Proving Safety Properties of an Aircraft Landing Protocol Using I/O Automata and the PVS Theorem Prover: A Case Study[⋆]

Shinya Umeno and Nancy Lynch

Massachusetts Institute of Technology,
Computer Science and Artificial Intelligence Laboratory,
32 Vassar Street, Cambridge MA 02139, USA
{umeno, lynch}@csail.mit.edu

Abstract. This paper presents an assertional-style verification of the aircraft landing protocol of NASA's SATS (*Small Aircraft Transportation System*) concept [1] using the I/O automata framework and the PVS theorem prover. We reconstructed the mathematical model of the landing protocol presented in [2] as an I/O automaton. In addition, we translated the I/O automaton into a corresponding PVS specification, and conducted a verification of the safety properties of the protocol using the assertional proof technique and the PVS theorem prover.

1 Introduction

Safety critical systems have been the subject of intensive study of applications of formal verification techniques. As a case study, we conduct an assertional-style verification of one such safety critical system in this paper: the aircraft landing protocol that is part of NASA's SATS (*Small Aircraft Transportation System*) concept of operation [1].

The SATS program aims to increase access to small and medium sized airports. The situation is significantly different in these airports from large airports, where separation assurance services are provided by the Air Traffic Control (ATC). Due to the limited facilities and inferior infrastructure in such airports, in the SATS concept of operations, a centralized air traffic management system is automated as the Airport Management Module (AMM), and does a minimal job to achieve the safe landing of the aircraft. It is the pilots' responsibility to determine the moment when their aircraft initiate the final approach initiation to the ground. Pilots follow the procedures defined in the SATS concept of operation to control their aircraft in a designated area in the air space of the airport, called the Self Controlled Area.

It is crucial to guarantee a safe separation of the aircraft in the Self Controlled Area when each pilot follows the procedures of the SATS concept. For this reason, a mathematical model of the landing and departure protocol of

⋆ The project is supported by Air Force Contract FA9550-04-C-0084.

SATS is presented in [2]. The model is a finite-state transition system obtained from a mathematical abstraction of the real system. In addition, in the paper, some properties of the model that represent the safe separation of the aircraft have been exhaustively checked using a model-checking technique. These include properties such as a bound on the number of aircraft on a particular portion of the airport (for example, no more than four aircraft are in the entire Self Controlled Area; or there is at most one aircraft at a certain part of the airspace in the airport).

Our objective in this paper is to carry out a proof of properties of the model proposed in [2] using inductive proof techniques that have been used in computer science literature, as opposed to an exhaustive state exploration used in [2]. We used I/O automata framework [3] to reconstruct the model, and have rigorously checked all proofs in this paper using the PVS mechanical theorem prover [4].[1] I/O automata have been successfully used to model nondeterministic distributed systems and to prove properties of them. Their treatment of nondeterminism is suitable for the model in this paper in which the next possible step that the model can take is nondeterministically defined.

There are three main contributions in this study. First, we present a reconstructed mathematical model of the SATS landing protocol using the I/O automata framework. This model gives us a more standardized and comprehensive description of the protocol than the model in [2]. Second, our inductive proof brings more insight into the protocol. Though a proof of our style may cost more than a state exploration method in terms of time and manpower, it often brings us a clearer view of how the system works, and what kinds of properties are crucial for guaranteeing the required behavior of the system. In this paper, we define a notion of *blocking of aircraft* in Section 4.2, which captures an auxiliary information of why the protocol works correctly. Third, this case study demonstrates the feasibility of using a mechanical theorem prover to prove properties of a moderately large and complex system in the context of the I/O automata framework.

The paper is constructed as follows. In Section 2, we present a reconstructed mathematical model of the SATS landing protocol, both the formal definition of the actual I/O automaton and the informal explanation of how the system works. In Section 3, we introduce the seven main properties that we will prove in this paper. Section 4 is devoted to the proof of the main properties, some of which have to be strengthened to make an inductive proof work. Finally, in Section 5, we summarize the results in the paper and discuss future work.

2 Abstract Model

In this section, we present an I/O automaton model for SATS, based on the model presented in [2]. In the model, the space of the airport is discretized, and

[1] Complete I/O automata and PVS specification codes, and PVS proof scripts are available at http://theory.csail.mit.edu/∼umeno/. The full version of this paper [5] includes more detailed discussions on the model, the main properties, the auxiliary lemmas, and their proof.

is divided into several zones. These zones are represented as part of the state components of the automata, and the model can be used to check if the desirable upper bound on the number of aircraft in a specific zone is satisfied.

We will present a formal definition of the model as an I/O automaton in Section 2.5.

2.1 Logical Zones

The space of the airport used for landings is logically divided into 13 zones (see Figure 1). Each zone is modeled as a first-in first-out queue of aircraft. Only the first aircraft of a zone can move to another zone, and when an aircraft moves from one zone to another, it is removed from the head of the queue that it leaves, and is added to the end of the queue that it joins. Some zones have a symmetric structure with respect to the left side and the right side, for instance, holding3(right) and holding3(left).[2]

