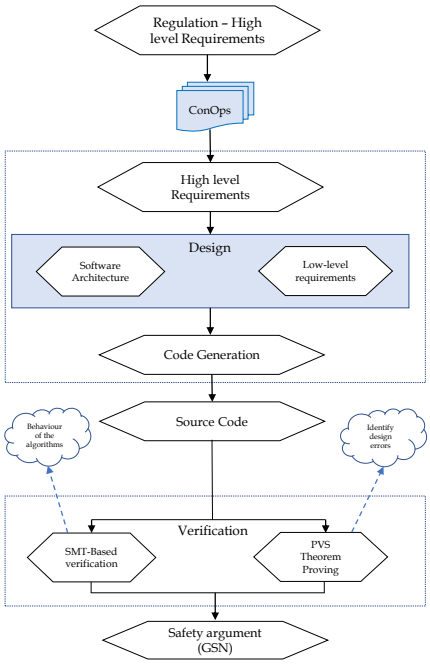

Fig. 2. Assurance workflow.

- *Formal verification of properties.* is used to prove and guarantee that software complies with the requirements [8]. In order to prove and guarantee compliance with requirements, a set of software properties is defined. When a set of software properties fully define a set of requirements, the formal analysis can be used to prove that the set of software properties hold true [7].
- *Sound verification.* The verification method may only be considered formal if its determination of property is sound. DO-333 explains this as follows: *"Sound analysis means that the method never asserts a property to be true when it is not true"*.

Table I summarises some objectives of DO-333- *Verification of Verification Objectives*. We argue about them in the next sections.

Serial #	**Objectives**
FM.A-3.8	Formal analysis cases and procedures are correct.
FM.A-3.9	Requirements formalization is correct.
FM.A-3.11	Formal method is correctly defined, justified, and appropriate.
A-3.5	High-level requirements conform to standards.
A-3.7	Algorithms are accurate

TABLE I
SUMMARY OF MAIN OBJECTIVES OF DO-333 TABLE FM.A-3: VERIFICATION OF VERIFICATION OBJECTIVES. [7]

A. SMT-based Verification

1) ***Objective FM.A-3.8****:* The objective *"formal analysis cases and procedures are correct"* is met through review to ensure that the formal method is correct. We argue about this

objective by illustrating the SMT-based verification approach and how we model the TPN solution finder, the properties, and the SMT solver used.

SMT-based verification uses Z3 SMT Solver [9] to verify the core functions of TPN solution finder. SMT problem is a decision problem for logical formulas with respect to combinations of background theories such as arithmetic, bit-vectors, arrays, and uninterpreted functions. Fig. 3 illustrates an overview of SMT-based verification plan.

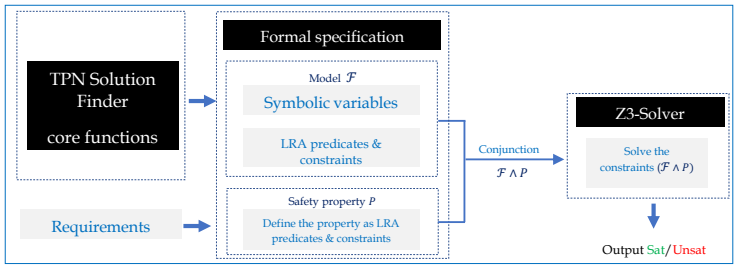

Fig. 3. Overview of the SMT-based verification plan.

Formal specification targets to formally specify the TPN solution finder core functions and the set of requirements. Formally, an SMT instance is a formula in FOL (First-Order-Logic), where some function and predicate symbols have additional predicates have additional interpretations, and SMT is a problem of determining whether a such formula is satisfiable (Sat) or unsatisfiable (Unsat). The predicates are classified according to each respective theory assigned. For instance, linear inequalities over real variables are evaluated using the rules of the theory of *Linear Real Arithmetic* (LRA) [10]. The authors of [10] define LRA as *"a logic theory that allows inequalities and equations over real numbers"*. In this verification approach, we use LRA predicates and constraints to formally specify the TPN core functions and requirements. The SMT Solver used in our approach is Z3-SMT, which is an efficient SMT Solver freely available from Microsoft Research [11]. Z3-SMT uses a linear arithmetic solver based on the algorithm proposed in [12]. The proof of the correctness of the employed algorithm is provided in [13]. As illustrated in Fig. 3, the verification process is established by formally specifying the function variables and instructions to Z3 variables and LRA constraints. The requirements are also defined as LRA constraints. The overall system will be a conjunction of these variables and constraints. The verification process will be then reduced to solve this system using Z3-SMT solver. This allows an exhaustive verification.

