Proving safety properties of an aircraft landing protocol using timed and untimed I/O automata: a case study

by

Shinya Umeno

B.S., Information Science (2004)

Tokyo Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

February 2007

Signature of Author . . . .
Department of Electrical Engineering and Computer Science
October 25th, 2006

Certified by . . . .
Nancy A. Lynch
Professor of Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Srinivas Devadas
Chairman, Department Committee on Graduate Students

Proving safety properties of an aircraft landing protocol using timed and untimed I/O automata: a case study

by

Shinya Umeno

Submitted to the Department of Electrical Engineering and Computer Science
on October 25th, 2006, in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis presents an assertional-style verification of the aircraft landing protocol of NASA's SATS (*Small Aircraft Transportation System*) concept of operation [16] using the timed and untimed I/O automata frameworks. We construct two mathematical models of the landing protocol using the above stated frameworks. First, we study a discrete model of the protocol, in which the airspace of the airport and every movement of the aircraft are all discretized. The model is constructed by reconstructing a mathematical model presented in [2] using the untimed I/O automata framework. Using this model, we verify the safe separation of aircraft in terms of the bounds on the numbers of aircraft in specific discretized areas. In addition, we translate this I/O automaton model into a corresponding PVS specification, and conduct a machine verification of the proof using the PVS theorem prover.

Second, we construct a continuous model of the protocol by extending the discrete model using the timed I/O automata framework [6]. A refinement technique has been developed to reason about the external behavior between two systems. We present a new refinement proof technique, *a weak refinement using a step invariant.* Using this new refinement, we carry over the verification results for the discrete model to the new model, and thus guarantee that the safe separation of aircraft verified for the discrete model also holds for the new model. We also prove properties specific to the new model, such as a lower bound on the spacing of aircraft in a specific area of the airport, using an invariant-proof technique.

Thesis Supervisor: Nancy A. Lynch
Title: Professor

Acknowledgments

First of all, I would like to thank my thesis supervisor, Professor Nancy Lynch. Without her continuous support, I could not have written this thesis. Through interactions with her for this thesis, I learned many things, such as how to organize chapters and paragraphs, how to make rigorous mathematical arguments, and lots and lots.

I also would like to thank G. Dowek and C. Muñoz and V. Carreño, the authors of the paper [2, 12], which inspires my research presented in this thesis. Indeed, what we call the "discrete model" in this thesis is originally constructed by them in [2]. In addition, the construction of the "continuous model" in this thesis follows a scheme analogous to the one used for the "hybrid model" presented in [12].

I am grateful to my office mates and the members of Theory of Distributed Algorithms. In particular, I thank Sayan Mitra for having both technical and non-technical discussions with me. I also thank Calvin Newport. As a student who entered MIT in the same year as I, I often talked with him about the progress of each other's master's theses and other things that came up for us at the same time (how to write a proposal for a thesis, how to apply a teaching assistantship, and so forth).

I also thank Professor Osamu Watanabe and Professor Roger Pulvers in Tokyo Institute of Technology. Prof. Watanabe was the supervisor for my bachelor's thesis, and I leaned basics of proof techniques in computer science literature from him. Prof. Pulvers taught me English in several courses, and I also discussed with him about how to write a statement of objectives for graduate school applications. Without these two people, I would not have entered MIT.

A special thanks goes to my friends, especially, F. L., N. I., K. K., A.T., K. S., P. S., H. S.-B., N. Y., T. I., R. Y., S. R., H. O., T. A., S. T., M. H., N. N., H. I., M. S., Y. K., Y. S., S. K., K. E., A. E., Y. P., S. P., M. T., A. P., S.C., K. H., K. S., M. K., J. W., S. P. These people helped me to temporarily get relief from pressure of writing this thesis. Thank you, all!

Last, but not least, I thank my parents and my wife Chieko. Chieko always supported me mentally, and also by delicious dinners. With her Japanese-style dinners, I did not need to miss our country, Japan, too much, and could concentrate on writing the thesis.

Contents

1 Introduction **9**

2 Discrete model of SATS **12**

2.1 Introduction 12

2.2 Discrete model 12

2.2.1 Logical zones 13

2.2.2 Leader aircraft 14

2.2.3 Paths of aircraft 15

2.3 Formal code for the discrete model 17

2.3.1 Types and auxiliary functions 17

2.3.2 Transition signatures and state variables of the discrete model 25

2.3.3 Mathematical Notations 27

2.3.4 Transitions of the discrete model 28

3 Properties of the discrete model and their proof **39**

3.1 Introduction 39

3.2 Main Properties 39

3.3 Auxiliary invariants 43

3.3.1 Uniqueness of aircraft and the correspondence between the logical zones and the landing sequence 43

3.3.2 Key Invariants 54

3.4 Proving the main properties, Part 1: Properties that can be proved by straightforward induction 61

3.5 Blocking of aircraft 63

3.6 Proving the main properties, Part 2: strengthening Property 6 73

3.7 Proving the main properties, Part 3: the key lemma, and the remaining properties 76

3.7.1 Intuition behind the lemma 77

3.7.2 The key lemma 85

3.7.3 Proof of Lemma 3.26 90

3.7.4 Proof of Property 2 112

3.8 Proof using the PVS theorem prover 113

3.8.1 Steps we have taken to use a theorem-prover 113

3.8.2 Theorem-proving process in PVS 114

3.8.3 Our experience obtained by this case study and possible improvement for mechanical proof processes using theorem-provers 116

4 Timed I/O automata framework 121

4.1 Timed I/O automata 121

4.1.1 Functions 121

4.1.2 Time 121

4.1.3 Static and dynamic types 122

4.1.4 Trajectories 122

4.1.5 Hybrid Sequences 124

4.1.6 Timed Automata 126

4.1.7 Timed I/O Automata 129

4.2 Simulation relation and refinement proof techniques for timed I/O automata . . 130

4.2.1 Forward simulation and refinement for timed I/O automata 130

4.2.2 Weak refinement using step invariants 135

4.2.3 Close correspondence between executions of two automata implied by a forward simulation 138

4.2.4 Invariants deduced from a forward simulation 139

5 Continuous model of SATS and its safe separation property 140

5.1 Introduction 140

5.2 Hybrid model of [12] 140

5.3 Our New Continuous model 143

5.3.1 Formal Specification for *ContSATS* 145

5.3.2 State of *ContSATS* 148

5.3.3 Transitions inherited from the discrete model to *ContSATS* 148

5.3.4 New Internal Transitions in *ContSATS* 152

5.3.5 Trajectories of *ContSATS* 153

5.3.6 Assumptions for *ContSATS* 154

5.4 Basic invariants of $ContSATS$ 156
5.5 Carrying over results from the discrete model using a refinement 159
5.5.1 Refinement mapping 160
5.5.2 Auxiliary invariants needed for refinement proof 162
5.5.3 Refinement proof 173
5.6 Spacing properties of aircraft in $ContSATS$ 177
5.6.1 S_{T}: the spacing of aircraft in the approach area 177
5.6.2 $\mathsf{S}_{(\mathsf{H3,B})}$ and $\mathsf{S}_{(\mathsf{L,B})}$: the spacing between aircraft in the initiation zones (holding3 and lez) and aircraft in the base zones 180
5.6.3 $\mathsf{S}_{(\mathsf{M,H3})}$: the spacing of aircraft in a missed approach path, part 1 181
5.6.4 S'_{M} and $\mathsf{S}_{(\mathsf{T,M})}$: the spacing of aircraft in a missed approach path, part 2 . 182
5.6.5 $\mathsf{S}_{(\mathsf{M,H3})}$: the spacing of aircraft in a missed approach path, part 3 189

6 Conclusions and Future Work **191**

A PVS code **197**

List of Figures

2.1 13 logical zones in SATS 13

2.2 Logical zones divided into four areas 14

2.3 Paths of aircraft 16

2.4 Paths of aircraft that have missed the approach 17

2.5 Vocablary Part 1 of 2 19

2.6 Vocablary Part 2 of 2 20

2.7 Auxiliary functions defined in PVS: Part 1 of 2 21

2.8 Auxiliary functions defined in PVS: Part 2 of 2 22

2.9 The virtual number of aircraft 24

2.10 Code for the discrete model [Part 1 of 2. Signature and States] 26

2.11 Code for the discrete model [Part 2 of 2. Transitions] 29

2.12 Missed Approach Initiation 35

2.13 The double transition of aircraft in the third case of LowestAvailableAltitude(right) 37

3.1 An example of a state in which the virtual number of aircraft on the right side is more than two 41

3.2 The left side is b-blocked, where b is the first aircraft of `lez(right)` 65

3.3 c is not ready to go to the opposite side until b initiates the approach 66

3.4 Transitions between cases creating a circle 81

3.5 Transitions between seven cases in the lemma 86

3.6 A typical node in a PVS proof tree 115

3.7 Use of labels 115

3.8 Use of labels (when another formula is inserted to the assumptions) 115

5.1 Sub-area that exhibits a continuous behavior in the hybrid model of [12] 143

5.2 The continuous model $ContSATS$ 144

5.3 Vocabulary for $ContSATS$ 146

5.4 Formal code for $ContSATS$, Part 1 of 3 148

5.5 Formal code for $ContSATS$, Part 2 of 3 149

5.6 Formal code for $ContSATS$, Part 3 of 3 150

5.7 Lower bounds on the spacing of aircraft in two consecutive zones in $ContSATS$. 178

5.8 Two aircraft are respectively in the different sides of the base zones. 179

5.9 Two lemmas and one theorem that state the minimum spacing of aircraft 184

Chapter 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 safety verification of one of such safety critical systems: an aircraft landing protocol that is part of NASA's SATS (*Small Aircraft Transportation System*) concept of operation [16].

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, a centralized air traffic management system is automated as a module called the Airport Management Module, and gives supplemental information to pilots 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 protocols of SATS is presented in [2]. The model is a finite-state transition system obtained from a mathematical abstraction of the real system. In [2], some properties of the discrete model that represent the safety 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 in a particular portion of the airport (for example, no more than four aircraft are in the entire Self Controlled Area; or at most one aircraft is at a certain part of the airspace in the airport).

As mentioned above, in the discrete model, the geographical and kinematic information of the real system is discretized. This model can be used to prove the safe separation of aircraft in terms of the bounds on the numbers of aircraft in specific discretized areas. However, to examine properties that involve more realistic dynamics of aircraft, such as the spacing between aircraft, we need a more detailed modeling of the aircraft kinematics and the geometry of the airport. To treat such properties, a *hybrid model* of the protocol is presented in [12], in which the movement of the aircraft in some specific air space of the airport is modeled as a continuous behavior. Using this model, a lower bound on the spacing between two aircraft in this specific area (where aircraft moves continuously) of the hybrid model is claimed and exhaustively checked using a symbolic model-checking technique. A limitation of this hybrid model is that it captures only the dynamic behavior of aircraft in some specific area of the airport.

In this thesis, we construct two mathematical models of the protocol using the timed and untimed I/O automata framework [10, 6], and conduct a safety verification of them using assertional-style proof techniques.

First, we present a discrete model that is constructed by reconstructing the model presented in [2] using the untimed I/O automata framework. 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 constructing a discrete model of the landing protocol in which the next possible step that the model can take is nondeterministically defined. Using our reconstructed model, we carry out a proof of these properties using inductive proof techniques that have been used in the computer science literature, as opposed to an exhaustive state exploration used in [2]. We also translate this I/O automaton model into a corresponding PVS specification, and conduct a machine verification of the proof for the properties of the discrete model using the PVS theorem prover.[1] Thus, 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 second model we present in this thesis is a *continuous model* that more realistically reflects the dynamics of aircraft movement in a real system than the discrete model we study first or the hybrid model presented in [12]. In contrast to the above mentioned models, our continuous model captures the continuous movement of aircraft in the entire Self Controlled Area. As discussed in Chapter 5, some problems that arises from the discretization of the aircraft dynamics are resolved in this model. Using this model, we first formally verify the safe separation properties proved for the discrete model also hold in our new continuous model. We use the *refinement* technique to carry over the results for the discrete model to the continuous model. In doing so, we introduce a new refinement definition, *a weak refinement using a step invariant*, that makes use of invariants of automata in a refinement proof. Next, we verify several

[1] Complete I/O automata and PVS specification codes, and PVS proof scripts are available at `http://theory.csail.mit.edu/∼umeno/`

minimum spacing properties including those verified in [12]. These properties are proved using an invariant-proof technique.

This thesis is organized as follows. Chapter 2 is devoted to presenting the discrete model of the protocol. We start by an overview of the model. Next we present formal I/O automata code of the model, and then closely examine auxiliary functions used for the automata, and state variables and transitions of the model. In Chapter 3, we introduce the main properties of the discrete model we want to prove. These properties mainly state upper bounds on the number of aircraft in particular areas of the airport. Next, we prove some auxiliary invariants that we consider to be the most basic properties of the model. Using these invariants, we prove the main properties. We also introduce an important notion of "blocking" of aircraft. Using this notion, we strengthen some of the main properties in order for them to be proved inductively. We also discuss some issues concerning proofs in PVS. In Chapter 5, we present the continuous model of the protocol, and verify the safe separation properties of it. We introduce the formal description of the model, and examine it by comparing it with the discrete model in Chapter 2, and the hybrid model of [12]. Next we carry over the results for the discrete model presented in Chapter 2 to the new model by using a refinement technique. Finally, we verity several spacing properties that represents a finer geographical and dynamical claim than can be expressed using the discrete model.

Finally in Chapter 6, we summarize the results of this thesis, and give an evaluation. We also discuss future work in this chapter.

The code used for the PVS proof is attached as Appendix A

Chapter 2

Discrete model of SATS

2.1 Introduction

In this chapter, we present an I/O automaton model for SATS, based on the discrete model presented in [2]. Whereas the model in [2] describes both aircraft landings and departures, in this thesis we restrict our attention to landings. The main reason for this restriction is that the interesting procedures of SATS are performed in the landing part, and the properties that are not trivial, that is, those we cannot prove by just examining the preconditions of the transitions, are defined for the landing protocol.[1]

In this discrete model, the space of the airport used for landings of aircraft is is divided into several zones. These zones are represented as state components of the automaton, and the model can be used to check if the desirable upper bound on the number of aircraft in a specific zone is satisfied. However, to verify the properties that involve more realistic dynamics of aircraft, such as a property of the spacing between aircraft, we need a more detailed model of the aircraft kinematics and the geometry of the airport. A continuous model, such as the hybrid model presented in [12], or the continuous model we will present in Chapter 5, is suitable to deal with such properties.