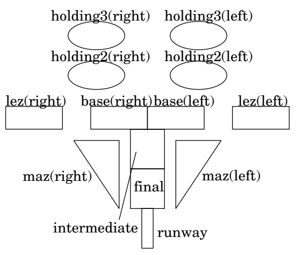

Fig. 1. 13 logical zones in SATS

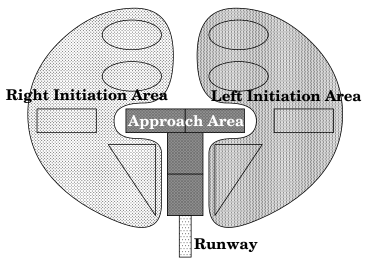

Fig. 2. Logical zones divided into four areas

For the sake of an easier understanding of the big picture of how each zone is used, we group these 13 zones into the following four areas, depending on how they are used in the system (see Figure 2). The *left initiation area* consists of holding3(left) and holding2(left), which represent the zones to hold the aircraft at 3000 feet and 2000 feet, respectively, and which are used for the vertical approach initiation from the left side of the airport; lez(left) (*lateral entry zone*), which is used for the lateral approach initiation from the left side; and maz(left) (*missed approach zone*), which is used as the path that an aircraft that has missed the approach goes through to initiate the approach operation again. The *right initiation area* is a symmetric counterpart of the left initiation area, and is analogously defined. The *approach area* consists of base(right), base(left), intermediate, and final, which make a T-shaped area for the aircraft to land. The *runway* consists of zone runway. We say that an aircraft is *on the approach* if it is in the approach area. In addition, we often refer to the combined area of the two initiation areas and the approach area (thus, it consists of all logical zones except for runway) as the *operation area*. Actually, this area is the abstraction of

[2] Note that this right and left are determined with respect to a pilot's view; thus it is the opposite to what we actually see in the picture (for instance, holding3(right) is on the *left* side in the picture.).

the Self Controlled Area that we mentioned in Section 1. In this paper, we focus on the safety conditions in the operation area.

2.2 Aircraft

An aircraft is defined as a tuple that has two attributes: the mahf assignment, which will be explained shortly, mahf of type Side (an enumeration of left and right); and a unique ID, id, which is encoded as a natural number in the abstract model.

```
Aircraft tuple [
      mahf: Side, % Missed approach holding fix assignment.
      id  : ID  ] % ID of the aircraft
```

2.3 Landing Sequence

When an aircraft enters the system, the system (AMM) assigns its *leader* aircraft, or the aircraft it has to follow. This relation of a leader constructs a chain: the first aircraft that enters the system does not have a leader, the second aircraft that enters the system is assigned the first aircraft as the leader, the third one is assigned the second one as the leader, and so on. A leader is an important notion of the system since it is used as a part of the conditions to decide if an aircraft can initiate the final approach to the ground. As we will examine closely later, an aircraft cannot go to the approach area until its leader has gone there. We will see formally defined conditions in Section 2.5.

In our abstract model, we encode this notion of the leader aircraft as an explicit queue of aircraft, called the "*landing sequence.*" When an aircraft enters the operation area, it is also added to the end of the landing sequence. We define the leader of aircraft a in the landing sequence as the aircraft just in front of a in the sequence. By this definition of the leader, this abstract sequence represents the chain of the *leader* relation in reality discussed above. When an aircraft lands or exits from the operation area, it is removed from the landing sequence.

The assignment of the leader for an aircraft does not change once it is assigned if that aircraft lands successfully in the first try. However, an aircraft does not always succeed in landing at the first attempt, that is, it may miss the approach. In such a case, its leader is *reassigned* and it has to redo the landing process. We will later look at the case when an aircraft misses the approach.

2.4 Paths of Aircraft

Here we present a high level picture of how an aircraft enters and moves in the logical zones, initiates the approach to the ground, and lands on the runway. All movements are represented as the transitions of the model. We refer to the corresponding transitions' names in parentheses when explaining the movements of aircraft in the following. In Section 2.5, we will examine the details of the important transitions.

An aircraft can enter the logical zones by entering either holding3 (VerticalEntry) or lez (LateralEntry) of either side. An aircraft that has entered holding3 descends to holding2 of the same side (HoldingPatternDiscend), and initiates the approach to the ground from there (VerticalApproachInitiation). An aircraft that has entered lez can go directly to the approach area if specific conditions are met; otherwise, it first goes to holding2 (LateralApproachInitiation). Every aircraft that initiates the approach first goes to the base zone of the same side where it initiates the approach. Once aircraft enter base, they merge into intermediate (Merging), then proceed to final (FinalSegment) and land on runway (Landing). This progression of the movement of aircraft is depicted in Figure 3.

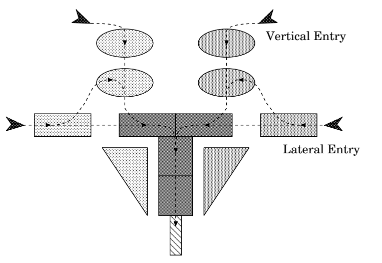

Fig. 3. Paths of aircraft

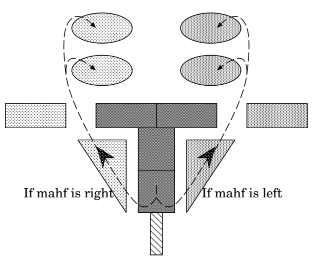

Fig. 4. Paths of aircraft that have missed the approach

An aircraft may miss the approach to the ground at the final zone. In such a case, it once again goes back to a zone where it can initiate the approach again, and make the next try to land.

An aircraft has to determine the side of an initiation area to which it has to go in case it misses the approach. For this purpose, the assignment of the side, called the "*mahf (missed approach holding fix)*" is given by the AMM to an aircraft when it enters the system, based on a system variable nextmahf. The variable nextmahf is of type Side, and is used by the AMM to keep track of the last assignment of mahf to aircraft that have entered the system. The system flips the value of nextmahf, either from left to right or vice versa, every time it assigns the mahf to an aircraft. This produces an *alternate assignment* of the left side and the right side to the aircraft in the landing sequence.