*2) **Objective FM.A-3.9**:* This objective is met through review to ensure that the formal properties are correct and reflect the requirements. For the TPN solution finder, the requirement is that TPN should generate a route that does not violate geographical fences. To formally specify this requirement, we consider the case of static and dynamic NFZ.

- *Formal properties.* Let's consider the property P_1 that specifies that the route generated by TPN should not violate static and dynamic NFZ. Fig. 4 illustrates an example of an intersection between edges and NFZ. Since the NFZ are configured as a set of boundaries, where each boundary point is specified by a pair (Lat, Lon) expressed in degrees, the property P_1 can be expressed as: *"none of the edges of a route should intersect with any boundary of NFZ"*. We formally specify this property as:

$$
\begin{aligned}
P_1 =& (L_{edge} \wedge L_{boundary}) \\
& \wedge (((zx_1 < x_2) \wedge (zx_1 > x_1)) \vee ((zx_1 > x_2) \wedge (zx_1 < x_1))) \\
& \wedge (((zx_1 < x_3) \wedge (zx_1 > x_4)) \vee ((zx_1 > x_3) \wedge (zx_1 < x_4)))
\end{aligned}
$$

where L_{edge} and $L_{boundary}$ are the equations defining the edge line and boundary line respectively. x_1 and x_2 defines two points of the edge line and x_3 and x_4 defines two points of the boundary line. Indeed, the coordinates are expressed in NED (North East Down) coordinate system, then converted into 2D coordinates. zx_1 represents the x-coordinate of the intersection point.

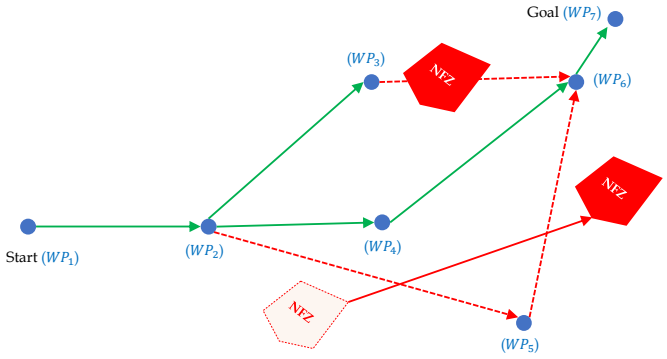

Fig. 4. TPN with static and dynamic NFZ blocking edges.

*3) **Objective FM.A-3.11**:* The objective *"Formal method is correctly defined, justified, and appropriate"* is met through review to ensure that SMT-based verification is correctly defined, justified, and appropriate. In our case, SMT-based verification allows us to verify all the possible behaviour of TPN regarding the specified properties (P_1), and this by searching a counter-example that violates the property, i.e. the set of LRA constraints.

*4) **Objective FM.A-3.5**:* the objective *"High-level requirements conform to standards"* is met through review to ensure that the requirement *Safe_UAV_1* conforms to standards. In fact, as described in section III, this requirement conforms to EASA [1] compliance and guidance described in [5]. The guidelines are based on article 15, GM3 Article 3(4) U-space airspace (Internal Geographical Zones): The U-space airspace may encompass sub-geographical zones as defined in Article 15 of Regulation (EU) 2019/947 and in the related AMC and GM: (1) zones limited in place and time, (2) zones restricted to UAS [2] operations that fulfill a specific set of conditions and

[1]European Aviation Safety Agency.
[2]Unmanned Aircraft Systems

specific authorizations and (3) ones of exclusion where UAS operations are prohibited (e.g. NFZ).

*5) **Objective A-3.7**:* The objective *"Algorithms are accurate"* aims to ensure that the SMT-based verification artifacts demonstrate that the algorithms are accurate and the implementation correctly realizes the algorithms. Here we need to illustrate how we formally specify the core functions of TPN that are used to check the NFZ. For the static NFZ, we will illustrate this process for the function `check_two_lines_intersection`, and for the dynamic NFZ, we will focus on the function `calc_dynamic_nfz_collisions`.