We start with an informal explanation of the discrete model in Section 2.2, and present the formal definition of the model as an I/O automaton in Section 2.3. We also closely examine each transition of the model in Section 2.3.4.

2.2 Discrete model

In this section, we will present a high-level overview of the model used in this chapter.

[1] Indeed, for the departure part, the properties examined in [2] can be proved immediately from how the preconditions of the transitions for the departure procedures are defined.

2.2.1 Logical zones

The space of the airport used for landings is logically divided into 13 zones (see Figure 2.1). 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, and share the same name but have different attribute values designating their side, for instance, `holding3(right)` and `holding3(left)`.[2]

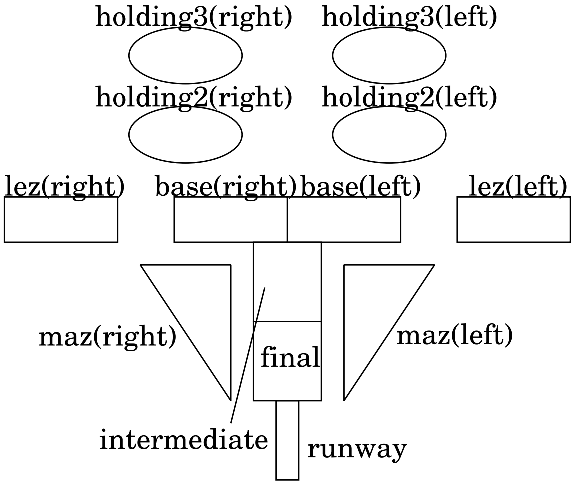


Figure 2.1: 13 logical zones in SATS

For the sake of easier understanding 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: the left initiation area, the right initiation area, the approach area, and the runway (see Figure 2.2). The *left initiation area* consists of `holding3(left)` (*holding fix at 3000 feet*) and `holding2(left)`(*holding fix at 2000 feet*), 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 counterpart of the left initiation area, consisting of `holding3(right)`, `holding2(right)`, `lez(right)`, and `maz(right)`. The *approach area* consists of `base(right)`,

[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.)

`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 the Self Controlled Area that we mentioned in Chapter 1. In this thesis, we focus on the safety conditions in the operation area.

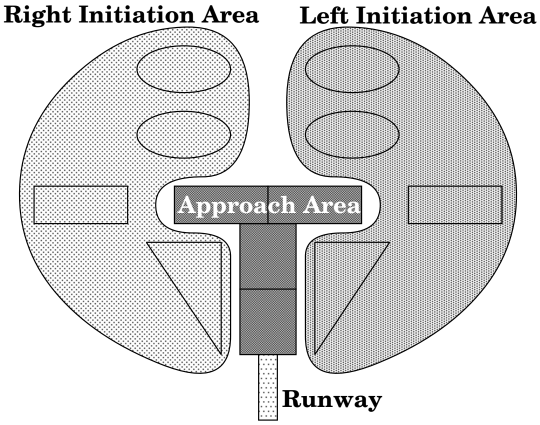


Figure 2.2: Logical zones divided into four areas

2.2.2 Leader aircraft

When an aircraft enters the system, the system assigns its *leader* aircraft, or the aircraft it has to follow. This leader relation 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 within 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 present the initiation conditions formally in Section 2.3.4.

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 from the approach initiation. We will closely look at the case when an aircraft misses the approach in Section 2.3.4.

2.2.3 Paths of aircraft

In the SATS concept, a centralized ground-based automation system, called the Airport Management Module, is used in the protocol to give aircraft information needed to decide whether or not each aircraft can proceed to the next zone at that moment. This module would typically be located at the airport and would calculate information for aircraft from aircraft performance, aircraft position information, and a set of predetermined operation rules of the concept. It is worth to note here that the module does not decide the exact order of the progression of the aircraft in the Self Controlled Area. Each aircraft uses the information given by the module, and individually decides whether or not it can proceed to the next zone.

In the discrete model, the information given by the module is implicitly used within the preconditions of the transitions that represents when specific movements of aircraft are allowed. When several transitions are allowed to perform at the same state of the model, the next possible transition of the system is *nondeterministically* determined. Details of the movement of an aircraft in the logical zones – such as when a specific procedure (movement) of the aircraft is allowed, or under what conditions it can initiate the approach to the ground – are presented in Section 2.3.4. Here we present a high level picture of how an aircraft moves from the entry to the logical zones, initiates the approach to the ground, and lands on the runway. We refers to the corresponding transitions' names in parentheses when explaining the movements of aircraft in the following.

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, but if the conditions are not satisfied, 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: for instance, an aircraft that initiates the approach from `holding2(right)` goes to `base(right)`. Once an aircraft enters `base`, it merges into `intermediate` (`Merging`), then proceeds to `final` (`FinalSegment`) and lands on `runway` (`Landing`). This progression of the movement of aircraft is depicted in Figure 2.3.

An aircraft may miss the approach to the ground at the `final` zone. In such a case, it has to execute the landing operation again from the initiation of the approach. Thus, it once again goes back to a zone where it can initiate the approach again, and make the next try to land.

The problem that arises here is how to decide which side of the area it should go back to in

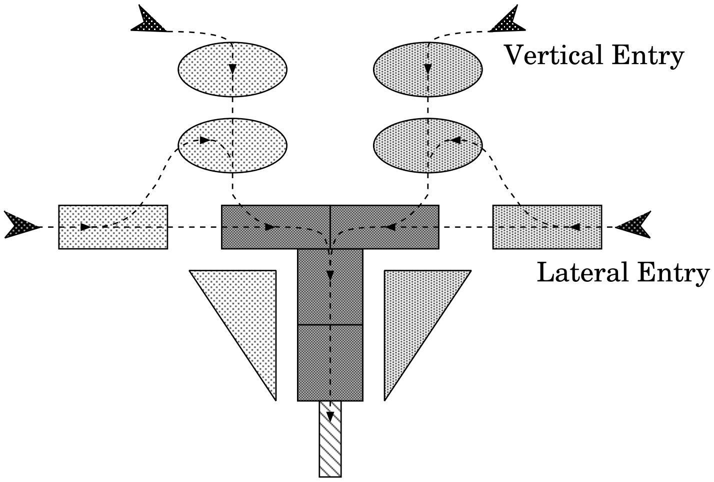


Figure 2.3: Paths of aircraft

order to initiate the approach (remember that an aircraft can initiate the approach either from the left side or the right side of the logical zones). The system solves the problem by assigning the side of an approach initiation area to which a particular aircraft has to go in case it misses the approach. This assignment of the side, called the "*missed approach holding fix (mahf)*" is given by a centralized automated system called Airport Management Module, to an aircraft when it enters the system, based on a global system variable `nextmahf`. The variable `nextmahf` is of type `Side`, an enumeration of `left` and `right`, and is used by the system 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 *alternating assignment* of the left side and the right side to the aircraft in the landing sequence. Note that despite the use of the centralized module, as we will discuss in Section 2.3.4, the protocol actually exhibits nondeterministic behavior: Though the centralized module gives the information to aircraft so that each aircraft can individually decide the moment when that aircraft will proceed to the next zone, the module does not determine the exact order in which aircraft proceed in the system.

An aircraft that has missed the approach is treated in a way analogous to a newly entering aircraft in terms of the operation in the landing sequence: it is first taken from the head of the landing sequence, then the mahf assignment of it is *reassigned* based on `nextmahf`, and it is again added to the end of the landing sequence. The variable `nextmahf` is flipped in this case as well, so that the alternating assignment will be preserved even in case some aircraft miss the approach.

In the logical zones, a missed aircraft, with the reassignment as stated above, first goes to `maz` of the side that it is assigned as its mahf (`MissedApproach`), and from there it goes back to either `holding2` or `holding3` of the same side as the side of `maz` where it leaves (`LowestAvailableAltitude`). Whether it goes to `holding2` or `holding3` is determined by the situation at the time it leaves `maz`. Details of the transition for a case of a missed approach are discussed in Section 2.3.4. These paths for aircraft that have missed the approach are shown in Figure 2.4.

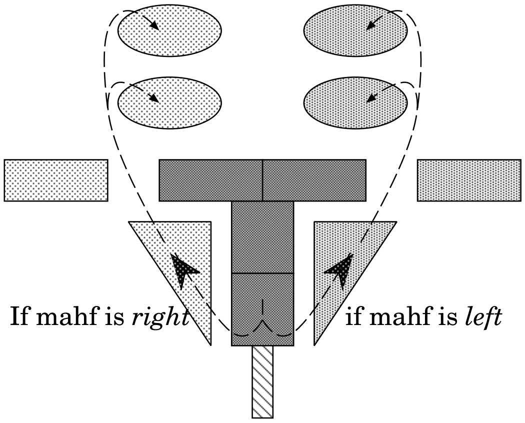


Figure 2.4: Paths of aircraft that have missed the approach

2.3 Formal code for the discrete model

In this section, we present formal code for the landing protocol of SATS. The language we used is a subset of the Timed I/O Automata formal language (TIOA) [3], which is intended to model systems that involve time-dependent behavior or continuous dynamics, using the notion of trajectories. Since we have just discrete transitions for the model treated in this chapter, we used a subset of the language without trajectories.

2.3.1 Types and auxiliary functions

The automaton code imports the *vocabulary* statement (Figures 2.5 and 2.6). The vocabulary is used to define or declare types and auxiliary functions. For example, an aircraft type, `Aircraft`, is defined, and auxiliary functions such as `leader` or `virtual` are declared, in the vocabulary. Recursive types (such as `queue`) or recursive functions (such as `leader`) cannot currently be defined in TIOA. Thus we just declare recursive types and the types of recursive functions, and other functions that use those, in TIOA code, and define them in the PVS specification language. Then, we import the definitions given in this PVS code in a translation process from

IOA code to PVS code. Figures 2.7 and 2.8 show the PVS file that defines recursive functions and types, and other auxiliary functions that use those, for the discrete model.

For the auxiliary predicates, we use the "_qn" suffix for their name, whereas in PVS, we use the "?" suffix. This is because "?" is a reserved word in the TIOA language. In a translation process from TIOA code to PVS code, every "_qn" suffix of function names is replaced by the "?" suffix.

Here we explain types and auxiliary functions defined or declared in the vocabulary.

Types (lines 2 - 16)

queue, Zone (lines 3 and 4): Each zone is defined as a queue of type Aircraft. The length of a queue represents the number of aircraft in the zone represented by that queue.

ID (line 6): The type ID is used to express the ID of aircraft.

Side (line 7): The type Side is defined as an enumeration of `right` and `left`.

Aircraft (lines 8 - 11): An aircraft is defined as a tuple that has two attributes: one is the mahf assignment, `mahf` of type `Side`, set by the system when it enters the system; and the other is a unique ID, `id`, which is encoded as a natural number in the discrete model. Note that there is no attribute for the leader, since the leader relation is defined using an explicit queue of aircraft, called the landing sequence, in the model as explained in Section 2.3.2[3]

z_name (line 13): The type for a name of the logical zones is defined as z_name, an enumeration of the names of all thirteen zones. These names are used in the following zone_map.

zone_map (line 16): This type is used as the type for P`zones`, a state variable of the automaton. A function of this type maps a name of the logical zones (z_name) to an actual queue representing that zone.

Operators (lines 18 - 100)

The section of the code in lines 18 - 100 defines auxiliary functions (operators).

__[__], assign (lines 19 and 20): These are auxiliary functions for `zone_map`. The operator `__[__]` is used to access the queue from a given zone mapping and a given zone name. For example, `zones[holding3L]` denotes the queue for `holding3L` that `zones` maps to. Operator `assign` is used to re-assign the value of a function of type `zone_map` for a particular argument. For example, `assign(zones, holding3L, add(zones(holding3L),a))` represents a new zone

[3] We could have expressed the leader relation as an attribute of an aircraft (it forms a structure similar to a linked list), but we used a queue, since queues are defined using a PVS primitive list structure, and all auxiliary functions and libraries for that structure can be used in the machine proof.

```
vocabulary SatsVocab
  types
    queue,       %% defined in PVS as a queue of aircraft.
    Zone,        %% defined in PVS as the above queue of aircraft. It has a different name
                 %% just to differenciate logical zones and the landing sequence.
    ID,          %% defined as positive natural number type in PVS
    Side enumeration [right,left],
    Aircraft tuple [
      mahf: Side, % Missed approach holding fix assignment.
      id  : ID    % ID of the aircraft
      ],
  %% name of zones
    z_name enumeration [holding3L, holding3R, holding2L, holding2R, lezL, lezR,
                        mazL, mazR, baseL, baseR, intermediate, final, runway]

    zone_map,    %% a type for mappings from a name of a zone to an actual queue of aircraft.

  operators
    __[__]: zone_map, z_name -> Zone,           %% accessor for zone_map
    assign: zone_map, z_name, Zone -> zone_map, %% for assigning zone_map


    % a function that maps any zone name to an empty queue.
    initialZones: -> zone_map,


  %%%%%%%%%% basic queue functions %%%%%%%%%%%%%%%%%%%%%%%%%
    empty: -> Zone,
    empty: -> queue,
    empty_qn: queue -> Bool,
    empty_qn: Zone -> Bool,
    length: queue -> Nat,
    length: Zone -> Nat,
    add: queue, Aircraft -> queue,
    add: Zone, Aircraft -> Zone,
    first: queue-> Aircraft,
    first: Zone -> Aircraft,
    rest: queue -> queue,
    rest: Zone -> Zone,
    in_queue_qn:Aircraft, queue-> Bool,
    in_queue_qn:Aircraft, Zone-> Bool,

  %%%%%%%% Side -> z_name %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% maps zone(side) to a zone name.
    %% e.g., holding3(right) = holding3R
    holding3:Side->z_name,  holding2:Side->z_name,
    lez:Side->z_name,  maz:Side->z_name,  base:Side->z_name

  %%%%%%%% Opposite side %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    opposite:Side->Side,
```