In the logical zones, a missed aircraft, with the reassignment as stated above, first goes to maz of the side assigned as its mahf (MissedApproach), and from there it goes back to either holding2 or holding3 of the side of maz where it leaves (LowestAvailableAltitude). Whether it goes to holding2 or holding3 is determined by the situation at the moment it leaves maz. These paths for aircraft that have missed the approach are shown in Figure 4.

2.5 Transitions

Twelve transitions are defined in the model based on the original procedures in SATS. Each one represents either a movement of an aircraft from one logical zone to another, an entry of an aircraft into the logical zones, or a removal of an aircraft from the logical zones.

Some transitions have an attribute of Side because they can be performed either from the right side or the left side of the airport. For example, VerticalApproachInitiation(right) represents the approach initiation of an aircraft from holding2(right).

Each transition has its own *precondition.* A transition can occur only when its precondition is satisfied. We say that a transition is *enabled* at a particular state of the model if its precondition is satisfied in that state.

One interesting notion in the SATS concept that the precondition of some transitions refers to is the *potential number of aircraft.* The potential number of aircraft in the initiation area of side σ counts not only the *actual* number of aircraft in that area, but also the number of *potential aircraft* that may possibly come to the area σ if they miss the approach, that is, aircraft outside of that area that are assigned σ as its mahf. The potential number of aircraft is expressed by the function virtual as follows, where assigned(zone,side) is the function to calculate the number of aircraft assigned side in zone.

```
virtual(z:zone_map,side:Side): nat =
  length(z(holding3(side))) + length(z(holding2(side))) +
  length(z(lez(side))) + length(z(maz(side))) +
  assigned(z(holding3(opposite(side))),side) +
  assigned(z(holding2(opposite(side))),side) +
  assigned(z(lez(opposite(side))),side) +
  assigned(z(maz(opposite(side))),side) +
  assigned(z(base(right)),side)+assigned(z(base(left)),side)+
  assigned(z(intermediate),side) + assigned(z(final),side)
```

To help a reader's intuition toward why the protocol has the rules represented by the preconditions of the transitions, here we briefly present some of the safety properties of the model that we will prove.

We will prove upper bounds on the numbers of aircraft in the vertical and lateral initiation areas (holding2, holding3, and lez): there is at most one aircraft in each of these zones. Now a reader may easily understand, for instance, why it is reasonable that the precondition of entry and descend transitions checks the emptiness of the zone that an aircraft goes to.

On the other hand, a more complicated precondition is defined for other transitions: for example, some preconditions refer to the potential number of aircraft, or whether the leader of the moving aircraft is in a specific area of the logical zones. We have to make use of these more complicated preconditions in order to prove the bound on the number of aircraft in some specific zone such as maz. This complication comes from the fact that, the transition representing a missed approach does not have a "guard" in a precondition that prevents the transition from being performed. This is quite reasonable, considering the real system: an aircraft cannot just assume some specific condition that prevents it from missing the approach. For this reason, some of the main properties we will prove do not immediately follow from the preconditions of the transitions, and thus we need a more intelligent way to prove them.

We present an IOA code for the SATS aircraft landing protocol in the following. It is actually described in the subset of the timed I/O automata specification language [6]. It imports a vocabulary called SatsVocab, which appear in the extended paper [5]. The vocabulary defines types and auxiliary functions that the automaton definition uses. In this paper, we give an informal description of these types and functions. The functions in_queue?(a,q) and on_zone?(q,a) are predicates that checks if aircraft a is in q. We just have two predicates to differentiate zones and sequences, which are intrinsically same in our model. We use on_approach?(a) to check if aircraft a is on the approach, and use on_approach?(side) to check if there is an aircraft assigned side in the approach area. The predicate on_zones?(a) is to check if aircraft a is in the operation area.

Here, we examine some important transitions to prove the main properties.

VerticalEntry: A newly entering aircraft is assigned its mahf from the system. As we explained before, the assignment is determined according to nextmahf (see the definition of the function aircraft). Also, a unique ID is given to a new aircraft when it enters the system. The uniqueness of its ID is guaranteed by the part of the precondition that is universally quantified. The precondition also checks the condition on the potential number of aircraft in the initiation area of the side where the new aircraft enters (virtual(*side*)), as well as the emptiness of some zones. In a real system, this information is given by the Airport Management Module, which typically resides at the airport ground.

LateralEntry: It has a definition analogous to VerticalEntry. Note, however, that the precondition checks if the value of virtual(*side*) is zero. It implies that, in the state of the model that this transition is enabled, there is no aircraft in that area, and also no aircraft assigned *side* as its mahf outside of the area.

VerticalApproachInitiation: An aircraft initiates the approach from holding2 by this transition. Note that the precondition checks if the moving aircraft is either the first aircraft of the landing sequence (first_in_seq?(a)), or its leader aircraft has already initiates the approach (that is, it is in the approach area: on_approach?(leader(a,landing_seq))). This precondition is used as the "threshold" that delays the final approach initiation to the ground until when the safe separation of the aircraft in the system is guaranteed.

LateralApproachInitiation: The transition is different from VerticalApproachInitiation, in that it is always enabled whenever lez is not empty. Nevertheless, the aircraft can directly proceed to base only when specific conditions, which are equivalent to the precondition of VerticalApproach- Initiation, are met. Otherwise, the aircraft first moves to holding2.