- *check_two_lines_intersection:* This function is one of the core function that checks static NFZ. It takes as input: $(x_i, y_j) \mid i, j \in \{1, 2\}$ and returns a Boolean value. $(x_1, y_1), (x_2, y_2)$ are used to specify the edge points and $(x_3, y_3), (x_4, y_4)$ are used to specify one boundary of the static NFZ. These coordinates are in 2D dimensions converted from NED coordinate system. We formally model this function using SE_1, SE_2, and SE_3, where SE means *Symbolic Execution* related to each possible execution branch of the function. A pseudo-code of this function is illustrated in Listing 1. The algorithm takes as input two line segments and computes two intervals with the minimum and maximum values of the points. The algorithm then checks if the x value of a possible point of intersection is in the interval previously computed. It also checks if it is possible to create the interval. Then, by using the equation of a line, the intersection point is computed and then it checks if it verifies the two segments.

```
bool check_two_lines_intersection(float x1, float y1,
     float x2, float y2, float x3, float y3, float x4,
     float y4) {
    // SE1
    float I1[2], I2[2], Ia[2];
    I1[2]={std::min(x1, x2), std::max(x1, x2)};
    I2[2]={std::min(x3, x4), std::max(x3, x4)};
    Ia[2]={std::max(std::min(x1,x2),std::min(x3,x4)),
      std::min(std::max(x1, x2), std::max(x3, x4))};
    if (std::max(x1, x2) < std::min(x3, x4)) {
        return false; // no mutual abcisses
    }
    // SE2
    float A1 = (y1 - y2) / (x1 - x2);
    float A2 = (y3 - y4) / (x3 - x4);
    float b1 = y1 - A1*x1;
    float b2 = y3 - A2*x3;
    float b11 = y2 - A1*x2;
    float b22 = y4 - A2*x4;
    assert(std::abs(b1 - b11) < 0.001);
    assert(std::abs(b2 - b22) < 0.001);
    if (std::abs(A1 - A2) < 0.001) {
        return false; // Parallel segments
    }
    // SE3
    float Xa = (b2 - b1) / (A1 - A2);
    if((Xa<std::max(std::min(x1,x2),std::min(x3,x4)))||
      (Xa>std::min(std::max(x1,x2),std::max(x3,x4)))){
        return false; // interestion out of bound
    }
    else {
        return true;
    }
}
```

Listing 1. check_two_lines_intersection

SE_1 considers the case where there are no mutual abcisses between two segments $S_1 = \{(x_1, y_1), (x_2, y_2)\}$ and $S_2 = \{(x_3, y_3), (x_4, y_4)\}$, formally:

$$\begin{aligned} SE_1 = (x_1 > x_2 \wedge x_3 < x_4 \wedge x_1 < x_3) \\ \vee (x_1 > x_2 \wedge x_3 > x_4 \wedge x_1 < x_4) \\ \vee (x_1 < x_2 \wedge x_3 < x_4 \wedge x_2 < x_3) \\ \vee (x_1 < x_2 \wedge x_3 > x_4 \wedge x_2 < x_4) \end{aligned}$$

SE_2 considers if the segments are parallel, here, the source code uses a threshold value equal to 0.001, SE_2 is formally specified as:

$$\begin{aligned} SE_2 = (((y1 - y2)/(x1 - x2) > (y3 - y4)/(x3 - x4)) \wedge \\ (((y1 - y2)/(x1 - x2)) - ((y3 - y4)/(x3 - x4)) < 0.001)) \\ \vee (((y1 - y2)/(x1 - x2) < (y3 - y4)/(x3 - x4)) \wedge \\ (((y3 - y4)/(x3 - x4)) - ((y1 - y2)/(x1 - x2)) < 0.001)) \end{aligned}$$