Figure 2.5: Vocablary Part 1 of 2

```
%%%%%% recursive functions defined in PVS %%%%%%%%%%%%%%%%%%%%%%%%%
  %% leader aircraft
  leader: Aircraft, queue -> Aircraft,

  %% a precedes b in seq q
  precedes_qn: Aircraft, Aircraft, gueue ->Bool,

  %% Number of aircraft in a zone that is assigned to one side
  assigned: Zone, Side -> Nat,

%%%%% on_zone?, assigned?, etc %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% Is any aircraft in a particualr zone assigned to a particular side as its mahf?
  assigned_qn:Zone,Side-> Bool,

  %% Is a particular aircraft in a particular zone?
  on_zone_qn:Zone,Aircraft->Bool,

  %% Is a particular aircraft in a particular side?
  on_qn:Side,Aircraft->Bool,

  %% Is a paricular aircraft on the approach?
  on_approach_qn:zone_map,Aircraft-> Bool,

  %% Is aircraft a in the operation area?
  on_zones_qn:zone_map,Aircraft-> Bool,

  %% The number of aircraft on the approach assigned to a particular side as their mahf.
  assigned_approach:zone_map,Side->Nat,

  %% Is any aircraft on the approach assigned to a particular side as its mahf?
  on_approach_qn:zone_map,Side-> Bool,

%%%%%% actual, virtual %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% the acutal number of aircraft at the initiation area of a particular side
  actual:zone_map,Side->Nat,

  %% the virtual number of aircraft at the initiation area of a particular side
  virtual:zone_map,Side-> Nat,

%%%%%% arrival_op, assigned2fix  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% the number of aircraft assigned to a particular side as their mahf
  assigned2fix:zone_map,Side -> Nat,

  %% the number of aircraft in the operation area.
  arrival\_op:zone_map -> Nat,

%%%%%%%%%% move %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % predicated parameters
  % an aircraft moves from z_from to z_to
  move:z_name, z_name, zone_map -> zone_map,
```

Figure 2.6: Vocablary Part 2 of 2

```
%% This file defines recurcive functions and types,
%% and other auxiliary functions that use those.

    %% accessing a zone by side
    holding3(side:Side):z_name = IF side = left THEN holding3L ELSE holding3R ENDIF
    holding2(side:Side):z_name = IF side = left THEN holding2L ELSE holding2R ENDIF
    lez(side:Side):z_name      = IF side = left THEN lezL      ELSE lezR      ENDIF
    maz(side:Side):z_name      = IF side = left THEN mazL      ELSE mazR      ENDIF
    base(side:Side):z_name     = IF side = left THEN baseL     ELSE baseR     ENDIF

    %% side of a zone
    side(z:z_name|
             z=holding3L OR z=holding3R OR z=holding2L OR z=holding2R OR z=lezL OR z=lezR OR
             z=mazL OR z=mazR OR z=baseL OR z=baseR): Side =
         IF z=holding3L OR z=holding2L OR z=lezL OR z=mazL OR z=baseL THEN left ELSE right ENDIF

 %% Opposite side
 opposite(side:Side) : Side =
   IF side = right THEN left
   ELSE right
   ENDIF

 %% leader of an aircraft
 leader(a: Aircraft, q:queue| a /= first(q)): RECURSIVE Aircraft =
    IF a = first(rest(q)) THEN first(q)
                          ELSE leader(a,rest(q))
    ENDIF
    MEASURE length(q)

 %% Is b the leader aircraft of a ?
 leader?(a,b:Aircraft, q:queue): bool =
   b = leader(a,q)

 %% Does aircraft 'a' exist in the queue?
 in_queue?(a:Aircraft, q:queue): bool = member(a,q)

  %% Is 'a' precedes 'b' in the landing sequence?
 precedes?(a,b:Aircraft, q:queue) : RECURSIVE bool =
   IF empty?(q) OR first(q) = b THEN false
   ELSIF first(q)=a THEN in_queue?(b, rest(q))
   ELSE  precedes?(a,b,rest(q))
   ENDIF
   MEASURE length(q)

 %% Number of aircraft in a zone to assigned to one side
 assigned(z:Zone,side:Side): RECURSIVE nat =
   IF empty?(z) THEN 0
   ElSIF mahf(first(z)) = side THEN 1+assigned(rest(z),side)
   ELSE assigned(rest(z),side)
   ENDIF
   MEASURE length(z)

 %% Is any aircraft in zone z assigned to the mahf side ?
 assigned?(z:Zone,side:Side): bool =
   assigned(z,side) /= 0

 %% Is an aircraft in zone z ?
 on_zone?(z:Zone,a:Aircraft) :bool = in_queue?(a,z)
```

Figure 2.7: Auxiliary functions defined in PVS: Part 1 of 2

```
%% Is aircraft a on this side?
on?(side:Side, a:Aircraft, z:zone_map):bool =
              on_zone?(z(holding3(side)),a) OR on_zone?(z(holding2(side)),a) OR
              on_zone?(z(lez(side)),a) OR  on_zone?(z(maz(side)),a)

%% Is aircraft a on the approach ?
on_approach?(z:zone_map,a:Aircraft): bool = on_zone?(z(baseR),a) or on_zone?(z(baseL),a)
                                  or on_zone?(z(intermediate),a) or on_zone?(z(final),a)

%% Is aircraft a on any zone (excluding runway)?
on_zones?(zones:zone_map,a:Aircraft): bool =
  EXISTS (z:z_name) : on_zone?(zones(z),a) AND z/=runway

%% # of aircrafts with this mahf on approach
assigned_approach(z:zone_map,side:Side): nat =
  assigned(z(baseR),side) + assigned(z(baseL),side) +
  assigned(z(intermediate),side) + assigned(z(final),side)

%% Is any aircraft on the approach assigned to the mahf side ?
on_approach?(z:zone_map,side:Side): bool =
  assigned?(z(baseR),side) or assigned?(z(baseL),side) or
  assigned?(z(intermediate),side) or assigned?(z(final),side)

%% Acutal number of aircraft at one side (excluding the approach)
actual(z:zone_map,side:Side):nat =
  length(z(holding3(side)))+length(z(holding2(side)))+length(z(lez(side)))+
  length(z(maz(side)))

%% Virtual number of aircraft at one fix
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)

%% Number of aircraft assigned to a fix
assigned2fix(z:zone_map,side:Side):nat =
  assigned(z(holding3R),side) + assigned(z(holding3L),side) +
  assigned(z(holding2R),side) + assigned(z(holding2L),side) +
  assigned(z(lezR),side) +      assigned(z(lezL),side) +
  assigned(z(baseR),side) +     assigned(z(baseL),side) +
  assigned(z(intermediate),side) + assigned(z(final),side) +
  assigned(z(mazR),side) +      assigned(z(mazL),side)

%% Total number of simultaneous arrival operations
arrival_op(z:zone_map):nat =
  actual(z,right) + actual(z,left) +
  length(z(baseR)) + length(z(baseL)) + length(z(intermediate)) + length(z(final))

 % define movement of aircrafts
 % an aircraft moves from z_from to z_to
 move(z_from, z_to: z_name, zones:zone_map| z_from /= z_to AND NOT empty?(z_from)):
       zone_map =  zones WITH [(z_from) := rest(zones(z_from)),
               (z_to)   := add(zones(z_to), first(zones(z_from)))]
```

Figure 2.8: Auxiliary functions defined in PVS: Part 2 of 2

mapping that is the same as the original mapping `zones` except for that the queue for `holding3L` is updated to `add(zones(holding3L),a)`.

initialZones (line 24): This function represents the empty zones.

Basic queue operations (lines 27 - 41): These functions represent basic queue operations.

Operators to access a zone name by a side (lines 46 and 47): The auxiliary functions defined in this part maps a side to a zone name. For example, `holding3(right) = holding3R`.

opposite (line 50): The function `opposite` maps a given side to the opposite side of it. For example, `opposite(right) = left`.

leader (line 53): We define the leader of an aircraft a in a given sequence as the aircraft just in front of a in the sequence. Function `leader`, which represents the leader of a given aircraft in a given sequence, is formally defined in PVS in Figure 2.7, lines 25 - 29. The first IF clause in the formal code works as a "guard" that guarantees that the value of `leader` is well-defined even if the given aircraft `a` is not in the given queue `q`. If `a` is not in `q`, `leader` returns `a` by default.

precedes_qn (line 56): To state some auxiliary lemmas in Chapter 3, we need the notion that aircraft a *precedes* b in the given sequence. A mathematical definition of this relation is given in Section 2.3.3. This *precedes* relation is defined in PVS in Figure 2.7, lines 39 - 44.

assigned (line 59): This function calculates the number of aircraft a in `zone` such that $a.\mathsf{mahf} =$ `side`. This function is defined in PVS in Figure 2.7, lines 47 - 52.

assigned_qn (line 63): The predicate `assigned_qn`(z, σ) checks if the value of `assigned`(z, σ) is not zero.

on_zone_qn (line 63): The predicate `on_zone_qn`(z, a) checks if aircraft a is in zone z. This predicate is defined using *in_queue_qn* (Figure 2.7, line 59).

on_qn (line 69): The auxiliary predicate `on_qn(side,a)` checks if aircraft `a` is in the initiation area of `side` in state `s`, that is, if `a` is either in `holding3(side)`, `holding2(side)`, `lez(side)`, or `maz(side)`. Note that `base(side)` zone is not included, since, even though `base` has a right/left symmetric structure, it is part of the approach area, but not the initiation area. This predicate is defined in PVS in Figure 2.8, lines 61 - 63.

on_approach_qn for aircraft (line 72): The auxiliary predicate `on_approach_qn` checks if aircraft `a` is in the approach area, that is, if `a` is either in `base(right)`, `base(left)`, `intermediate`, or `final`. This predicate is defined in PVS in Figure 2.8, lines 66 and 67.

on_zones_qn (line 75): The auxiliary predicate `on_zones_qn` checks if aircraft `a` is in the operation area, that is, if `a` is in a logical zone except for `runway`. This predicate is defined in

PVS in Figure 2.8, lines 70 and 71.

assigned_approach (line 78): The auxiliary function `assigned_approach` calculates the number of aircraft a on the approach such that $a.\mathsf{mahf} = \mathtt{side}$. This function is defined in PVS in Figure 2.8, lines 74 - 76.

on_approach_qn for a side (line 81): The auxiliary predicate `on_approach_qn` checks if there is an aircraft a on the approach such that $a.\mathsf{mahf} = \mathtt{side}$. This predicate is defined in PVS in Figure 2.8, lines 79 - 81. Note we have two `on_approach_qn` predicates with different types. The one explained earlier is a predicate over the set of aircraft that checks if a given aircraft is on the approach.

actual, virtual (lines 85, 88): One interesting notion in the SATS concept that is used within the preconditions of some transitions is the *virtual number of aircraft* in one side. The virtual number of aircraft in side σ counts the *actual* number of aircraft in side σ, *plus* the number of "potential aircraft" that may possibly come to side σ in the near future if they miss the approach, that is, aircraft that are outside of the initiation area of side σ, but have the mahf assignment of σ. Figure 2.9 shows an example of the virtual number of aircraft in the right initiation area.

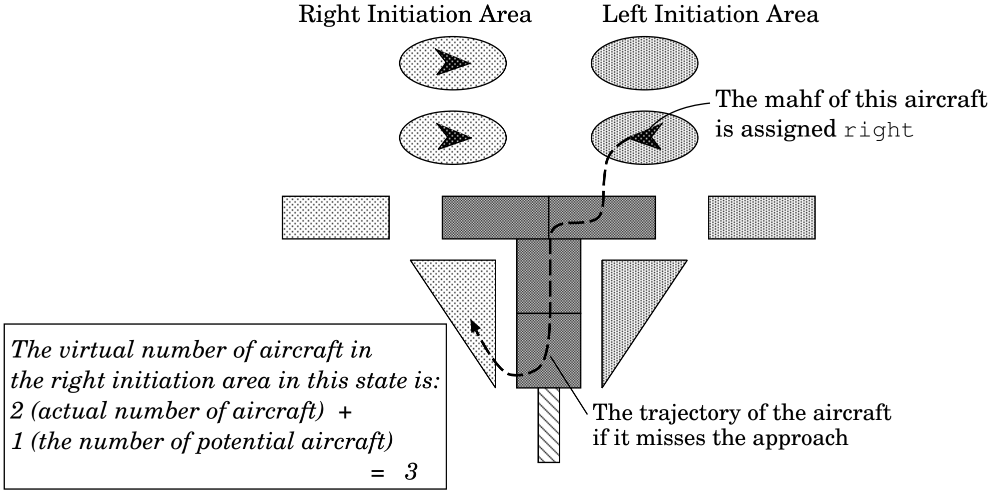

Right Initiation Area
Left Initiation Area
The mahf of this aircraft
is assigned right
The virtual number of aircraft in
the right initiation area in this state is:
2 (actual number of aircraft) +
1 (the number of potential aircraft)
= 3
The trajectory of the aircraft
if it misses the approach

Figure 2.9: The virtual number of aircraft

The actual number is expressed in PVS by the function `actual` defined in Figure 2.8, lines 84 - 86. The virtual number of aircraft is expressed in PVS by the function `virtual` defined in Figure 2.8, lines 89 - 95.

assigned2fix (line 92): The function `assigned2fix` calculates the number of aircraft a in the logical zones such that $a.\mathsf{mahf} = \mathtt{side}$. This function is defined in PVS in Figure 2.8, lines

98 - 104.

arrival_op (line 95): The function `arrival_op` is used to calculate the total number of aircraft in the operation area, and is defined in PVS in Figure 2.8, lines 107 - 109. It counts all aircraft in the logical zones except for the `runway` zone. This function is defined in PVS in Figure 2.8, lines 107 - 109.

move (line 100): The function `move` represents the movement of aircraft in the logical zones. Given two zone names `z_from` and `z_to` and a zone mapping that represents a status of the logical zones, it returns a new zone mapping that represents a new status of the logical zones in which the first aircraft of `z_from` is moved to the last of `z_to`. This function is used in transitions that represents a movement of aircraft. This function is defined in PVS in Figure 2.8, lines 113 - 115.

2.3.2 Transition signatures and state variables of the discrete model

Now we present a formal definition of the model in a subset of TIOA. We split the code into two. The former part states signatures of transitions and state variables, and the latter part states the definition of the transitions (the precondition and effects of them).

We first show the former part of the code in Figure 2.10.

All transitions are declared as output transitions. By having all transitions as output, we consider every transition of the discrete model is an external behavior of the model.

Three state variables are defined for the discrete model (described in the "`states`" section in Figure 2.10). Variable `zones` is a zone mapping from a zone name to an actual zone queue. It represents the status of the logical zones: What aircraft are located on which zone, and the attributes of those aircraft. Initially, this variable is assigned the empty zones.