MissedApproach: This transition is enabled whenever final is not empty. It reflects that there is no "guard" that prevents an aircraft from missing the approach, as discussed before. A missed aircraft gets reassigned its mahf according to nextmahf (see the definition of the function reassign), and is added to one of the maz zones according to its mahf before the reassignment. In the landing sequence, the aircraft is removed from the head of the sequence, and added to the end of it with the reassignment. The variable nextmahf is flipped in this case as well, so that the alternate assignment will be preserved even in case some aircraft miss the approach.

```
automaton SATS
    imports SatsVocab

signature
    internal
        VerticalEntry(ac:Aircraft, id:ID, side:Side),
        LateralEntry(ac:Aircraft, id:ID, side:Side),
        HoldingPatternDescend(ac:Aircraft,side:Side),
        VerticalApproachInitiation(ac:Aircraft,side:Side),
        LateralApproachInitiation(ac:Aircraft,side:Side),
        Merging(ac:Aircraft,side:Side),
        Exit(ac:Aircraft),
        FinalSegment(ac:Aircraft),
        Landing(ac:Aircraft),
        Taxiing(ac:Aircraft),
        MissedApproach(ac:Aircraft),
        LowestAvailableAltitude(ac:Aircraft,side:Side)

states
    zones : zone_map, % mapping from a zone name to a zone
    nextmahf : Side, % Next missed approach holding fix
    landing_seq : queue % landing sequence is defined as a queue
    initially
        zones = initialZones ∧
        nextmahf = right ∧
        landing_seq = empty

    let
        %% access to the state components
        holding3(side: Side) = zones[holding3(side)];
        holding2(side: Side) = zones[holding2(side)];
        lez(side: Side) = zones[lez(side)];
        maz(side: Side) = zones[maz(side)];
        base(side: Side) = zones[base(side)];
        intermediate = zones[intermediate];
        final = zones[final];
        runway = zones[runway];

        %% first aircraft in the landing sequence?
        first_in_seq?(a:Aircraft) = (a = first(landing_seq));

        %% definig functions on a zone_map as functions on a state
        on_approach?(a:Aircraft) = on_approach?(zones, a);
        on_approach?(side:Side) = on_approach?(zones,side);
        actual(side:Side) = actual(zones,side);
        virtual(side:Side) = virtual(zones,side);

        %% new aircraft
        aircraft(side:Side, id_:ID) = [IF empty?(landing_seq) THEN side ELSE nextmahf, id_];

        %% reassign aircraft
        reassign(a:Aircraft) = set_mahf(a, IF empty?(landing_seq) THEN a.mahf ElSE nextmahf);

        %% the first aircraft of z_from moves to z_to in zones_
        move(z_from, z_to: z_name, zones_: zone_map | z_from ≠ z_to ∧ ¬ empty?(z_from)) =
            assign(assign(zones_, z_to, add(zones_[z_to], first(zones_[z_from]))),
                    z_from, rest(zones_[z_from]))

        %% new aircraft enters a zone
        enter(z_enter: z_name, side:Side, id:ID, zones_:zone_map) =
            assign(zones_, z_enter, add(zones[z_enter], aircraft(side,id)));
```

transitions

```
internal VerticalEntry(a, id, side)
pre virtual(side) < 2 ∧
    ¬on_approach?(side) ∧
    empty?(maz(side)) ∧
    empty?(lez(side)) ∧
    empty?(holding3(side)) ∧
    a = aircraft(side,id) ∧
    ∀ac: Aircraft
     ((on_zones?(ac) ∨
        in_queue?(ac, landing_seq) ∨
        on_zone?(runway, ac)) ⇒ ac.id ≠ id)
eff zones := enter(holding3(side),side,id,zones);
    landing_seq := add(landing_seq, a);
    nextmahf := opposite(a.mahf);

internal LateralEntry(a, id, side)
pre virtual(side) = 0 ∧
    a = aircraft(side,id) ∧
    ∀ac: Aircraft
     ((on_zones?(ac) ∨
        in_queue?(ac, landing_seq) ∨
        on_zone?(runway, ac)) ⇒ ac.id ≠ id)
eff zones := enter(lez(side),side,id,zones);
    landing_seq := add(landing_seq,a);
    nextmahf := opposite(a.mahf);

internal HoldingPatternDescend(a, side)
pre ¬(empty?(holding3(side))) ∧
    a = first(holding3(side)) ∧
    empty?(holding2(side))
eff zones:=
    move(holding3(side),holding2(side),zones)

internal VerticalApproachInitiation(a, side)
pre ¬(empty?(holding2(side))) ∧
    a = first(holding2(side)) ∧
    length(base(opposite(side))) ≤ 1 ∧
    (first_in_seq?(a) ∨
        on_approach?(leader(a,landing_seq)))
eff zones :=
    move(holding2(side),base(side),zones)

internal LateralApproachInitiation(a, side)
pre ¬(empty?(lez(side))) ∧
    a = first(lez(side))
eff IF length(base(opposite(side))) ≤ 1 ∧
        (first_in_seq?(a) ∨
        on_approach?(leader(a,landing_seq)))
    THEN
        zones :=
        move(lez(side),base(side),zones)
    ELSE
        zones :=
        move(lez(side),holding2(side),zones)
    FI

internal Merging(a, side)
pre ¬(empty?(base(side))) ∧
    a = first(base(side)) ∧
    (first_in_seq?(a) ∨
    on_zone?(intermediate,
            leader(a,landing_seq)) ∨
    on_zone?(final,leader(a,landing_seq)))
eff zones := move(base(side),intermediate,zones)

internal Exit(a)
pre ¬(empty?(intermediate)) ∧
    ¬(empty?(landing_seq)) ∧
    a = first(intermediate) ∧
    first_in_seq?(a)
eff zones:=
    assign(zones,intermediate,rest(intermediate));
    landing_seq := rest(landing_seq)

internal FinalSegment(a)
pre ¬(empty?(intermediate)) ∧
    a = first(intermediate)
eff zones := move(intermediate, final, zones)

internal Landing(a)
pre ¬(empty?(final)) ∧
    ¬(empty?(landing_seq)) ∧
    a = first(final) ∧
    empty?(runway)
eff zones := move(final,runway,zones);
    landing_seq := rest(landing_seq);

internal Taxiing(a)
pre ¬(empty?(runway)) ∧
    a = first(runway)
eff zones:= assign(zones, runway, rest(runway));

internal MissedApproach(a)
pre ¬(empty?(final)) ∧
    ¬(empty?(landing_seq)) ∧
    a = first(final)
eff zones:= assign(zones, final, rest(final));
    zones:= assign(zones, maz(a.mahf),
        add(maz(a.mahf),reassign(a)));
    landing_seq :=
        add(rest(landing_seq),reassign(a));
    nextmahf := opposite(reassign(a).mahf);

internal LowestAvailableAltitude(a, side)
pre ¬(empty?(maz(side))) ∧
    a = first(maz(side))
eff IF empty?(holding3(side)) ∧
        empty?(holding2(side))
    THEN
        zones :=
        move(maz(side),holding2(side),zones)
    ELSE
        IF empty?(holding3(side)) THEN
            zones :=
            move(maz(side),holding3(side),zones)
        ELSE
            zones :=
            move(maz(side),holding3(side),
            move(holding3(side),holding2(side),
            zones))
        FI
    FI
```