SE_3 checks if an intersection exists. Let $A_1 = (y_1 - y_2)/(x_1 - x_2)$, $A_2 = (y_3 - y_4)/(x_3 - x_4)$, $b_1 = y_1 - A_1 * x_1$, $b_2 = y_3 - A_2 * x_3$ and $xa = (b_2 - b_1)/(A_1 - A_2)$. SE_3 is formally specified as:

$$\begin{aligned} SE_3 = ((x_1 < x_2 \wedge x_3 < x_4 \wedge x_1 < x_3 \wedge x_3 > xa) \vee \\ (x_1 < x_2 \wedge x_3 < x_4 \wedge x_1 > x_3 \wedge x_1 > xa) \vee \\ (x_1 < x_2 \wedge x_3 > x_4 \wedge x_1 < x_4 \wedge x_4 > xa) \vee \\ (x_1 < x_2 \wedge x_3 > x_4 \wedge x_1 > x_4 \wedge x_1 > xa) \vee \\ (x_1 > x_2 \wedge x_3 < x_4 \wedge x_2 < x_3 \wedge x_3 > xa) \vee \\ (x_1 > x_2 \wedge x_3 < x_4 \wedge x_2 > x_3 \wedge x_2 > xa) \vee \\ (x_1 > x_2 \wedge x_3 > x_4 \wedge x_2 < x_4 \wedge x_4 > xa) \vee \\ (x_1 > x_2 \wedge x_3 > x_4 \wedge x_2 > x_4 \wedge x_2 > xa) \vee \\ (x_1 < x_2 \wedge x_3 < x_4 \wedge x_2 < x_4 \wedge x_2 < xa) \vee \\ (x_1 < x_2 \wedge x_3 < x_4 \wedge x_2 > x_4 \wedge x_4 < xa) \vee \\ (x_1 < x_2 \wedge x_3 > x_4 \wedge x_2 < x_3 \wedge x_2 < xa) \vee \\ (x_1 < x_2 \wedge x_3 > x_4 \wedge x_2 > x_3 \wedge x_3 < xa) \vee \\ (x_1 > x_2 \wedge x_3 < x_4 \wedge x_1 < x_4 \wedge x_1 < xa) \vee \\ (x_1 > x_2 \wedge x_3 < x_4 \wedge x_1 > x_4 \wedge x_4 < xa) \vee \\ (x_1 > x_2 \wedge x_3 > x_4 \wedge x_1 < x_3 \wedge x_1 < xa) \vee \\ (x_1 > x_2 \wedge x_3 > x_4 \wedge x_1 > x_3 \wedge x_3 < xa)) \\ \wedge ((A_1 > A_2 \wedge \neg((A_1 - A_2) < 0.001)) \vee \\ (A_1 < A_2 \wedge \neg((A_2 - A_1) < 0.001))) \end{aligned}$$

In order to verify this function, we will check the SMT formula $Sys = (SE_1 \wedge SE_2 \wedge SE_3 \wedge P_1)$ using Z3-SMT solver. This solver returns *Sat* and a *model*, which represents a counter-example. Indeed, this counter-example is due to the use of the threshold value 0.001 in the SE_2.

- *calc_dynamic_nfz_collisions:* The dynamic NFZ is calculated as:
  - Consider two objects that are moving with constant velocity, so two linear equations of motion.
  - The agent (RPA) has the position WP_i^{ned} and is moving in the direction $\Delta_{WP_iWP_j}$.
  - As the function aims to calculate the intersection point of these two moving objects, we equate these two equations per coordinate axis. NFZ^{ned} and $\Delta_{WP_iWP_j}$ can be considered as a kind of speed vector. As they do not have the correct length and speed can be varied, there are the unknown factors $z1$ and $z2$.

– After transposing the equation, we calculate z1.

TPN source code considers only the case where $z1 > 0$, therefore we check if there is no intersection in case of $z1 \leq 0$. We denote this case as SE_4. Let $\Delta_x = WP2_x - WP1_x$, $\Delta_y = WP2_y - WP1_y$, $z2 = (NFZ_x^{org} - WP1_x - NFZ_y^{org} * NFZ_x^{ned}/NFZ_y^{ned} + WP1_x * NFZ_x^{ned}/NFZ_y^{ned})/(\Delta_x - \Delta_y * NFZ_x^{ned}/NFZ_y^{ned})$ and $z1 = (WP1_x + z2 * \Delta_x - NFZ_x^{org})/NFZ_x^{ned}$. We model $NFZ_x^{ned} = cos(rad(\theta))$ and $NFZ_y^{ned} = sin(rad(\theta))$ using the constraint $C_1 : (NFZ_x^{ned} * *2 + NFZ_y^{ned} * *2) == 1$, where θ is the azimuthal angle of the NFZ. Then, SE_4 is defined formally as: $SE_4 = z1 \leq 0 \wedge C_1$. To check SE_4 we are using the same property P_1. The SMT-based verification generates a counter-example illustrating the violation of this property. The main reason is that TPN only considers the forward case where $z1 > 0$, however, the case $z1 \leq 0$ can also be realistic and lead to a violation of this property. We illustrate this counter-example in Fig. 5.