As explained in Section 2.2.3, variable `nextmahf` is used when the system assigns the `mahf` attribute of entering aircraft. Initially, this variable is assigned `right`.

In our abstract model, we encode the notion of the leader aircraft explained in Section 2.2.2 as an explicit queue of aircraft, called the "*landing sequence.*" Variable `landing_seq` represents the landing sequence in the model. When an aircraft first enters the system, in addition to being added to the logical zones, it is also added to the end of the landing sequence. When an aircraft lands or exits from the operation area, it is removed from the landing sequence. We will closely look at these additions and removals of aircraft in Section 2.3.4. Using function `leader`, the leader of aircraft a in the landing sequence is defined as the aircraft just in front of a in the sequence. By this definition of the leader, the landing sequence represents the chain of the leader relation discussed in Section 2.2.2. The leader relation in the landing sequence is used within the precondition of some specific transitions. We will guarantee by Invariant 3.6 that,

```
automaton SATS
   imports SatsVocab

signature
   output
      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 : Zone % 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_qn(a:Aircraft) = (a = first(landing_seq));

      %% define functions on a zone_map as functions on a state
      on_approach_qn(a:Aircraft) = on_approach_qn(zones, a);
      on_approach_qn(side:Side) = on_approach_qn(zones,side);
      actual(side:Side) = actual(zones,side);
      virtual(side:Side) = virtual(zones,side);
      assigned2fix(side:Side) = assigned2fix(zones,side);
      arrival_op = arrival_op(zones);

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

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

      %% 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)));
```

Figure 2.10: Code for the discrete model [Part 1 of 2. Signature and States]

whenever the precondition is satisfied, the given aircraft `a` for `leader` used in the precondition is in the landing sequence (the given queue for `leader`).

A state of the model is implicitly expressed in TIOA code (for example, `landing_seq` denotes the landing sequence for a given state). On the other hand, an explicit state structure is constructed at a translation stage to PVS code. For instance, in PVS code, the landing sequence is represented by `landing_seq`(s) for a given state s. Analogously, for instance, auxiliary function `actual(side)` will have an argument for a state in PVS (`actual(s,side)` for a given state s)

2.3.3 Mathematical Notations

We have shown the types and auxiliary functions in Section 2.3.1, and formal code for the state variables of the model in Section 2.3.2. We use the following notations in this thesis to mathematically express the conditions on these types and auxiliary functions for a given state of the model.

1. We use the notation *opposite*(σ) for side σ to represent the opposite side of σ.

2. We use || notation to express the number of aircraft in a given zone. For example, $|\texttt{maz}(\sigma)|$ represents the number of aircraft in $\texttt{maz}(\sigma)$ (that is, the length of $\texttt{maz}(\sigma)$). A zone is indexed starting from 0.

3. We use the notation $z[i]$ to express the aircraft in position i in z.

4. We use $z_1 \circ z_2$ to represent the concatenation of two zones (seen as two sequences) z_1 and z_2.

5. We use "." notation to access attributes of aircraft. For example, $a.\mathsf{mahf}$ represents the value of the mahf attribute of aircraft a.

6. We use mathematical notation *leader*(a) to represent a leader of aircraft a in the landing sequence.

7. In Section 2.3.1, we explained auxiliary predicate `precedes_qn` defined in PVS. Mathematically, the *precedes* relation is defined as follows.

   Definition 2.1. The relation *precedes* for a given sequence is the irreflexive transitive closure of the leader relation for the same sequence.

   For example, if a is the leader of b and b is the leader of c, then we say that a precedes c. And for any aircraft a, a does not precede a.

8. We use the mathematical notation *assigned*(z, σ) to express the number calculated by `assigned`(z, σ).

9. We say that *aircraft* a *is in side* σ if and only if `on_qn(`a,*side*`)` holds.

10. We say that a is *on the approach* if and only if on_approach_qn(a) holds.

11. We use the mathematical notation *assigned_approach*(σ) to represent a value of `assigned_approach` for σ.

12. We say that *on_approach_qn*(σ) holds if and only if `on_approach_qn(`σ`)` holds.

13. We use the mathematical notations *actual*(σ) and *virtual*(σ) to represent the values of `actual`(σ) and `virtual`(σ), respectively.

14. We use the mathematical notation $assigned2fix(\sigma)$ to represent the value of `assigned2fix(`σ`)`.

15. We use the mathematical notation *arrival_op* to represent the value of `arrival_op`.

2.3.4 Transitions of the discrete model

In this subsection, we closely examine each transition in the automaton described in the previous subsection. The code for the transition definitions of the automata is shown in Figure 2.11. The transitions of the automaton is described in "*precondition-effect*" style.

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 of the transitions have an attribute `side` because some transitions 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.

Many of the transitions represent the effects that the first aircraft of a specific zone is removed, and is added to the end of another zone. In the rest of the thesis, we refer to this kind of effects (the first aircraft of a zone z_1 is removed and added to the end of another zone z_2) as a *movement of the first aircraft from zone* z_1 *to* z_2. In the formal automaton definition, this effect is described by using function `move` explained in Section 2.3.

transitions

```
output VerticalEntry(a, id, side)
 pre virtual(side) < 2 ∧
     ¬on_approach_qn(side) ∧
     empty_qn(maz(side)) ∧
     empty_qn(lez(side)) ∧
     empty_qn(holding3(side)) ∧
     a = aircraft(side,id) ∧
     ∀ac: Aircraft
      ((on_zones_qn(ac) ∨
        in_queue_qn(ac, landing_seq) ∨
        on_zone_qn(runway, ac)) ⇒ ac.id ≠ id)
 eff zones := enter(holding3(side),side,id,zones);
     landing_seq := add(landing_seq, a);
     nextmahf := opposite(a.mahf);

output LateralEntry(a, id, side)
 pre virtual(side) = 0 ∧
     a = aircraft(side,id) ∧
     ∀ac: Aircraft
      ((on_zones_qn(ac) ∨
        in_queue_qn(ac, landing_seq) ∨
        on_zone_qn(runway, ac)) ⇒ ac.id ≠ id)
 eff zones := enter(lez(side),side,id,zones);
     landing_seq := add(landing_seq,a);
     nextmahf := opposite(a.mahf);

output HoldingPatternDescend(a, side)
 pre ¬(empty_qn(holding3(side))) ∧
     a = first(holding3(side)) ∧
     empty_qn(holding2(side))
 eff zones:=move(holding3(side),holding2(side),zones)

output VerticalApproachInitiation(a, side)
 pre ¬(empty_qn(holding2(side))) ∧
     a = first(holding2(side)) ∧
     length(base(opposite(side))) ≤ 1 ∧
     (first_in_seq_qn(a) ∨
        on_approach_qn(leader(a,landing_seq)))
 eff zones := move(holding2(side),base(side),zones)

output LateralApproachInitiation(a, side)
 pre ¬(empty_qn(lez(side))) ∧
     a = first(lez(side))
 eff IF length(base(opposite(side))) ≤ 1 ∧
          (first_in_seq_qn(a) ∨
          on_approach_qn(leader(a,landing_seq)))
     THEN
        zones := move(lez(side),base(side),zones)
     ELSE
       zones := move(lez(side),holding2(side),zones)
     FI

output Merging(a, side)
 pre ¬(empty_qn(base(side))) ∧
     a = first(base(side)) ∧
     (first_in_seq_qn(a) ∨
    on_zone_qn(intermediate,leader(a,landing_seq))∨
     on_zone_qn(final,leader(a,landing_seq)))
 eff zones := move(base(side),intermediate,zones)
```

```
output Exit(a)
 pre ¬(empty_qn(intermediate)) ∧
     ¬(empty_qn(landing_seq)) ∧
     a = first(intermediate) ∧
     first_in_seq_qn(a)
 eff zones:=
    assign(zones,intermediate,rest(intermediate));
     landing_seq := rest(landing_seq)

output FinalSegment(a)
 pre ¬(empty_qn(intermediate)) ∧
     a = first(intermediate)
 eff zones := move(intermediate, final, zones)

output Landing(a)
 pre ¬(empty_qn(final)) ∧
     ¬(empty_qn(landing_seq)) ∧
     a = first(final) ∧
     empty_qn(runway)
 eff zones := move(final,runway,zones);
     landing_seq := rest(landing_seq);

output Taxiing(a)
 pre ¬(empty_qn(runway)) ∧
     a = first(runway)
 eff zones:= assign(zones, runway, rest(runway));

output MissedApproach(a)
 pre ¬(empty_qn(final)) ∧
     ¬(empty_qn(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);

output LowestAvailableAltitude(a, side)
 pre ¬(empty_qn(maz(side))) ∧
     a = first(maz(side))
 eff IF empty_qn(holding3(side)) ∧
        empty_qn(holding2(side))
     THEN
       zones := move(maz(side),holding2(side),zones)
     ELSE
     IF empty_qn(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
```

Figure 2.11: Code for the discrete model [Part 2 of 2. Transitions]

To help the reader to understand 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. Each of these bounds is desirable basic property of the model, considering the safety separation of aircraft in each holding fix. Now a reader may easily understand, for instance, why it is reasonable that the precondition of entry and descend transitions (`VerticalEntry`, `LateralEntry`, and `HoldingPatternDescend`) check the emptiness of the zone that an aircraft goes to. On the other hand, more complicated preconditions are defined for some transitions: for example, some preconditions refer to a condition on the virtual number of aircraft, or a condition on 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 bounds on the number of aircraft in some zones such as `maz`. This complication comes from the fact that, as we will see in the following, 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.

Entry to the logical zones: VerticalEntry and LateralEntry

An aircraft can enter the operation area in either a vertical way or a lateral way. A vertical entry is represented by the `VerticalEntry` transition, and a lateral entry is represented by the `LateralEntry` transition.

VerticalEntry(a, id, σ)**:** This transition represents the vertical entry of aircraft a to `holding3`(σ) zone. The transition has an attribute `Side`: a vertical entry can be performed either from the right side or from the left side of the airport.

The entering aircraft a is assigned its mahf by this transition. The assignment is determined according to a global system variable, called `nextmahf`. The aircraft is assigned the same side as the current value of `nextmahf`, and every time the system assigns the mahf to an aircraft, it flips the value of `nextmahf`. If there is no aircraft in the system when the new aircraft enters, its mahf is set to the side it enters, and the value of `nextmahf` is set to the opposite side of the mahf of that aircraft. Also, a new aircraft is given a unique ID.

A vertical entry to `holding3(`σ`)` by aircraft a – `VerticalEntry(`a, id, σ`)`[4]– is enabled if the following conditions are true:

- $virtual(\sigma) < 2$.
- $on\_approach\_qn(\sigma)$ does not hold.
- $|\texttt{lez}(\sigma)| = 0$.
- $|\texttt{maz}(\sigma)| = 0$.
- $|\texttt{holding3}(\sigma)| = 0$.
- For any aircraft b already in the logical zones, $a.\mathsf{ID} \neq b.\mathsf{ID}$

The last condition in the above precondition guarantees that only an aircraft whose ID is different from any other aircraft's ID enters the system. This unique ID is given by the Aircraft Management Module in the real system.

The effect of the transition is as follows:

- a is added to the end of `holding3(`σ`)`. $a.\mathsf{mahf}$ is set to the value of `nextmahf` if the landing sequence is not empty; otherwise it is set to the side a enters (σ).
- a is also added to the end of the landing sequence.
- The value of `nextmahf` is set to $opposite(a.\mathsf{mahf})$.

LateralEntry(a, id, σ)**:** This transition represents the lateral entry of aircraft a to $\texttt{lez}(\sigma)$, a lateral entry zone. The transition has an attribute of `Side`, as does `VerticalEntry`. A lateral entry to `lez(`σ`)` by aircraft a – `LateralEntry(`a, id, σ`)` – is enabled if the following conditions are true:

- $virtual(\sigma) = 0$.
- For any aircraft b already in the logical zones, $a.\mathsf{ID} \neq b.\mathsf{ID}$

[4] In the rest of the thesis (mainly in the proofs), we sometimes use the notation VerticalEntry(σ) to just specify the side for which the transition is performed. As we see in the formal description of the abstract model defined as an I/O automaton, this representation of the transition (`VerticalEntry(`σ`)`) is not rigorously correct: the actual `VerticalEntry` transition is parameterized by the new aircraft, the unique ID of it, and the side it enters. Thus, it has the form `VerticalEntry(`a`,`id`,`σ`)`, where a is of type `Aircraft`, id is of type `ID`, and σ is of type `Side`. However, we omit these parameters except for the side in order to simplify the statement. We also sometimes omit attributes of other transitions other than the side attribute.

And the effect of the transition is as follows: transition:

- a is added to the end of lez(σ). a.mahf is set to the value of nextmahf if the landing sequence is not empty; otherwise it is set to the side a enters (σ).
- a is also added to the end of the landing sequence.
- The value of nextmahf is set to $opposite(a.\mathsf{mahf})$.

Note that the first condition in the precondition – $virtual(\sigma) = 0$ – implies that there is no aircraft on side σ (holding2(σ), holding3(σ), lez(σ), and maz(σ)), and that there is no aircraft a with $a.\mathsf{mahf} = \sigma$ in the operation area.

Descend from 300 to 200 feet: HoldingPatternDescend

HoldingPatternDescend(a, σ)**:** This transition represents a descent of the altitude of aircraft a from a holding fix at 3000 feet (holding3) to a holding fix at 2000 feet (holding2).

The first aircraft a on holding3(σ) is allowed to descend (move) to holding2(σ) by HoldingPatternDescend(σ) if no aircraft is in holding2(σ).

Approach Initiation: VerticalApproachInitiation and LateralApproachInitiation

An aircraft can initiate the approach to the ground either from the holding fix at 2000 feet (holding2), or from the lateral entry zone (lez). Every aircraft that initiates the approach first enters the T-shaped approach area by moving to base zone of the side it comes from. The approach initiation from holding2 is represented by VerticalApproachInitiation, and the approach initiation from lez is represented by LateralApproachInitiation.

VerticalApproachInitiation(a, σ)**:** This transition represents a vertical approach initiation of aircraft a from holding2(σ) to base(σ). The first aircraft a of holding3(σ) can initiate the approach to base(σ) by VerticalApproachInitiation(σ) if the following conditions hold:

- a is the first aircraft in the landing sequence, or its leader is already on the approach.
- $|\mathsf{base}(opposite(\sigma))| \leq 1$.