3 The Main Properties

In this section, we present the main properties that represents the safe separation of aircraft. There are seven properties taken from the original paper [2]. In PVS, each property is expressed as a predicate over the states, and is declared as an invariant as follows:

```
Invariant_#: LEMMA ( FORALL (s:states): reachable(s) => Inv#(s));
```

where Inv# is the predicate that expresses the property, and # is replaced by the actual number of the property. In the following, we describe the seven properties, along with the corresponding predicates in PVS. The predicate reachable(s) checks if s is a reachable state of the system.

Property 1: The total number of aircraft in the operation area (represented by arrival_op; a formal definition is in [5]) is *at most four.*

```
Inv1(s:states):bool = arrival_op(s) <= 4
```

Property 2: The total number of aircraft in each initiation area is *at most two.*

```
Inv2(s:states):bool = FORALL (side:Side): actual(s,side) <= 2
```

Property 3: The number of aircraft in each vertical holding fix (holding2 and holding3 of each side) is *at most one.*

```
Inv3(s:states):bool = FORALL (side:Side):
   length(holding3(side,s)) <= 1 AND length(holding2(side,s)) <= 1
```

Property 4: The number of aircraft on a missed approach zone (maz(right) and maz(left), respectively) is *at most two.*

```
Inv4(s:states):bool = FORALL (side:Side): length(maz(side,s)) <= 2
```

Property 5: The number of aircraft on a lateral entry zone (lez(right) and lez(left), respectively) is *at most one.*

```
Inv5(s:states):bool = FORALL (side:Side): length(lez(side,s)) <= 1
```

Property 6: If a lateral entry zone of side σ (lez(σ)) is not empty, holding2(σ), holding3(σ), and maz(σ) are all empty.

```
Inv6(s:states):bool = FORALL (side:Side):
   NOT(empty?(lez(side,s))) IMPLIES empty?(holding2(side,s)) AND
                                    empty?(holding3(side,s)) AND
                                    empty?(maz(side,s))
```

Property 7: The total number of aircraft assigned to one side as their mahf in the operation area (represented by assigned2fix; a formal definition is in [5]) is *at most two.*

```
Inv7(s:states):bool = FORALL (side:Side): assigned2fix(s,side)<=2
```

4 Proof of the Properties

Almost all properties are proved by induction over steps of the abstract model (the length of the sequence of transitions the model ever takes), some of which need to be strengthened to make an inductive proof work.

It turns out that some properties depend on other properties, and thus we have to prove them in such an order that a proof of each property just depends on the properties that have been proved. Because of this, the order of the proof in this section does not exactly match the numbering of the properties.[3]

4.1 Properties Part 1: Properties That Can Be Proved Without a Strengthening

In this subsection, we prove the properties that can be proved straightforwardly by induction without strengthening them (Properties 1, 7, and 5).

Theorem 1. (Property 1) *For any reachable state of the abstract model, the number of aircraft in the operation area is at most four.*

Proof. By induction. The base case is easy to prove.

[Induction step]: From the induction hypothesis, the number of aircraft in the operation area is at most four in the pre state. Two transitions, VerticalEntry and LateralEntry, add an aircraft to the operation area.

First, consider the case that VerticalEntry(side) is performed. If the number of aircraft in the area is strictly less than four in the pre state, the condition holds since the transition just adds one aircraft to the area. Now suppose the number of aircraft in the area is exactly four in the pre state. From the fact that the assignments of the mahf alternate in the landing sequence, it follows that there are exactly two aircraft assigned to each side. It implies that the value of virtual(side) is at least two in the pre state considering that, from the definition of virtual, the value is always more than or equal to the number of aircraft assigned σ. This contradicts virtual(side)¡2 from the precondition.

In the case that LateralEntry(side) is performed, we can prove the condition analogously to the case of VerticalEntry(side) using the fact that the transition checks if the value of virtual(side) is zero.

Theorem 2. (Property 7) *For any reachable state of the abstract model and a side σ, the number of aircraft on the operation area assigned σ as their mahf is at most two.*

Proof. From theorem 1 (Property 1), the number of aircraft in the operation area is at most four. Since the aircraft get alternate assignments, the number of assignments to one side is at most two.

Theorem 3. (Property 5) *For any reachable state of the abstract model and a side σ, the number of aircraft on* lez(σ) *zone is at most one.*

Proof. By induction. We prove it for an arbitrary side σ. The base case is easy.