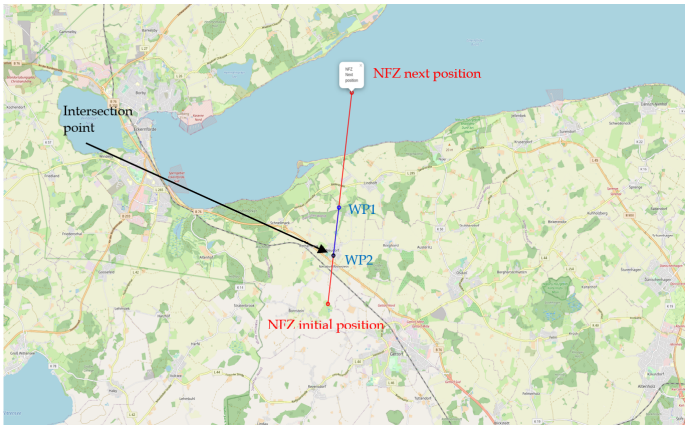

Fig. 5. Illustrating the counter-example for the dynamic NFZ.

B. PVS Theorem-Proving

We make use of theorem-proving techniques to not only check properties in the algorithms tackling the avoidance of NFZ, but also to check design implementation details, e.g. non-zero division, set emptiness, typechecking tasks, etc.

Theorem provers help us to provide the mathematical proofs for the properties we want to verify over the algorithms implementing the planning component. We make use of the Program Verification System (PVS) to assist with the verification process. PVS provides a collection of libraries with many proven properties, e.g. the trigonometric library, which facilitates the generation of the proofs, allowing us to focus on the details implementing the main no-fly avoidance properties.

The results generated would provide the evidence guaranteeing the correct behaviour of the algorithms implementing the planning component introduced in II-B. Proofs can also help us to identify bugs in the implementation, in the design, etc. To be able to perform the analysis, we shall make use of the implementation of the planning component provided in II-B.

Recall that the planning component was implemented in `C++`, aiming at satisfying the safety requirements of the use case presented in III. The safety requirement considers the avoidance of NFZ which in this context can be of two types: static or dynamic. The main algorithms implementing the planning component shall provide the input for the analysis that shall be done. Notice that these algorithms correspond to the implementation of the NFZ avoidance behaviour. The main algorithms of the implementation are divided into the following way:

- Search algorithms to find the best flight plan.
- Algorithms to deal with static NFZ.
- Algorithms to deal with dynamic NFZ.

In the context of the paper, we are not interested in providing evidence of the correct implementation of search algorithms, we just focus on the algorithms for static and dynamic NFZ.

1) PVS Prover: PVS typechecker makes use of proof obligations (type correctness conditions or TCCs) to check if the specification matches the expected types, subtypes, or subgoals. Each proof goal in PVS is represented as a *sequent* consisting of a sequence of formulas called *antecedents* and a sequence of formulas called consequents. The sequent is displayed as

$$\begin{aligned} \{-1\}\ & A_1 \\ \{-2\}\ & A_2 \\ [-3]\ & A_3 \\ & \dots \\ & \vdash \\ \{1\}\ & B_1 \\ [2]\ & B_2 \\ \{3\}\ & B_3 \end{aligned}$$

Where the A_i and B_j are sequent formulas where A_i are the antecedents and B_j are the consequents. An interpretation of the sequents is that the conjunction of antecedents implies the disjunction of consequents (i.e. $A_1 \wedge A_2 \wedge A_3 \cdots \supset B_1 \vee B_2 \vee B_3 \dots$). A PVS proof command (applied to a sequent) provides the means to build proof trees. Proof commands can be used to apply decision procedures, eliminate quantifiers, introduce lemmas, etc. The action that results from a proof command is called a proof step. A proof strategy is composed of several proof steps. For more details about proofs in PVS refer to the PVS manual [14].