The effect of the transition is the movement of the first aircraft of holding2(σ) to base(σ).

The first condition above ensures that if aircraft a has its leader aircraft, then the leader aircraft has to initiate the approach before a does so. This part of the precondition is described in the actual TIOA code as
"(first_in_seq?(a) ∨ on_approach?(leader(a,landing_seq)))."

LateralApproachInitiation(a, σ)**:** This transition represents a lateral approach initiation of aircraft a from `lez(`σ`)` to `base(`σ`)`. The lateral approach initiation procedure is different from its vertical-case counterpart, `VerticalApproachInitiation`, in that the transition for a lateral approach initiation is always enabled for the first aircraft of `lez`, but it can directly proceed to `base` only when some specific conditions are met. These specific conditions are stated below as Conditions 1 and 2, and actually, as we can see, they are equivalent to the precondition of `VerticalApproachInitiation`, which represents the guard that delays the approach initiation. If the conditions are not met, the aircraft first moves to `holding2` of the same side that the aircraft comes from, and tries to initiate the approach from there by `VerticalApproachInitiation`. Thus, even though the transition is always enabled whenever `lez` is not empty, the effect of the transition is based on the current situation, and thus the aircraft will not actually initiate the approach by this transition in some cases. This is reasonable considering that preserving the order of the leader relation is crucial to satisfy the desirable safety conditions that we will prove in this thesis. (See Figure 2.3 again to see the paths of aircraft that leave from the lateral entry zone.)

The transition `LateralApproachInitiation(`a, σ`)` is always enabled if `lez(`σ`)` is not empty. The effect of the transition is the movement of the first aircraft a of `lez(`σ`)` to `base(`σ`)` if the following two conditions hold:

1. a is the first aircraft in the landing sequence, or its leader is already on the approach.

2. $|\texttt{base}(\mathit{opposite}(\sigma))| \leq 1$.

If the conditions are not satisfied, the effect of the transition is the movement of the first aircraft of `lez(right)` to `holding2(right)`.

Progression in the approach area: Merging, FinalSegment, Exit, and Landing

After the approach initiation, aircraft go through the T-shaped approach zone. The progression of aircraft in this approach zone is represented by three transitions `Merging`, `FinalSegment`, and `landing`. An aircraft under a specific condition may also exit from the airport during the approach to the ground. This procedure is represented by the `Exit` transition.

Merging(a)**:** Once an aircraft enters a `base` zone of either the right side or the left side in the approach area, it merges to the center of the approach area (`intermediate` zone). Since the system aims to ensure that the order of the actual landing of aircraft is the same as the order in the landing sequence, this merging procedure has a precondition analogous to that for the approach initiation procedures.

The first aircraft a of `base(`σ`)` can proceed (move) to `intermediate` by `Merging(`σ`)` if a is the first aircraft in the landing sequence, or its leader is in either the `intermediate` zone or the `final` zone.

FinalSegment(a)**:** The first aircraft a of `intermediate` can always proceed (move) to `final` zone. More specifically, the transition is always enabled if `intermediate` is not empty, and the effect is the movement of the first aircraft of `intermediate` to `final`.

Exit(a)**:** The first aircraft a of `intermediate` may also exit the airport if it is the first aircraft in the landing sequence. Thus the transition `Exit` is enabled when `intermediate` is not empty, and the effect of the transition is the following:

- The first aircraft of `intermediate` is removed from the zone.
- The first aircraft of the landing sequence is removed from the sequence.

Note that the precondition of the transition ensures that the system removes the same aircraft from both the logical zones and the landing sequence.

Landing(a)**:** The first aircraft a of `final` may land on the ground, i.e., proceeds to `runway` if no aircraft is on `runway`. The effect of the transition is the following.

- The first aircraft of `final` moves to `runway`.
- The first aircraft of the landing sequence is removed from the sequence.

At the same time when the first aircraft of `final` is removed from that zone, the first aircraft of the landing sequence is also removed from the sequence. This is because we do not count aircraft on `runway` as active aircraft (they are all “done”), and this is consistent with the fact that we separate `runway` from the operation area.

As opposed to `Exit` transition, the precondition of `Landing` does not guarantee that the system removes the same aircraft from both the `final` zone and the landing sequence, since it is not directly guaranteed from the precondition that the first aircraft in the `final` zone and the first aircraft in the landing sequence is the same aircraft. Instead, we will prove this fact in Section 3.3 as an invariant of the model (Corollary 3.8). That is, we will prove that the first aircraft of `final` is the same as the first aircraft of the landing sequence whenever `final` is not empty.

Taxiing from the runway: Taxiing

Taxiing(a)**:** The transition `Taxiing` removes the first aircraft a of `runway`, and it is always enabled when `runway` is not empty.

Note that the transition does not remove an aircraft from the landing sequence since it has already been removed from there when it lands, by the `Landing` transition.

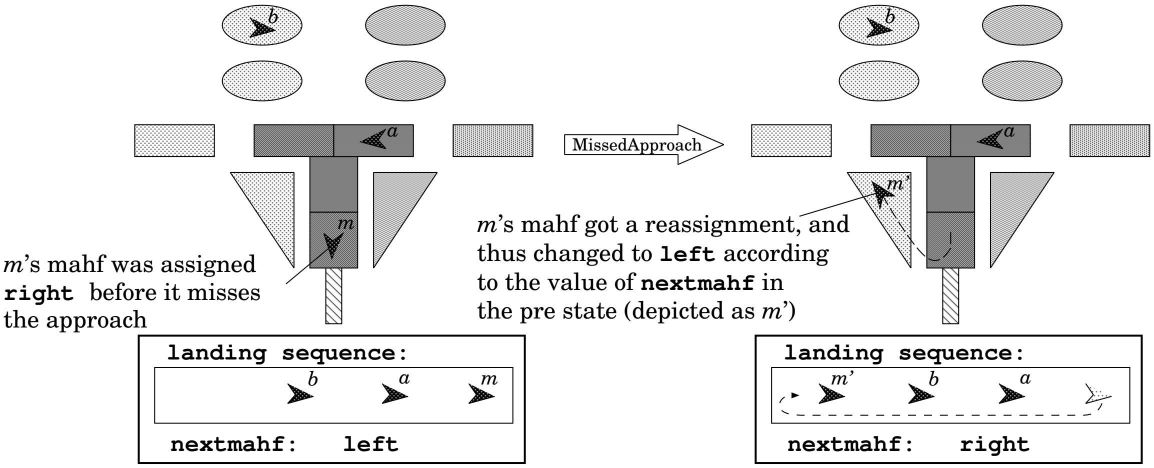


Figure 2.12: Missed Approach Initiation

Missed Approach Initiation: MissedApproach

MissedApproach(a)**:**

An aircraft on `final` zone may miss the approach. A missed approach is represented by the `MissedApproach` transition. The transition is always enabled if `final` is not empty.

By the transition, an aircraft that misses the approach goes to `maz` zone at the side that it is assigned as `mahf`. The landing sequence also changes as we present in the following. A key point of the transition is that the aircraft is reassigned its mahf when it misses the approach.

The first aircraft of the landing sequence is removed, and the same aircraft with the *reassignment* of the mahf is added to the end of the sequence. This reassignment process is done in a way analogous to the case of the entry of an aircraft as we see in the following.

The effect of `MissedApproach(`a`)` is explained in the following.

- The first aircraft a of the landing sequence is removed from the sequence.
- The first aircraft of `final` is removed from there.
- Aircraft a', the aircraft a with the *reassignment* of mahf, is added to the end of the

sequence. The re-assignment is done as follows: If the landing sequence is not empty, then a.mahf is set to the current value of `nextmahf`; otherwise, a.mahf is not re-assigned.

- a' is also added to the end of maz(a.mahf) (note a.mahf is the mahf attribute of a before the re-assignment.)
- The value of nextmahf is set to *opposite*(a'.mahf) (note we are referring to a', not a).

The reassigned first aircraft of the landing sequence, a', is added to *both* the landing sequence and a maz zone. Without guaranteeing that the first aircraft of the landing sequence is the same as the first aircraft of the final zone, the aircraft added to a maz zone by the transition may potentially be totally irrelevant to the first aircraft of final zone. Indeed, the precondition of the transition does not guarantees the coincidence of two aircraft. However, as we mentioned in the discussion of the Landing transition, we will prove the invariant that states the first aircraft of final is the same as the first aircraft of the landing sequence.[5]

An example of the transition MissedApproach is illustrated in Figure 2.12.

Return to the vertical initiation zone from missed approach: LowestAvailableAltitude

LowestAvailableAltitude(a, σ)**:**

After aircraft a has missed the approach and thus goes to missed approach zone, it goes back to the vertical initiation area, holding2(σ) or holding3(σ) depending on the situation at the moment when it leaves maz(σ). This procedure is represented by LowestAvailableAltitude(a, σ) transition. As the name of the transition implies, the aircraft seeks the lowest available altitude from holding2(σ) and holding3(σ) (the former is lower than the other) in the current situation, and moves to that holding fix. An interesting point of this transition is that it may exhibit a “double transition” with which two aircraft moves the logical zones with one transition. This is the only transition that may exhibit a multiple transition of aircraft; the rest of the transitions handle just one aircraft at a time.

More formally, the transition LowestAvailableAltitude(a, σ) is enabled for the first aircraft a of maz(σ) whenever maz(σ) is not empty, and the transition has the following three effects depending on the state of the model in which the transition is performed:

- If both holding2(σ) and holding3(σ) are empty, then the first aircraft of maz(σ) moves to holding2(σ).

[5] We could have taken another approach in the definition of the model: adding the reassigned first aircraft of final to maz, and the reassigned first aircraft of the landing sequence to the sequence. But even with this approach, we again need the equality of the first aircraft of final and the landing sequence in order to guarantee that the model is adding the same aircraft to the landing sequence and the operation area.

- If `holding3(`σ`)` is empty, but `holding2(`σ`)` is non-empty, then the first aircraft of `maz(right)` moves to `holding3(right)`.

- If `holding3(`σ`)` is not empty, then the first aircraft of `maz(`σ`)` moves to `holding3(`σ`)`, and at the same time, the first aircraft of `holding3(`σ`)` moves to `holding2(`σ`)`.

The double transition of aircraft in the third case is illustrated in Figure 2.13.

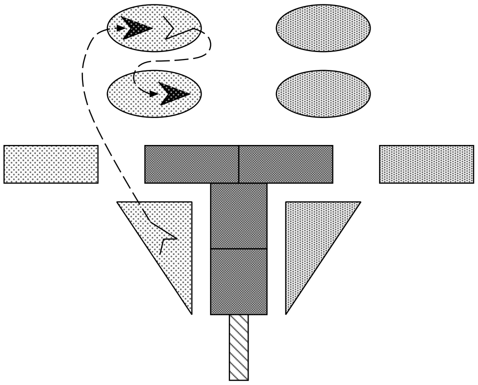


Figure 2.13: The double transition of aircraft in the third case of LowestAvailableAltitude(right)

When a double transition of aircraft occurs, an aircraft in `holding3(`σ`)` goes to `holding2(`σ`)` without any special condition (the transition is always enabled when `maz(`σ`)` is not empty.) It might seem that this fact jeopardizes the upper bound $|$`holding2`$(\sigma)| \leq 1$, stated in the introduction of this section: if there is an aircraft in `holding2(`σ`)`, `holding3(`σ`)`, and `maz(`σ`)`, respectively, in the pre state of the transition, then $|$`holding2`$(\sigma)|$ exceeds one in the post state. However, we can guarantee that this scenario would never occur: we will prove (as part of the conditions for Lemma 3.26) that whenever both `holding3(`σ`)` and `maz(`σ`)` are non-empty, `holding2(`σ`)` is empty.

Here we consider why the original authors of the discrete model of [2] needed this double transition for `LowestAvailableAltitude`. By moving an aircraft in `holding3`(σ) to `holding2`(σ) when another aircraft is entering `holding3`(σ) (a double transition), the discrete model of [2] guarantees that the bound $|$`holding3`$(\sigma)| \leq 1$ would not be violated after the `LowestAvailableAltitude` transition. Indeed, if `LowestAvailableAltitude`(σ) does not exhibit a double transition, but move only the first aircraft of `maz`(σ) to `holding3`(σ) when $|$`holding3`$(\sigma)| = 1$, the post state of the transition would violate the bound $|$`holding3`$(\sigma)| \leq 1$. In the original paper [2], the authors partially justified using this double transition in the discrete model, by claiming that the double transition exhibited in

LowestAvailableAltitude would never occur in a real protocol since the SATS concept precludes an aircraft from hovering at one altitude when a lower altitude is available. This indicates that when a missed aircraft moves to holding3 by LowestAvailableAltitude, the aircraft previously in holding3 has already descended to holding2 by HoldingPatternDescend. However, we cannot verify such a property for the discrete model since the model does not have any time-dependent constraints, such as the time bound for HoldingPatternDescend to be performed after the moment a lower altitude becomes available (that is, holding2 becomes empty). We will present a new time-dependent model in Chapter 5 that does not exhibit any multiple transition of aircraft. In this new model, the above mentioned property for an order of LowestAvailableAltitude and HoldingPatternDescend is proved by assuming a reasonable time-dependent constraints, and thus LowestAvailableAltitude deals with only one aircraft at a time in the new model.

Chapter 3

Properties of the discrete model and their proof

3.1 Introduction

In this chapter, we state and prove safety properties of the discrete model of SATS presented in Chapter 2. The properties are taken from the original paper [2].

In Section 3.2, we present the main properties we prove in this chapter. The properties mainly represent upper bounds on the number of aircraft in particular areas of the airport. Section 3.3 is dedicated to introducing and proving the auxiliary invariants that we consider to be the most basic properties of the model. In Section 3.4, we start proving the main properties, using the basic properties proved in Section 3.3. In Section 3.5, we also introduce the important notion of "blocking" of aircraft. Using this notion, we strengthen some of the main properties, and prove them in Sections 3.6 and 3.7. In Section 3.8, we discuss some issues concerning proofs in PVS.

3.2 Main Properties

In this section we present the main properties of the abstract model of SATS that we will prove in the rest of the chapter. We present seven properties taken from the original paper [2], which are chosen since these properties can be also used to prove the safe separation property of aircraft in the continuous model presented in Chapter 5, by carrying over the results in this chapter using a refinement mapping. Five main properties out of seven state upper bounds on the number of aircraft in specific zones or areas.