[Induction step]: From the induction hypothesis, the number of aircraft in lez(σ) is at most one. The only transition that increases the number of aircraft in the zone is LateralEntry(σ). From the precondition of it, the value of virtual(σ) is zero. It implies that there is no aircraft in lez(σ) before the transition. Thus the bound holds after the transition.

[3] Since we did not know what the order of the proof should be when we defined these properties in PVS, we just listed the properties in the order as appear in this paper. Though we could have re-numbered the properties so that it matches up the order of the proof, in order to maintain the consistency with the code in PVS, we numbered them in the same order as the code.

4.2 Blocking of Aircraft

In order to prove the rest of the properties, we have to strengthen them using a notion of *blocking of aircraft* introduced in this subsection. To see an example of why an inductive proof of the properties does not work without a strengthening, let us consider Property 2. As we mentioned in Section 2.5, there is no "guard" to prevent an aircraft from missing the approach (MissedApproach is enabled whenever the final zone is not empty). Thus if there are already two aircraft in the right initiation area, for example, and there is an aircraft assigned right in final, the bound would be violated by the MissedApproach transition.

One might consider strengthening the condition using the *potential number* of aircraft introduced in Section 2, instead of using the actual number of aircraft. Since the potential number is always greater than or equal to the actual number, we could prove the property by proving the bound on the potential number. However, this approach would not work, since the potential number *can* exceed two in some reachable states, as depicted in Figure 5. In the state depicted in the figure, the potential number of aircraft in the right initiation area is *three.*

Even if the potential number of aircraft exceeds two, the above scenario would not jeopardize Property 2. The potentially problematic scenario is that c initiates and misses the approach after the situation in the figure. However, this scenario would not happen because aircraft c has the leader aircraft b. From this fact and the rule of the approach initiation, the leader b has to leave the right initiation area *before* c initiates the approach. In other words, the approach initiation of aircraft c is "*blocked*" until b initiates the approach. This example leads to a notion of *blocking* of aircraft. That is, if all aircraft in the left side are either assigned left, or are preceded by some other aircraft b in the landing sequence, no aircraft assigned right can initiate the approach from the left side until the blocking aircraft b initiates the approach.

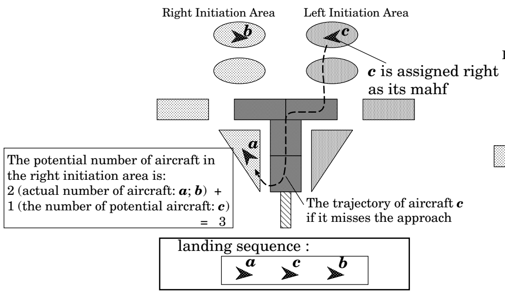
Right Initiation Area
Left Initiation Area
b
c
c is assigned right as its mahf
a
The potential number of aircraft in the right initiation area is: 2 (actual number of aircraft: a; b) + 1 (the number of potential aircraft: c) = 3
The trajectory of aircraft c if it misses the approach
landing sequence :
a c b

Fig. 5. The potential number of aircraft on the right initiation area is more than two

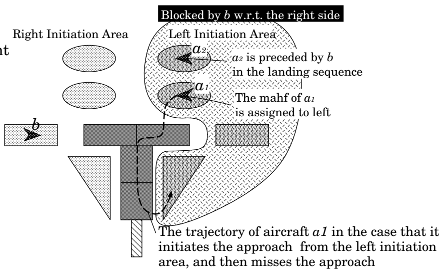
Blocked by b w.r.t. the right side
Right Initiation Area
Left Initiation Area
a2
a1
a2 is preceded by b in the landing sequence
The mahf of a1 is assigned to left
b
The trajectory of aircraft a1 in the case that it initiates the approach from the left initiation area, and then misses the approach

Fig. 6. The left initiation area is blocked by the first aircraft of lez(right)

The formal definition of blocking of aircraft in PVS is as follows, where precedes?(a,b,q) checks if aircraft a precedes aircraft b in sequence q, and on?(side,a,s) checks if aircraft a is in the initiation area of side in state s.

The first predicate represents the blocking condition between two aircraft (a is blocked by b), and the second predicate represents the blocking condition that implies all aircraft assigned side in the initiation area of the opposite side of side cannot initiate the approach until the blocking aircraft b initiates the approach. See Fig. 6 for an example of the blocked initiation area.

```
blocked_by?(a,b:Aircraft, side:Side, s:states):bool =
   mahf(a) = opposite(side) OR
   precedes?(b, a, landing_seq(s))
blocked_opposite_side?(b:Aircraft, side:Side, s:states):bool =
   Forall (a:Aircraft):
      on?(opposite(side),a,s) IMPLIES blocked_by?(a,b,side,s)
```

4.3 Properties Part 2: Strengthening Property 6

In this subsection, we strengthen Property 6 using the blocking condition defined in the previous subsection. We also presents a proof sketch of the strengthened property.

Consider proving Property 6 by induction for an arbitrary side σ. We have to ensure that there is no aircraft assigned σ in the approach area, since otherwise, one missed approach would violate the condition. Now in turn, to prove this condition, we have to guarantee that no aircraft assigned σ will initiate the approach when lez($sigma$) is not empty. Thus we need a blocking condition to hold in order to prevent such an approach initiation from the opposite side of σ. From the above discussion, we strengthen Property 6 as follows.

```
Lem1(s:states):bool = FORALL (side:Side):
   NOT (empty?(lez(side,s))) IMPLIES
        empty?(holding2(side,s)) AND empty?(holding3(side,s)) AND
        empty?(maz(side,s)) AND
        NOT on_approach?(s,side) AND
        blocked_opposite_side?(first(lez(side,s)),side,s)
```

Lemma 4. (Strengthened Property 6) *For any reachable state of the abstract model, the strengthened Property 6 holds.*

```
Lemma_1: LEMMA ( FORALL (s:states): reachable(s) => Lem1(s));
```

Proof. By induction. We prove it for an arbitrary side σ. The base case is easy. [Induction step]: In the case of LateralEntry(σ): It adds a new aircraft to lez(σ). The precondition of the transition ensures that virtual(σ)=0. It implies that there is no aircraft in either holding3(σ), holding2(σ), or maz(σ), and there is no aircraft assigned σ outside of the initiation area of side σ. The condition follows from these facts.