2) Static no-fly zones: As introduced in Section II, static NFZ are used to state constraints in the airspace representing restrictions on the ground due to, for example, governmental buildings, military bases, airports, etc. In the implementation of the planning component, static NFZ were tackled by the algorithm in charge of checking whether a given segment of the NFZ and the segment created by the current position of the aircraft and the possible next waypoint in the flight plan intersect. The algorithm was implemented in `C++` and the function implementing the algorithm is described in Listing

1. We shall dissect each of the parts of the static algorithm and discuss what kind of properties shall we verify. We start by specifying the algorithm in linting 1 in PVS language.

3) Specification: We specify the algorithm in 1 in PVS language as follows:

```
check_two_lines_intersection: THEORY
BEGIN
min(X1:real, X2:real): real =
    IF X1 >= X2 THEN X2 ELSE X1 ENDIF
max(X1:real, X2:real): real =
    IF X1 >= X2 THEN X1 ELSE X2 ENDIF
abs(X:real): real = IF X < 0 THEN -X ELSE
X ENDIF
check_lines_intersection(X1: real, Y1:
real, X2: real, Y2: real, X3: real, Y3:
real, X4: real, Y4: real): bool=
    (max(X1, X2) >= min(X3,X4)) IMPLIES
    (X1 - X2) /= 0 IMPLIES
    (X3 - X4) /= 0 IMPLIES
    LET A1 = (Y1 - Y2) / (X1 - X2) IN
    LET A2 = (Y3 - Y4) / (X3 - X4) IN
    LET b1 = Y1 - A1*X1 IN
    LET b2 = Y3 - A2*X3 IN
    LET b11 = Y2 - A1*X2 IN
    LET b22 = Y4 - A2*X4 IN
    %instantiate the theory assert_th
    %IMPORTING assert_th[real,b1,b11]
    %IMPORTING assert_th[real, b2, b22]
    NOT abs(A1 - A2) < 0.001 IMPLIES
    NOT b1 - b11 < 0.001 IMPLIES
    (A1 - A2) /= 0 IMPLIES
    LET Xa = (b2 - b1) / (A1 - A2) IN
    % The last thing to do is check that
    % Xa is included into Ia:
    IF ((Xa < max(min(X1, X2), min(X3, X4
    ))) OR (Xa > min(max(X1, X2), max(X3,
    X4)))) THEN False ELSE True ENDIF
END check_two_lines_intersection
```

Notice that we needed to include a couple of extra conditions, i.e. `(X1 - X2) /= 0, (X3 - X4) /= 0` to avoid divisions by 0. Some extra functions were specified to help proving some additional properties. For example, the following function returns the possible intersection point.

```
get_x_intersection: THEORY
BEGIN
get_x_intersection(X1: real, Y1: real,
X2: real, Y2: real, X3: real, Y3: real,
X4: real, Y4: real): real=
    LET A1 = (Y1 - Y2) / (X1 - X2) IN
    LET A2 = (Y3 - Y4) / (X3 - X4) IN
    LET b1 = Y1 - A1*X1 IN
    LET b2 = Y3 - A2*X3 IN
    LET b11 = Y2 - A1*X2 IN
    LET b22 = Y4 - A2*X4 IN
    %instantiate the theory assert_th
    %IMPORTING assert_th[real,b1,b11]
    %IMPORTING assert_th[real, b2, b22]
    LET Xa = (b2 - b1) / (A1 - A2) IN
    % The last thing to do is check that
    % Xa is included into Ia:
    IF ((Xa < max(min(X1, X2), min(X3,
    X4))) OR (Xa > min(max(X1, X2),
    max(X3, X4)))) THEN 0.0 ELSE Xa ENDIF
END get_x_intersection
```

We specify as well the following function to help us checking violations on any of the edges of NFZ.

```
extra_properties: THEORY
BEGIN
IMPORTING check_two_lines_intersection
IMPORTING vectors@vect2D
points_nfz(N:posnat): TYPE = [below(N)
-> Vect2]
on_any_edge?(N:posnat, P:points_nfz(N),
S1:Vect2, S2:Vect2): bool=
    EXISTS(i:below(N)):
        LET nexti = (IF i < N-1 THEN
        i+1 ELSE 0 ENDIF) IN
            check_lines_intersection(
            P(i)(0),P(i)(1),P(nexti)
            (0),P(nexti)(1),
            S1(0),S1(1),S2(0),S2(1))
blocked_edge(N:posnat,
P:points_nfz(N), S1:Vect2,S2:Vect2):
bool=
    EXISTS(i:below(N)):
        LET nexti = (IF i< N-1 THEN
        i+1 ELSE 0 ENDIF) IN
            P(i)=S1 AND P(nexti)=S2
END extra_properties
```

The `on_any_edge?` PVS function returns $true$ if the segment created with the current position of the aircraft and the possible position where the aircraft is going intersect any of the segments of the NFZ.