In the rest of the thesis, we will also present the corresponding PVS code used for mechanical theorem-proving of the properties.

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 a predicate that expresses the corresponding 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

Property 1: The total number of aircraft in the operation area is *at most four* ($arrival\_op \leq 4$).

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

Property 2: The total number of aircraft in one side is *at most two* ($\forall \sigma : Side, actual(\sigma) \leq 2$).

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

Property 3: $\forall \sigma : Side, |\mathsf{holding2}(\sigma)| \leq 1 \wedge |\mathsf{holding3}(\sigma)| \leq 1$. (The number of aircraft in each vertical holding fix is *at most one.*)

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

Property 4: $\forall \sigma : Side, |\mathsf{maz}(\sigma)| \leq 2$. (The number of aircraft in a missed approach zone is *at most two.*)

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

Property 5: $\forall \sigma : Side, |\mathsf{lez}(\sigma)| \leq 1$. (The number of aircraft in a lateral entry zone is *at most one.*)

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

Property 6: If $\texttt{lez}(\sigma)$ is not empty, then the vertical holding fixes and the missed approach zone of the same side ($\texttt{holding2}(\sigma)$, $\texttt{holding3}(\sigma)$, and $\texttt{maz}(\sigma)$) 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: $\forall \sigma : Side, assigned2fix(\sigma) \leq 2$. (The number of aircraft assigned to one side as their mahf in the operation area is *at most two.*)

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

All properties are proved using an induction over steps of the abstract model (the length of the sequence of transitions the model takes), some of which need to be strengthened, or need to be proved together to make an inductive proof work. We will closely examine the dependency and which properties have to be proved together in Sections 3.4 - 3.7, which are devoted to the proof of these main properties.

There are some facts that one may notice from these statements of properties. First, since an initiation area includes a missed approach zone of the same side, Property 2 ($actual(\sigma) \leq 2$) implies Property 4 ($|\texttt{maz}(\sigma)|$). Yet it is reasonable to have Property 2 and Property 4 separately, first because they are stated separately in [2], and more importantly because we actually have to use the stated upper bound on $|\texttt{maz}(\sigma)|$ to prove the stated upper bound on $actual(\sigma)$. Again, we will examine the details in Sections 3.4 - 3.7.

To see an example why some properties need to be strengthened, let us consider proving Property 2 ($\forall\sigma : Side, actual(\sigma) \leq 2$). We can see, from the following, the reason why this property by itself is not strong enough to make an inductive argument. Suppose, in some state s, aircraft a with $a.\texttt{mahf} = \sigma$ is in `final`, and it misses the approach. Thus an aircraft enters $\texttt{maz}(\sigma)$. Since we just assume $actual(\sigma) \leq 2$ in the induction hypothesis, if $actual(\sigma) = 2$ before the transition, we cannot guarantee that $actual(\sigma) \leq 2$ in the post state of the missed approach transition.

One might consider strengthening Property 2 using the *virtual number* of aircraft ($virtual(\sigma)$) introduced in Section 2.3.2, instead of using the actual number of aircraft ($actual(\sigma)$). Since $virtual(\sigma) \geq actual(\sigma)$ for any state, we could prove Property 2 by proving that $virtual(\sigma) \leq 2$ in any reachable state of the model. However, this approach would not work since the value of $virtual(\sigma)$ *can* exceed two in some reachable states. To see the reason, consider the following scenario depicted in Figure 3.1.

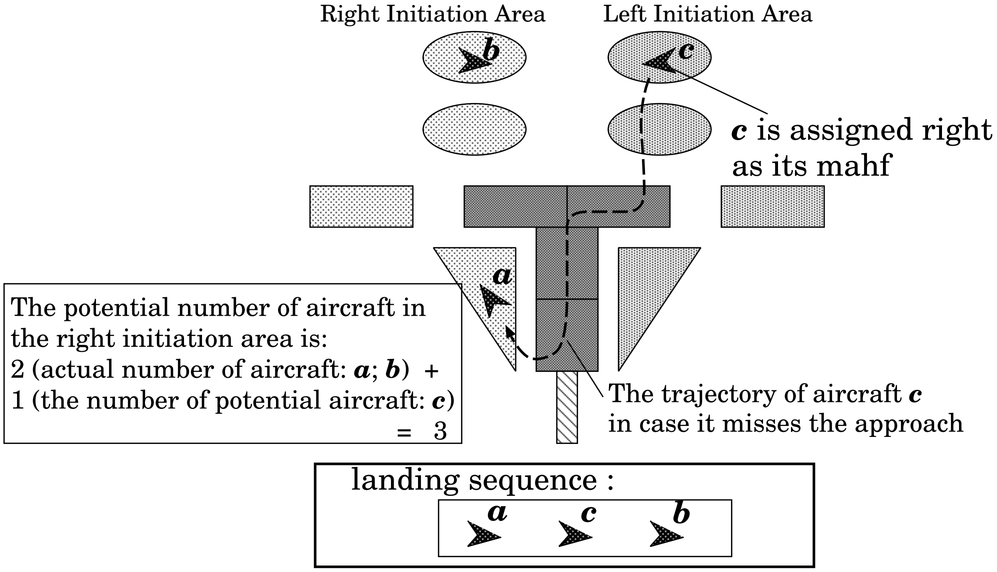

Right Initiation Area
Left Initiation Area
b
c
c is assigned right as its mahf
a
The potential number of aircraft in the right initiation area is:
2 (actual number of aircraft: a; b) +
1 (the number of potential aircraft: c)
= 3
The trajectory of aircraft c in case it misses the approach
landing sequence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

Figure 3.1: An example of a state in which the virtual number of aircraft on the right side is more than two

Such a situation can be achieved by the following execution A of the model.

Execution A:

1. First, aircraft a enters `holding3(right)` by `VerticalEntry`, and is assigned its mahf according to the side it enters, which is `right`, since it is the first aircraft that enters the

system. The value of `nextmahf` is set to `left` (the opposite side of the mahf of the new aircraft).

2. Second, aircraft a descends to `holding2(right)` by `HoldingPatternDescend`. This transition is enabled since there is no aircraft in `holding2(right)`.

3. Third, aircraft b enters `holding3(right)` by `VerticalEntry`, and is assigned `left` as its mahf. The value of `nextmahf` is flipped to `right`. we can easily verify that the transition is indeed enabled in this state.

4. Next, aircraft c enters `holding3(left)` by `VerticalEntry`, and is assigned `right` as its mahf. Again, we can easily verify that the transition is enabled in this state.

5. Then, aircraft a initiates the approach by `VerticalApproachInitiation`, and moves to `base(right)`. This transition is enabled since a is the first aircraft in the landing sequence, and $|\texttt{base}(\texttt{left})| = 0$.

6. Finally, aircraft a proceeds to `final` in the approach area by two transitions, `Merging` and `FinalSegment`. Then, it misses the approach (`MissedApproach`), and hence goes to `maz(right)` according to its mahf assignment. These two transitions are enabled in our scenario, considering that a is the first aircraft in the landing sequence.

Thus the state of the model after the above sequence of transitions is one of the reachable states. Furthermore, as we can see from the picture, the value of $virtual(\mathsf{right})$ is *three* in that state.

Even though $virtual(\texttt{right})$ exceeds two, the above scenario would not jeopardize the upper bound on $actual(\texttt{right})$ stated in Property 2. A potentially problematic scenario is that c initiates and misses the approach after following the above steps, and therefore goes to `maz(right)`. This leads to the case that $actual(\texttt{right}) = 3$. However, this would not be the case because aircraft c has the leader aircraft b, and b has not initiated the approach yet. Aircraft b has to leave the right initiation area before c initiates the approach. In other words, aircraft b "*blocks*" the aircraft c in this situation. This example leads to a notion of *blocking* of aircraft. To prove the upper bound on $actual(\sigma)$ stated as Property 2, we have to take into account not only mahf assignments of aircraft outside of side σ, but also a blocking relation between aircraft. We will define this notion of blocking formally in Section 3.5.

A scenario analogous to the one presented above was discussed in [2], and the authors argued that Property 2 would not be violated in this scenario since, as we discussed above, aircraft b has to leave the right initiation area before c initiates the approach. However, they did not need to define the notion of "blocking" to verify the property, since they verified it by a model-checking (state exploration).

3.3 Auxiliary invariants

Before we start proving the main properties presented in the previous section, we first state and prove auxiliary invariants of the abstract model. Invariants in this section are basic to the model, and are heavily used in proofs in the succeeding sections.

In Section 3.3.1, we prove basic properties that guarantee that the model is validly defined so that it reflects the system in reality. These properties include the uniqueness of aircraft and the correspondence between the operation area and the landing sequence.

In Section 3.3.2, we move on to higher-level invariants of the system that we can infer from each step of the model. For example, the model has a local rule for the order of approach initiations of aircraft: an aircraft cannot initiate the approach until its leader initiates. We extend this rule to a global rule that refers to any two aircraft in the entire landing sequence.

3.3.1 Uniqueness of aircraft and the correspondence between the logical zones and the landing sequence

The first basic property to be proved in this section is the uniqueness of aircraft in the system. In a real system of SATS, each aircraft has its own unique ID, by which the system distinguishes aircraft. This basic fact is important to the SATS concept since, as we have seen in Section 3.2, maintaining the correct order of aircraft in the system is crucial in satisfying the safety properties of the model: The existence of two or more identical aircraft could confuse the order of approach initiation and the landing. Thus, it is clearly desirable for the model to possess some kind of uniqueness property of aircraft.

In addition, we need some type of correspondence between the operation area and the landing sequence. This is because the landing sequence is an abstract notion we developed in order to model the leader relation in a real system, and we have to match the aircraft in this abstract sequence with the aircraft in the operational area, by proving, for example, if an aircraft is in the operation, it is also in the landing sequence.

Since invariants in this subsection are the most basic properties of the model, we have to use them in almost every proof of properties and other auxiliary invariants in the rest of this chapter. A proof for these invariants themselves is no exception: some invariants in this subsection have to be proved together to make an inductive proof work since they depend on each other.

We start by proving the uniqueness of aircraft. A natural way to describe the uniqueness property of aircraft is to define it in terms of their IDs (recall that an ID is one of the attributes of the Aircraft data type). That is, we define the uniqueness of aircraft as the fact that all aircraft in the system are respectively assigned different IDs. Since, if IDs of two aircraft are different, those aircraft are different (in terms of the attribute-wise equality, which is used in PVS), the uniqueness of ID immediately implies the uniqueness of aircraft.

Note also that we need the uniqueness property for both the logical zones and the landing sequence. This is because, before obtaining some correspondence results (which we will prove as invariants of the model *using* the uniqueness property), a uniqueness result for one of these two cannot carry over to a uniqueness result for the other.

Uniqueness of ID and aircraft on the logical zones

The uniqueness of ID in the logical zones is defined as follows.

Lemma 3.1. (Uniqueness of ID of aircraft on the logical zones) *For any reachable state of $SATS$, the following two conditions hold:*

(i). (The uniqueness of ID in one zone) *For any logical zone z and natural numbers i and j, where $i < j < |z|$, $z[i].\mathsf{ID} \neq z[j].\mathsf{ID}$.*

(ii). (The uniqueness of ID between different zones) *For any two logical zones z_1 and z_2 such that $z_1 \neq z_2$, and any two aircraft $a \in z_1$ and $b \in z_2$, $a.\mathsf{ID} \neq b.\mathsf{ID}$.*

The above lemma is stated formally in PVS as follows.

```
ID_uniqueness(zones:zone_map): bool =
  %% ID uniqueness on the same zone
  (FORALL (i,j: nat, z:z_name):
     i < length(zones(z)) and j < length(zones(z)) and i<j IMPLIES
       nth(zones(z),i)`id /= nth(zones(z),j)`id)
  AND
  %% ID uniqueness between different zones
  (FORALL (a,b: Aircraft, z1,z2:z_name):
     z1/=z2 AND
     (on_zone?(zones(z1),a) AND on_zone?(zones(z2),b)) IMPLIES
       a`id /= b`id)
ID_uniqueness_lemma: LEMMA
  (FORALL (s:states): reachable(s) => ID_uniqueness(zones(s)))
```

Proof. By induction. The base case vacuously holds since there is no aircraft on any logical zone in the initial state.

Inductive step: Suppose the condition holds in the pre state of a transition. When an aircraft is added to a logical zone, the system assigns an ID to the aircraft in a way that the ID is different from any other IDs of aircraft that are already in the logical zones, thus the condition holds in the post state in the case of `VerticalEntry` and `LateralEntry`.

In the case of the transition that moves an aircraft from zone z_1 to zone z_2, it is sufficient to check the uniqueness in z_1 and z_2, and the uniqueness between zones. The uniqueness of ID in z_1 is preserved by the transition since we just remove the first aircraft. The uniqueness of ID in z_2 holds since from the induction hypothesis, the ID of the aircraft that moves into z_2 is different from the ID of any aircraft on z_2 in the pre state. Finally, the uniqueness between zones is preserved because of the following reason. The uniqueness between z_1 and z_2 holds from the following: The IDs of aircraft in z_2 other than the newly added aircraft are different from those in z_1 since the uniqueness between zones holds in the pre state. And the ID of the newly added aircraft to z_2 is different from any aircraft in z_1 from the uniqueness of ID in z_1 in the pre state. The uniqueness between either z_1 or z_2 and any other zone z_3 holds from the uniqueness between z_1 and z_3, and between z_2 and z_3 in the pre state, as it implies all IDs of the aircraft in z_1 and z_2 are different from the ID of an aircraft on z_3. The case of `MissedApproach` can be treated same way as above since the reassignment of the mahf of an aircraft does not change its ID, and thus the transition can be considered as that an aircraft just moves from `final` zone to `maz` zone when we just focus on ID of aircraft.

The final case is for a transition that removes an aircraft from logical zones (`Taxiing` or `Exit`) and the condition immediately holds from the induction hypothesis. □

The uniqueness of aircraft in the logical zones are defined analogously to the uniqueness of ID. As we stated earlier in this subsection, the uniqueness of ID on the logical zones implies the uniqueness of aircraft on the logical zones.