In the case of VerticalEntry(σ): The precondition checks if lez(σ) is empty. Thus the transition is disabled when lez(σ) is not empty.

In the case of MissedAppraoch(σ): From the induction hypothesis, all aircraft in the approach area are assigned opposite(σ). Thus the missed aircraft goes to maz(opposite(σ)), and hence maz(σ) is not affected by the transition.

In the case of VerticalApproachInitiation(opposite(σ)): The initiation area of opposite(σ) is blocked in the pre state. It follows that the aircraft that initiates the approach must be assigned opposite(σ), since otherwise it violates the order of the approach initiation. Thus NOT on_approach? is preserved.

The rest of the cases are easy to prove, using some auxiliary lemmas that state that the blocking condition is preserved by some specific transitions. (See [5] for more details).

4.4 Properties Part 3: The Key Lemma, and the Remaining Properties

In this section, we present a key lemma to prove the rest of the main properties. The lemma has the longest and most complex statement, and the proof of it is also complicated because of the substantial number of case analyses and discussions on the blocking condition. It consists of nine conditions, where two of them are from main properties, Properties 3 and 4, and the remaining seven conditions construct case analyses of the blocking situation. The formal description of the lemma appears in the next page.

The condition on each of these cases has a form analogous to the strengthened Property 6 proved in the previous subsection. Indeed, they are from the same philosophy: Consider proving Property 3 – the number of aircraft in one maz zone is at most two – by induction. Analogous to Property 6, when there are already two aircraft in maz(σ) for side σ, we have to guarantee that there is no aircraft assigned σ in the approach area, since otherwise one missed approach would violate the bound. Now, to ensure the above fact, we need a blocking condition for the initiation area of the opposite side of σ. The conditions from this discussion are represented in Case 1 of the lemma.

In the strengthened Property 6, we only have to consider one situation, as opposed to the multiple (seven) cases in this lemma. This is because the number of aircraft in lez increases just by LateralEnty, and this transition has a strict examination of the safe separation in its precondition: the potential number of aircraft in the side of entry must be zero. As we saw in the proof sketch of the strengthened Property 6, this precondition directly implies the required blocking condition.

In contrast, the number of aircraft in maz increases by MissedApproach, and as we have stated, this transition has no "guard" in its precondition to examine the current situation. It implies that we need an analogous blocking condition to hold in the pre state before MissedApproach is preformed. For this purpose, we need Case 2, which has a form analogous to Case 1, but represents the situation just before the number of aircraft in maz gets two by MissedApproach. Analogously, we need more cases to support Case 2, and then new cases to supports those cases, and so on. This iteration of finding cases ends when we reach a point where where we can guarantee the blocking condition in one case from another case that has been discovered, or from other properties that have been proved.