4) Properties: Here we shall present the properties we have checked over the above specification. The following property states that there exists one unique point in which two segments intersect.

```
ax_two_lines_simple: THEORY
BEGIN
IMPORTING check_two_lines_intersection
IMPORTING get_x_intersection

X1,Y1,X2,Y2,X3,Y3,X4,Y4,X11,Y11,X22,Y22
,X33,Y33,X44,Y44 : VAR real
Ax_intersection_point_unique : AXIOM
FORALL X1,Y1,X2,Y2,X3,Y3,X4,Y4,X11,Y11,
X22,Y22,X33,Y33,X44,Y44 :
LET Xa1 = get_x_intersection(X1,Y1,X2,Y2,
X3,Y3,X4,Y4) IN
LET Xb = get_x_intersection(X11,Y11,X22,
Y22,X33,Y33,X44,Y44) IN
check_lines_intersection(X1,Y1,X2,Y2,X3,
Y3,X4,Y4) = True AND X1=X11 AND Y1=Y11
AND X2=X22 AND Y2=Y22 AND X3=X33 AND
Y3=Y33 AND X4=X44 AND Y4=Y44 IMPLIES
Xa1 = Xb
```

The following lemma states that if there is an intersection with any edge of the NFZ, then there exists a segment on the NFZ intersecting the segment created by the current position of the aircraft with the possible next position of the flight plan.

```
exist_lemma: LEMMA FORALL (N:posnat,
P:points_nfz(N), S1:Vect2, S2:Vect2):
on_any_edge?(N,P,S1,S2) IMPLIES EXISTS
(i:below(N)):
        LET nexti = (IF i< N-1 THEN i+1
        ELSE 0 ENDIF) IN
        check_lines_intersection(P(i)(0),
        P(i)(1),P(nexti)(0),P(nexti)(1),
        S1(0),
        S1(1),S2(0),S2(1))
```

In PVS automatic TCCs can be generated and proved. For example, additional TCCs can be generated to check for non-zero divisions whenever we have mathematical operations involving a division.

Properties in Table I were satisfied. In this approach, we make use of PVS as the theorem-proving tool. PVS is a well-known and established tool, containing many theories available in its library (nasalib), which helps to both make use of already-proving properties and facilitate the creation of new properties. The method we have followed corresponds to the standard and established way of proving properties in PVS,

therefore the method is correctly justified and appropriate. The high-level requirement is based on the regulations provided by the European commission (article 15). PVS provides proofs of the correctness of the algorithms based on the correct specification of the properties specified in IV-B4 regarding violations of static NFZ.

The results of the verification of the properties in IV-B4 show that the implemented algorithms satisfy the desired behaviour with respect to the avoidance of NFZ. It is important also to mention that there is no a perfect correspondence between the specification of the implementation in the original language (C++) and the one in PVS, which required to make some extra specifications for it to work.

Regarding dynamic NFZ we found out that in the original implementation there were issues regarding code design. The issues were related to non-zero divisions, operations that could have an undesired input value, e.g. square root of a negative number, there were conditions over sets not checking for emptiness (i.e. not considering cases when the sets were empty). All these conditions were added to the specification in PVS.

V. Safety Arguments

The idea of safety arguments is to detail the argumentation leading to a certain conclusion or *claim*. In the context of certification, a claim is an objective to be fulfilled by the applicant. In practice, the demonstration is based on the elicitation of requirements that correspond to the justifications that the objective is achieved. There exist several notations, either textual or graphical, to support the design of an assurance case, such as GSN [15]. As introduced in the previous section, in this paper, we use GSN to represent our safety arguments.