Corollary 3.2. (Uniqueness of aircraft on the entire logical zones) *For any reachable state of the abstract model, the following two conditions hold:*

(i). (The uniqueness of aircraft on one zone) *For any logical zone z and natural numbers i and j where $i < j < |z|$, $z[i] \neq z[j]$.*

(ii). (The uniqueness of aircraft between different zones) *For any two logical zones z_1 and z_2 such that $z_1 \neq z_2$, If $a \in z_1$, then $a \notin z_2$.*

Corollary 3.2 is stated in PVS as follows.

```
%% uniqueness for zones
uniqueness(zones:zone_map): bool =
   %% uniqueness on the same zone
   (FORALL (i,j: nat, z:z_name):
      i < length(zones(z)) and j < length(zones(z)) and i<j IMPLIES
        nth(zones(z),i) /= nth(zones(z),j))
   AND
   %% uniqueness between different zones
   (FORALL (a: Aircraft, z1,z2:z_name):
      z1/=z2 IMPLIES
      (in_queue?(a,zones(z1)) IMPLIES NOT in_queue?(a,zones(z2))))
uniqueness_lemma: LEMMA
  (FORALL (s:states): reachable(s) => uniqueness(zones(s)))
```

Correspondence of the number of aircraft between the operation area and the landing sequence

Before moving on to the uniqueness of ID of aircraft in the landing sequence, we prove the correspondence of the number of aircraft between the operation area and the landing sequence, which we need in order to prove the uniqueness of ID in the landing sequence. The lemma states that the number of aircraft in the operation area (*arrival_op*) exactly matches the number of aircraft in the landing sequence (|`landing_seq`|) in any reachable state. Using this lemma, we can also prove the upper bound on the number of the aircraft in the operation area by proving the upper bound on the number of aircraft in the landing sequence.

The statement of the lemma is as follows.

Lemma 3.3. *For any reachable state of $SATS$, $arrival\_op = |\texttt{landing\_seq}|$*

The lemma is formally stated in PVS as follows.

```
n_of_ac_coincides_lemma: LEMMA
        (FORALL (s:states):
         reachable(s) => (arrival_op(s) = length(landing_seq(s))))
```

Proof. A proof is straightforward by induction. For the base step, there is no aircraft in either the landing sequence or the operation area, thus the condition holds.

Inductive step: In the case that the action adds an aircraft to the operation area (`VerticalEntry` and `LateralEntry`), it also adds one aircraft to the landing sequence. Thus the condition holds. In the case that the action removes an aircraft from the operation area (`Exit` or `Landing`), the action also removes one aircraft from the landing sequence, thus the condition holds. The rest of the transitions do not add or remove an aircraft from either the landing sequence or the operation area. Note that `MissedApproach` moves aircraft with a reassignment of mahf, but does not change the total number of aircraft in both the logical zones and the landing sequence. □

Uniqueness of ID and aircraft in the landing sequence

Now we prove the uniqueness of ID of aircraft in the landing sequence. It turns out that in order to make an inductive proof work, we need to prove two more invariants together with this uniqueness property. We first state the uniqueness of ID of aircraft in the landing sequence as Invariant 3.4. Next we state two other invariants, Invariants 3.5 and 3.6. Finally we prove these three together as Lemma 3.7.

The uniqueness of ID of aircraft in the landing sequence is formally stated as follows.

Invariant 3.4. (Uniqueness of ID of aircraft on the landing sequence) *For any reachable state of SATS, the following condition holds:*

> *For any natural numbers i and j where $i < j < |\mathsf{landing\_seq}|$, $\mathsf{landing\_seq}[i].\mathsf{ID} \neq \mathsf{landing\_seq}[j].\mathsf{ID}$*

Invariant 3.4 is stated in PVS as follows.

```
%% ID uniqueness for one queue
ID_queue_uniqueness(q:queue): bool =
   (FORALL (i,j: nat):
      i < length(q) and j < length(q) and i<j IMPLIES
        nth(q,i)`id /= nth(q,j)`id)
%% ID uniqueness lemma for the landing sequence
landing_seq_ID_uniqueness_lemma: LEMMA
   (FORALL (s:states):
    reachable(s) => ID_queue_uniqueness(landing_seq(s)))
```

We now introduce the two invariants that need to be proved together with Invariant 3.4.

The first invariant states that the combined zone of `intermediate` and `final` (the concatenation `intermediate` ∘ `final`) is exactly the same as the first portion of the landing sequence of the same length. In other words, the order of aircraft in the approach area after merging from the `base` zones to the `intermediate` zone by `Merging` is exactly the same as the order of aircraft in the landing sequence. This property implies Corollary 3.8, stated later, which states that the aircraft that is about to land (that is, the first aircraft in `final` zone) is the first aircraft in the landing sequence, and therefore the order of the landings of aircraft matches the order of aircraft in the landing sequence. This corollary is used in almost every inductive proof in the rest of the paper when we analyze the cases of `Landing` and `MissedApproach`.

Invariant 3.5. (Strong correspondence between the landing sequence and zones intermediate and `final`) *Let $c =$ `intermediate` ∘ `final`. For any reachable state of SATS, the following condition holds:*

> *For any natural number i where $i < |c|$ and $i < |$`landing_seq`$|$, $c[i] = \mathsf{landing\_seq}[i]$.*

Invariant 3.5 is stated in PVS as follows.

```
first_final_support: LEMMA
       (FORALL (s:states):
         reachable(s) =>
           LET final_approach_area = append(final(s),intermediate(s)) IN
            FORALL (i:nat):
              i<length(landing_seq(s)) AND i<length(final_approach_area)
                IMPLIES  nth(landing_seq(s),i) = nth(final_approach_area,i))
```

The second invariant states that any aircraft in the operation area is also in the landing sequence. The correspondence here is weaker than Lemma 3.5 in the sense that we do not have the exact correspondence referring to the order of aircraft. However, this lemma applies to any aircraft in the operation area, not just in `intermediate` zone and `final` zone. This property is also a most basic properties of the model, and is used in a proof of main properties.

Invariant 3.6. (Weak correspondence between the landing sequence and the operation area) *For any reachable state of SATS, if an aircraft is in the operation area, then it is also in the landing sequence.*

Invariant 3.6 is stated in PVS as follows.

```
queue_correspondence_lemma: LEMMA
  (FORALL (s:states, a:Aircraft):
    reachable(s) =>
    (on_zones?(s,a) IMPLIES in_queue?(a, landing_seq(s))))
```

Now we prove these three invariants together.

Lemma 3.7. *For any reachable state of the abstract model, the invariant conditions stated in Invariants 3.4, 3.5, and 3.6 hold.*

Before going into a detailed proof, we here state why we need to prove these two additional invariants, Invariants 3.5 and 3.6, together with the uniqueness of ID in the landing sequence. As we examine in the following, we need Invariant 3.5 as an induction hypothesis to prove Invariant 3.4, and vice versa, and also need Invariant 3.5 as an induction hypothesis to prove Invariant 3.6 and vice versa.

We first consider the reason why Invariant 3.5 and Invariant 3.4 depend on each other. To prove the case of `MissedApproach` in an inductive proof for Invariant 3.4, we need the condition of Lemma 3.5 in the pre state for the following reason. In this case, the transition removes the first aircraft of the landing sequence, and adds the reassigned first aircraft of `final` to the landing sequence. We have to guarantee that the aircraft removed from `final` zone has the same ID as the aircraft added to `maz` (though it may have a different mahf assignment), since otherwise we do not know any relation between the ID of the added aircraft and the ID of aircraft that have already been in the landing sequence, and thus cannot guarantee the uniqueness of ID.

On the other hand, when we analyze the case of `Merging` in an inductive proof for Invariant 3.5, we need the uniqueness of aircraft in the landing sequence just as we will present it in the proof. Thus we need Invariant 3.4.

Now we consider why Invariant 3.6 and Invariant 3.5 depend on each other. We first examine the reason why we need Invariant 3.5 to prove Invariant 3.6. The statement of Invariant 3.6 may seem somewhat obvious: aircraft in the operation area are also in the landing sequence. However, we have not obtained the fact that the aircraft that lands by `Landing` transition, and thus moves out of the operation area, is indeed the first aircraft of the landing sequence. Hence, in the case of `Landing` action, we do not know if the model removes the same aircraft from the operation area and the landing sequence. Since Invariant 3.5 implies this fact, we need Invariant 3.5 to prove Invariant 3.6.

The reason why we need Invariant 3.6 to prove Invariant 3.5 is that, by using Invariant 3.6, we can guarantee that whenever a transition that uses the `leader` function (`VerticalApproachInitiation`, `LateralApproachInitiation`, or `Merging`) is enabled, `leader` indeed returns the correct leader of the given aircraft (recall that `leader` returns a default value if the aircraft given to the function is not in the given queue, in order to guarantee that the value of the function is always well-defined.) For example, the precondition of `Merging` refers to the leader of the first aircraft of `base` zone. Nevertheless, the precondition does not have an assumption that the first aircraft of `base` is actually in the landing sequence. If we assume Invariant 3.6 in the pre state, we can guarantee that the `leader` function indeed returns the correct leader of the given aircraft, since the lemma implies that the first aircraft of `base` is in the landing sequence.

Proof. We prove three conditions together by induction, that is, by assuming all three of them as the induction hypothesis. We refer to the combined zone of `final` and `intermediate` as just the "combined zone" in the following.

For the base case, the landing sequence and the operation area, and thus also the combined zone, are all empty. Hence three conditions trivially hold. Now we start the induction step.

First Condition: We start by proving the first condition, i.e., the condition for Lemma 3.4.

- *In the case of the transitions for an entry of aircraft (*`VerticalEntry` *and* LateralEntry*):* The transition adds an aircraft with the ID that is not equal to the ID of any aircraft that are already in the landing sequence. Thus the uniqueness of aircraft is preserved in the post state.

- *In the case of* `Exit` *and* `Landing`:

  The transition removes an aircraft from the operation area. Since the removal of aircraft does not affect the uniqueness of aircraft, it is preserved in the post state.

- *In the case of* `MissedApproach`:

  The transition removes the first aircraft of both `final` and the landing sequence, and adds the reassigned first aircraft of `final`. The second condition of the induction hypothesis implies the first aircraft of the `final` zone is also the first aircraft of the landing sequence. Thus, the transition removes the first aircraft from the landing sequence, and adds the same aircraft with new assignment of mahf to the sequence. Since this reassignment does not affect the ID of the aircraft, the uniqueness is still preserved in the post state.

The rest of the transitions do not affect the landing sequence, or the ID of aircraft. Thus the condition immediately follows from the induction hypothesis. [End of the proof of the first condition.]

Second Condition: Now we prove the second condition of the claim, that is, the condition for Lemma 3.5. we start by proving the most interesting case, the case of `Merging` transition.

- *In the case of* `Merging`:

  The transition moves an aircraft from `base` to `intermediate`. The condition for aircraft other than the aircraft that just joins c by the transition holds from the induction hypothesis since the transition just adds an aircraft to c. Now we show the condition for the aircraft that just joins c by the transition. Let us call this newly coming aircraft a.

  The precondition of the transition ensures that a is either the first aircraft of the landing sequence, or the leader of a is already in c. In the former case, if c is empty in the pre state, then we are done, since a will be the first aircraft in both the landing sequence and c in the post state, as required. We claim that c is empty in the pre state. For a contradiction, suppose it is not empty in the pre state. It follows that $|c| \geq 1$. From the induction hypothesis, the first aircraft of c is the same as the first aircraft of the landing sequence, which is a. It implies that a is both in c and in `base` in the pre state. This contradicts the uniqueness of aircraft in the logical zones stated as Corollary 3.2.

  Next, we consider the case that the leader of a is in c in the pre state. In this case, we use the condition of Invariant 3.6 to ensure that `leader` function returns the correct leader: from the induction hypothesis, every aircraft in the operation area is also in the landing sequence in the pre state. Since a is in the `base` zone, a is in the landing sequence. Furthermore, since we are considering the case in which a is not the first aircraft of the sequence, `leader(a,` landing_seq) returns the correct leader of a. Now we prove the correspondence. Let l be $|c|$ in the pre state. Thus for the last aircraft b in c in the pre state,$b = c[l-1]$. From Lemma 3.3, $|c| \leq |\texttt{landing\_seq}|$. Thus $\max(|c|, |\texttt{landing\_seq}|) = |\texttt{landing\_seq}|$. Since a becomes the aircraft at position l in c in the post state, it is sufficient to prove that a is also located at position l in the landing sequence. Suppose, for a contradiction, that a is located at another position $l' \neq l$ in the landing sequence. Since the leader of a is already in c, it follows from the induction hypothesis that for $leader(a)$, there is some position j such that $leader(a) = c[j] = \texttt{landing\_seq}[j]$. Considering the uniqueness of aircraft in the landing sequence in the pre state from the induction hypothesis, $leader(a)$ appear just once in the landing sequence, and thus is just positioned at $l' - 1$ in the landing sequence. It implies that $l' - 1 \leq l - 1$ because otherwise the position of the leader $l' - 1$ exceeds $|c|$ in the pre state. It follows that a is positioned at $l' < l$ in the landing sequence. This implies from the induction hypothesis that $a = \mathsf{landing\_seq}[l'] = c[l']$. It follows that a is in c and is also in base in the post state. However, this contradicts the uniqueness of aircraft on the operation area stated in Corollary 3.2.

- *In the case of* FinalSegment:

  The transition moves the first aircraft of intermediate to the end of final. Since the combined zone is not changed by the transition, the condition holds from the induction hypothesis.

- *In the case of* Landing *and* MissedApproach:

  The transition removes the first aircraft from both final zone and the landing sequence. The correspondence of aircraft in two sequences follows from the induction hypothesis.

- *In the case of* Exit:

  The transition removes the first aircraft from both intermediate zone and the landing sequence. If final zone is empty in the pre state, then the condition follows from the following reason. This assumption and the induction hypothesis implies that the first aircraft of intermediate is also the first aircraft of c. Thus the transition removes the first aircraft of both sequences, and therefore the correspondence is preserved by the transition. If final is not empty in the pre state, from the close correspondence stated in the second condition, the first aircraft of final is the first aircraft of the landing sequence in the pre state. Whereas, from the precondition of the transition, the first aircraft of intermediate is also the first aircraft of the landing sequence. However, this contradicts the uniqueness of the aircraft in the logical zones since the same aircraft is both on final zone and intermediate zone.