Following the above strategy, we constructed the seven cases, all of which depend on each other: we need some of the seven cases or two properties as an induction hypothesis to prove every single case. This is why the seven cases and two properties are defined as one lemma, and are proved together at the same time. Note that the blocking aircraft differs depending on the cases. That is, different cases uses the blocking aircraft in different positions. This represents the fact that the blocking aircraft can move by the transition, and thus we have to match up the blocking aircraft between the pre and post state.

```
%% case 1: two aircraft are in maz
Lem2_case1(s:states,side:Side):bool =
     length(maz(side,s))=2 IMPLIES
          empty?(holding2(side,s)) AND empty?(holding3(side,s)) AND
          NOT on_approach?(s,side) AND
          LET a1 = first(maz(side,s)) IN        %% first  aircraft in maz
          LET a2 = first(rest(maz(side,s))) IN  %% second aircraft in maz
          LET a  = IF mahf(a1) = side THEN a2 ELSE a1 ENDIF IN
          blocked_opposite_side?(a,side,s)
%% case 2: one aircraft is in maz and some aircraft assigned 'side' are on approach.
Lem2_case2(s:states,side:Side):bool =
     length(maz(side,s))=1 AND on_approach?(s,side) IMPLIES
          assigned_approach(s,side) <= 1 AND
          LET a1 = first(maz(side,s)) IN
          blocked_opposite_side?(a1,side,s)
%% case 3: one aircraft is in maz and some aircraft are in holding2/3
Lem2_case3(s:states,side:Side):bool =
     length(maz(side,s))=1 AND
    (NOT (empty?(holding2(side,s))) OR NOT (empty?(holding3(side,s))))
       IMPLIES
      length(holding2(side,s)) + length(holding3(side,s)) <= 1 AND
          NOT on_approach?(s,side) AND
          LET a1 = IF NOT (empty?(holding2(side,s)))
                        THEN first(holding2(side,s))
                        ELSE first(holding3(side,s)) ENDIF IN
          LET a2 = first(maz(side,s)) IN
          LET a  = IF mahf(a1) = side THEN a2 ELSE a1 ENDIF IN
          blocked_opposite_side?(a,side,s)
%% case 4: some aircraft assigned 'side' are on approach, and
%%         some aircraft are in hoding2/3.
Lem2_case4(s:states,side:Side):bool =
     (NOT (empty?(holding2(side,s))) OR NOT (empty?(holding3(side,s)))) AND
     on_approach?(s,side)
       IMPLIES
          length(holding2(side,s)) + length(holding3(side,s)) <= 1 AND
          empty?(maz(side,s)) AND
          assigned_approach(s,side) <= 1 AND
          LET a1 = IF NOT (empty?(holding2(side,s)))
                        THEN first(holding2(side,s))
                        ELSE first(holding3(side,s)) ENDIF IN
          blocked_opposite_side?(a1,side,s)
%% case 5: both holding2 and holding3 are not empty.
Lem2_case5(s:states,side:Side):bool =
     (NOT (empty?(holding2(side,s))) AND NOT (empty?(holding3(side,s)))) IMPLIES
          empty?(maz(side,s)) AND
          NOT on_approach?(s,side) AND
          LET a1 = first(holding2(side,s)) IN
          LET a2 = first(holding3(side,s)) IN
          LET a  = IF mahf(a1) = side THEN a2 ELSE a1 ENDIF IN
          blocked_opposite_side?(a,side,s)
%% case 6: there is an aircraft that is assigned 'side' and is not blocked
%%         in the opposite side, and some aircraft are in h2/h3
Lem2_case6(s:states,side:Side):bool =
     LET a1 = IF NOT (empty?(holding2(side,s)))
                  THEN first(holding2(side,s))
                  ELSE first(holding3(side,s)) ENDIF IN
     (NOT (empty?(holding2(side,s))) OR NOT (empty?(holding3(side,s)))) AND
     ac_ready_to_approach?(side,s)
       IMPLIES
          length(holding2(side,s)) + length(holding3(side,s)) <= 1 AND
          empty?(maz(side,s)) AND
          NOT on_approach?(s,side) AND
          blocked_except_for_one?(a1,side,s)
%% case 7: there is an aircraft that is assigned 'side' and is not blocked
%%         in the opposite side, and one aircraft is in maz
Lem2_case7(s:states,side:Side):bool =
     LET a1 = first(maz(side,s)) IN
     length(maz(side,s))=1 AND
     ac_ready_to_approach?(side,s)
       IMPLIES
          blocked_except_for_one?(a1,side,s)
%% Lemma 2: combination of seven cases, and invariants 3 and 4.
Lem2(s:states):bool =
     FORALL (side:Side):
        Inv3(s) AND Inv4(s) AND Lem2_case1(s,side) AND
        Lem2_case2(s,side) AND Lem2_case3(s,side) AND Lem2_case4(s,side) AND
        Lem2_case5(s,side) AND Lem2_case6(s,side) AND Lem2_case7(s,side)
```

We use new auxiliary predicates blocked_except_for_one? and ac_ready_to_approach?. We do not have a space to present a definition, but it appears in [5].

Lemma 5. (The key lemma) *For any reachable state of the abstract model, the lemma introduced in this subsection holds.*

The complete proof appears in [5]. Due to the substantial amount of the case analyses, the length of the proof becomes as long as ten pages. We followed a way analogous to the proof of Lemma 4. As opposed to Lemma 4, however, we have to be careful about matching the blocking aircraft as stated before.
Now we prove Property 2 using Lemma 5.

Theorem 6. (Property 2) *For any reachable state of the abstract model and side σ, the number of aircraft in one initiation area is at most two.*

Proof. Suppose there are more than two aircraft in one initiation area. For any possible position of these aircraft, it violates either Property 3, 4, 5, or 6, or Case 1 or 3 of Lemma 5. This is a contradiction.

5 Conclusions and Future Work

In this paper, we first reconstructed the mathematical model of an aircraft landing protocol presented in [2], using the I/O automata framework. Though the protocol is complex, the IOA code we gave has a manageable form. Using the reconstructed model, we verified some safe separation properties of aircraft in the Self Controlled Area. All proofs of the properties have been rigorously checked using PVS. We found that using a mechanical prover is very helpful in managing a large proof for a moderately complex system such as ours.

The model in the paper is a discrete model in that the airspace and every movement of the aircraft are discretized. Using this model, we can verify the safe separation of aircraft in terms of the bound on the number of aircraft in a specific discretized area. However, to examine properties that involve more realistic dynamics of aircraft, such as the spacing between aircraft, we need a more precise modeling of the aircraft kinematics and the geometry of the airport. A continuous model, such as the one presented in [7], is suitable to deal with such properties. We are constructing a continuous model that more realistically reflects the dynamics of the aircraft than the model in [7]. We will also explore if the results in this work can carry over to the new model using a refinement.

References

1. T.Abbott, Jones, K., Consiglio, M., Williams, D., Adams, C.: Small Aircraft Transportation System, High Volume Operation concept: Normal operations. Technical Report NASA/TM-2004-213022, NASA Langley Research Center, NASA LaRC,Hampton VA 23681-2199, USA (2004)
2. Dowek, G., Muñoz, C., Carreño, V.: Abstract model of the SATS concept of operations: Initial results and recommendations. Technical Report NASA/TM-2004-213006, NASA Langley Research Center, NASA LaRC,Hampton VA 23681-2199, USA (2004)
3. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc. (1996)

4. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In Kapur, D., ed.: 11th International Conference on Automated Deduction (CADE). Volume 607 of Lecture Notes in Computer Science., Saratoga, NY (1992) 748 – 752
5. Umeno, S.: Proving safety properties of an aircraft landing protocol using timed and untimed I/O automata: a case study. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA (2006)
6. Garland, S.: TIOA User Guide and Reference Manual. (2005)
7. Muñoz, C., Dowek, G.: Hybrid verification of an air traffic operational concept. In: Proceedings of IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation, Columbia, Maryland (2005)