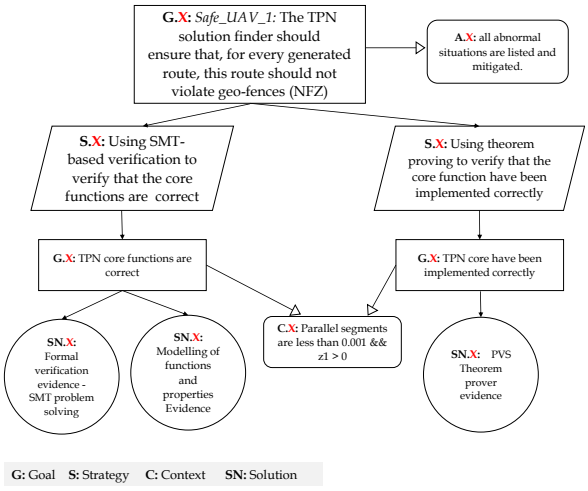

Fig. 6. A simple GSN-based interface to select the relevant solutions (Sn) from TPN required goals (G), including assumptions (A) and the agreed strategies (S) and solutions (SN).

Fig. 6 shows the reasoning to demonstrate that the TPN solution finder meets the requirement *Safe_UAV_1*. The argument is based on 2 sub-goals: the first is to demonstrate that the TPN core functions are correct regarding the requirements, we argue this by providing the assurance evidence for the modeling of the property and the results of SMT-based verification. The second sub-goal aims to demonstrate that TPN core functions have been implemented correctly.

VI. Conclusion

In this paper, we present how formal verification methods, in particular SMT-based verification and PVS theorem proving, can be used to ensure that all the outputs coming from the TPN solution finder are safe i.e., does not violate NFZ constraints. We highlighted the use of formal methods, SMT-based verification and PVS theorem proving, to achieve DO-333 verification objectives. In addition, we illustrated via GSN, how the generated assurance evidence can be used to argument about the safety of TPN solution finder.

The case illustrated in this paper is considered as a normal mission. However, other safety concerns should also be addressed, mainly in abnormal situations to check the robustness of TPN solution finder, one example could be the link-down between RPA (the planning component) and the RPAS, or sensors and GPS accuracy.

References

[1] Bloomfield, Robin, and Peter Bishop. "Safety and assurance cases: Past, present and possible future–an Adelard perspective." Making Systems Safer: Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, 9-11th February 2010. London: Springer London, 2009.

[2] Kim, Phil, Brian C. Williams, and Mark Abramson. "Executing reactive, model-based programs through graph-based temporal planning." IJCAI. 2001.

[3] . C. W. Group et al., "Goal structuring notation community standard version 2," 2018.

[4] S. Toulmin, The Uses of Argument. Cambridge University Press, 1958.

[5] European Aviation Safety Agency. Acceptable means of compliance and guidance material to part-med, 2011.

[6] Unija, Europska. "Commission implementing regulation (EU) 2019/947 of 24 May 2019 on the rules and procedures for the operation of unmanned aircraft." Official Journal of the European Union (2019).

[7] RTCA DO-333, Formal Methods Supplement to DO-178C and DO-278A, (Washington, DC:RTCA, Inc., December 2011)

[8] CRC, Developing Safety-critical software, a practical guide for aviation software and DO-178C compliance, Leanna rierson, Book CRC Press, 2013.

[9] Microsoft Z3. URL: https://www.microsoft.com/en-us/research/project/z3-3 (visited on 01.09.2023)

[10] Köhler, M. The theories of linear arithmetic and equality logic and uninterpreted functions in the context of satisfiablity modulo theories.

[11] Moura, L. D., & Bjorner, N. (2008, March). Z3: An efficient SMT solver. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 337-340). Springer, Berlin, Heidelberg.

[12] Dutertre, B., & Moura, L. D. (2006, August). A fast linear-arithmetic solver for DPLL (T). In International Conference on Computer Aided Verification (pp. 81-94). Springer, Berlin, Heidelberg.

[13] Dutertre, B., & De Moura, L. (2006). Integrating simplex with DPLL (T). Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-06-01.

[14] Shankar, Natarajan, et al. PVS prover guide. Computer Science Laboratory, SRI International, Menlo Park, CA 1 (2001): 11-12.

[15] Assurance Case Working Group et al. Goal structuring notation community standard. Rapport technique, Technical Report SCSC-141BA v2. 0, Safety Critical Systems Club, 2018. https://scsc.uk/r141C:1?t=1, 2018.

[16] UAV-Applications. https://droneii.com/237-ways-drone-applications-revolutionize-business (visited on 14/12/2022)