- *In the case of the transitions for an entry of aircraft (*VerticalEntry *and* LateralEntry*)*:
  The transition adds an aircraft to one of the initiation areas and the end of the landing sequence. Thus $|c|$ does not change. From Lemma 3.3, $|c| \leq |$landing_seq$|$. And since $|c|$ is not changed, the portion of the landing sequence that we have to prove the correspondence is the same portion as in the pre state. Thus the correspondence we have to show immediately follows from the induction hypothesis since the newly entering aircraft is added to a position of the landing sequence out of the range where the correspondence has to hold.

The rest of the transitions do not affect either the landing sequence or zone final or intermediate. Thus the condition immediately follows from the induction hypothesis. [End of the proof of the second condition.]

Third Condition: Now we prove the third condition of the claim, that is, the condition for Lemma 3.6.

- *In the case of the transitions for an entry of aircraft (*VerticalEntry *and* LateralEntry*)*:
  The condition for aircraft other than the newly entering aircraft follows from the induction hypothesis. For the newly entering aircraft, it is added to both the operation area and the landing sequence. Thus the condition holds.

- *In the case of* Exit *and* Landing:
  The transition removes the first aircraft of intermediate in the case of Exit, and removes the first aircraft of final in the case of Landing, and also removes the first aircraft of the landing sequence in both cases. In the case of the Exit action, the precondition of the action guarantees that the first aircraft of intermediate is the first aircraft of the landing sequence. In the case of the Landing, the induction hypothesis implies that $c[0] =$ landing_seq$[0]$. Since final is not empty from the precondition of Landing, $c[0] =$ final$[0]$. Thus, the first aircraft of final is the first aircraft of the landing sequence. Hence, in either case, the transition removes the same aircraft from both the operation area and the landing sequence. For aircraft that is not removed by the transition, the condition holds from the induction hypothesis. For the removed aircraft, we have to show that the aircraft is no longer in the operation area. This holds from the uniqueness of aircraft on the entire logical zones stated in Corollary 3.2.

- *In the case of* MissedApproach:
  From a discussion analogous to the case of Landing, it follows that the first aircraft of final

is same as the first aircraft of the landing sequence in the pre state. Thus the transition removes the same aircraft from both the operation area and the landing sequence. In addition, the transition adds the same reassigned aircraft to both the operation area and the landing sequence. To prove that the removed aircraft is no longer in the operation area, we can use Lemma 3.1

The rest of the transitions do not add or remove aircraft from either the operation area or the landing sequence. Thus the condition immediately follows from the induction hypothesis [End of the proof of the third condition.] □

As a corollary of Lemma 3.7 (Invariant 3.5), the following invariant holds.

Corollary 3.8. *For any reachable state of the abstract model, if* `final` *zone is not empty, then the first aircraft of* `final` *is the same as the first aircraft in the landing sequence.*

Corollary 3.8 is stated in PVS as follows.

```
first_final_lemma: LEMMA
   (FORALL (s:states):
      reachable(s) =>
        (NOT empty?(final(s)) IMPLIES
                 first(final(s)) = first(landing_seq(s))))
```

From this corollary, we can guarantee that the model is actually handling the same aircraft in the operational area and the landing sequence in the case of `Landing` and `MissedApproach` transitions. Recall that the precondition of these actions does not in itself ensure that the aircraft that the transition handles in the operation area is the same as the aircraft it handles in the landing sequence.

The uniqueness of aircraft in the landing sequence is defined analogously to the uniqueness of ID in the landing sequence. As in the case of the uniqueness of ID and aircraft in the operation area, the uniqueness of aircraft in the landing sequence follows from the uniqueness of ID of aircraft in the landing sequence.

Corollary 3.9. (Uniqueness of aircraft on the landing sequence) *For any reachable state of the abstract model, the following condition holds:*

For any natural numbers i and j where $i < j < |\mathsf{landing\_seq}|$, $\mathsf{landing\_seq}[i] \neq \mathsf{landing\_seq}[j]$

Corollary 3.9 is stated in PVS as follows.

```
queue_uniqueness(q:queue): bool =
   (FORALL (i,j: nat):
      i < length(q) and j < length(q) and i<j IMPLIES
        nth(q,i) /= nth(q,j))
landing_seq_uniqueness_lemma: LEMMA
   (FORALL (s:states):
    reachable(s) => queue_uniqueness(landing_seq(s)))
```

Correspondence of the number of aircraft assigned to one side as their mahf between the operation area and the landing sequence

The next correspondence we prove concerns the number of aircraft assigned a particular side as their mahf. It states that, for side σ, the number of aircraft a with $a.\mathtt{mahf} = \sigma$ in the operation area coincides with the number of aircraft a with $a.\mathtt{mahf} = \sigma$ in the landing sequence. By using this lemma, we can prove a upper bound on the number of mahf assignments to one side in the operation area by proving a corresponding upper bound on the number of mahf assignments to the same side in the landing sequence.

Lemma 3.10. *For any reachable state of the abstract model and side σ, $assigned(\mathsf{landing\_seq}, \sigma) = assigned2fix(\sigma)$.*

Lemma 3.10 is stated in PVS as follows.

```
assigned_seq(seq:queue, side:Side): RECURSIVE nat =
   IF empty?(seq) THEN 0
   ELSIF mahf(first(seq)) = side THEN 1+assigned_seq(rest(seq),side)
   ELSE assigned_seq(rest(seq),side)
   ENDIF
   MEASURE length(seq)
assigned_ac_coincides_lemma: LEMMA
   (FORALL (s:states, side:Side):
      reachable(s) =>
          (assigned2fix(s,side) = assigned_seq(landing_seq(s),side)))
```

We can prove this lemma analogously to Lemma 3.3, except that, unlike Lemma 3.3, we cannot just count the number of aircraft added or removed, since an aircraft added to (removed from) the operation area might potentially have a different mahf assignment from the assignment of the aircraft added to (removed from) the landing sequence by the same transition. We have to guarantee that when a transition adds (removes) an aircraft to (from) the operation area, it adds (removes) the same aircraft to (from) the landing sequence.

Proof. For the transitions for an aircraft entry, we can assure from the definition of the transitions that the same aircraft is added to both. In the case of `Exit` transition, the precondition ensures that the first aircraft of `final` is same as the first aircraft of the landing sequence, and the model removes this aircraft from both the operation area and the sequence. Thus the condition holds. In the case of `Landing`, Corollary 3.8 ensures that the first aircraft of the zone `final` is also the first aircraft in the sequence. Thus the model removes the same aircraft from both. The case of `MissedApproach` can be proved analogously using Corollary 3.8 in the same way. □

3.3.2 Key Invariants

The basic properties proved in Section 3.3.1 guarantees that the model constructed in this chapter behaves as intended; for example, there is no duplication of aircraft (which is guaranteed from the uniqueness of aircraft). In contrast, two invariants in this subsection state key properties of the model for why the system works correctly, rather than if the model behaves as intended.

The first invariant (Lemma 3.11) states that the local flipping rule of `nextmahf` – that the model flips the value of `nextmahf` each time it assigns a mahf to an aircraft – indeed constructs the alternating assignments of the mahf of aircraft in the landing sequence.

The second invariant (Lemma 3.13) states that the local order-preserving rule of the approach initiation – an aircraft cannot initiate the approach until its leader does so – can be extended to a similar property that refers to any two aircraft in the landing sequence, by using the notion that an aircraft *precedes* another aircraft, defined in Section 2.3.3.

These two invariants are useful in that they enable us to directly discuss any two aircraft in the landing sequence, not just an aircraft and its leader.

The first invariant is as follows.

Lemma 3.11. (Alternating assignment of right and left to mahf of aircraft) *For any reachable state of the abstract model, if the landing sequence is not empty, then the following two conditions hold: Let* $l = |\mathsf{landing\_seq}|$*, and* $m = a.$`mahf` *where* a *is the first aircraft in the landing sequence.*

(i). *For any natural number* $i < l$*,* $\mathsf{landing\_seq}[i].\mathsf{mahf}$ *is* m *if* i *is even, otherwise it is* $opposite(m)$*.*

(ii). *the value of* `nextmahf` *is* m *if* l *is even, otherwise it is* $opposite(m)$*.*

This lemma is stated in PVS as follows.[1]

[1] Note that, in PVS, we proved the condition on the number of assignments implied by the alternating assignment together with the above condition. Thus we have a conjunction of three conditions, as opposed to two, in which the first condition represents the condition on the number of assignments, and the remaining two conditions represents the two corresponding conditions in the above lemma. This additional condition is defined as a corollary of this lemma (Corollary 3.12). We split the conditions of the lemma in PVS into one lemma and one corollary to improve the readability of the conditions and a proof of them

```
alternate_assignment_lemma: LEMMA
   (FORALL (s:states, side:Side):
     reachable(s) =>
      NOT empty?(landing_seq(s)) IMPLIES
       LET first_mahf = first(landing_seq(s))`mahf IN
       LET length_seq = length(landing_seq(s)) IN
       (assigned_seq(landing_seq(s),side) =
               IF even?(length_seq) THEN length_seq/2
               ELSIF first_mahf = side
                    THEN (length_seq+1)/2
                    ELSE (length_seq-1)/2  ENDIF                  ) AND
       (FORALL (i:nat): i<length(landing_seq(s)) IMPLIES
         nth(landing_seq(s),i)`mahf =
             IF even?(i)          THEN first_mahf
                                  ELSE opposite(first_mahf) ENDIF ) AND
       (s`nextmahf =
             IF even?(length_seq) THEN first_mahf
                                  ELSE opposite(first_mahf) ENDIF ))
```

Recall that a zone queue is indexed starting from zero, and thus the first condition holds even in the case $i = 0$, in which i-th positioned aircraft is the first aircraft.

The first condition states the main statement of the alternating assignment property in the landing sequence. We have to prove the second condition (the condition derived from the local flipping rule of `nextmahf`) together to make an inductive proof work.

Proof. By induction over steps of the model. The landing sequence is empty in the initial states, thus the condition trivially holds for the base case. Now we focus on the induction step.

- *In the case of transitions for an entry of aircraft (*`VerticalEntry` *and* LateralEntry*)*:

  The first condition of the invariant: For an aircraft that is already in the landing sequence before the transition, the condition holds from the induction hypothesis since the transition just adds a new aircraft to the end of the sequence. For the newly entering aircraft, the mahf assignment of it is determined according to the value of `nextmahf` in the pre state. Suppose l is even in the pre state. It implies the new aircraft is assigned the same side as m, and this aircraft become the l-th positioned aircraft in the sequence in the post state (recall that the position starts from zero, and thus the last aircraft in the sequence of length $l + 1$ is in position l.) Since l is even from our assumption, the condition holds for the new aircraft in the post state. We can prove the case where l is odd analogously.

  The second condition of the invariant: The value of `nextmahf` is flipped by the transition. Since the parity of the length of the sequence changes as well, the condition holds.

- *In the case of the actions that removes the first aircraft from the landing sequence (*`Exit` *and* `Landing`*)*:

  The first condition of the invariant: The second aircraft in the landing sequence (landing_seq[1]) in the pre state becomes the first aircraft in the post state. From the induction hypothesis, $\mathsf{landing\_seq}[1].\mathsf{mahf} = \mathit{opposite}(\mathsf{landing\_seq}[0].\mathsf{mahf})$ in the pre state. At the same time, the parity of the position of each aircraft in the sequence changes by the transition. Hence the condition is preserved by the transition.

The second condition of the invariant: As we mentioned in the first case, the mahf of the first aircraft is flipped by the transition, whereas the parity of the length of the sequence changes. Thus the condition holds from the induction hypothesis.

- *In the case of* `MissedApproach`:

  The first condition of the invariant: The transition removes the first aircraft (let us call it a_0) from the sequence and adds a_0 to the end of the sequence with a reassignment of its mahf. For an aircraft other than a_0, the parity of its position changes by the transition since the first aircraft is removed. At the same time, the mahf of the first aircraft is flipped. Thus the condition is preserved by the transition. Now we consider the case for a_0. Since a_0 is added to $(l-1)$-th position (the end of the sequence of length l), the parity of the position of a_0 is odd from the assumption that l is even. The mahf of a_0 is reassigned according to the value of `nextmahf` in the pre state. Now suppose l is even. From the second condition that holds in the pre state, it implies that by the transition, a_0 gets assigned the same mahf as the first aircraft in the pre state, which is opposite to the mahf of the first aircraft in the post state. This is what is required to verify the condition since the position of a_0 in the landing sequence is odd in the post state. We can prove the case that l is odd analogously.

  The second condition of the invariant: The length of the sequence does not change by the transition. The value of `nextmahf` is flipped by the transition. At the same time, the mahf of the first aircraft is flipped as we stated above for the first condition. Thus the condition still holds in the post state.

The rest of the transitions do not affect either the landing sequence or the mahf assignment to aircraft. Hence the condition immediately holds from the induction hypothesis. □

As a corollary of Lemma 3.11, we can prove the following.

Corollary 3.12. *Let* $l = |\mathsf{landing\_seq}|$ *, and* a *be the first aircraft in the landing sequence. For any reachable state of* $SATS$*, the following conditions hold:*

(i). *If* l *is even,* $\mathit{assigned}(\mathsf{landing\_seq}, \sigma) = \frac{l}{2}$.

(ii). *If* l *is odd,* $\mathit{assigned}(\mathsf{landing\_seq}, \sigma)$ *is equal to* $\frac{l+1}{2}$ *if* $\sigma = a.\mathtt{mahf}$*, and it is* $\frac{l-1}{2}$*, otherwise.*

This corollary restates the alternating assignment property stated in Lemma 3.11 in terms of the number of aircraft assigned to one side in the landing sequence. Since the assignments to aircraft in the landing sequence alternate, if the length of the sequence is even, then the number of aircraft with the mahf assignment to one specific side is exactly half of the length. If the length of the sequence is odd, then the number is, of course, not exactly half, and depends on the mahf assignment of the first aircraft.

The next lemma states the order-preserving rule for the approach initiations of aircraft. As we mentioned earlier in this subsection, there is a local order-preserving rule between an aircraft and its immediate leader for approach initiation: For `VerticalApproachInitiation` and `LateralApproachInitiation` to be enabled, the leader of the aircraft initiating the approach by these transitions must have already initiated the approach (that is, must be on the approach area). The lemma extends this order-preserving rule between an aircraft and its leader to any two aircraft in the landing sequence using the notion of *precedes* defined in Section 2.3.

Lemma 3.13. (The order-preserving property between the order of the approach initiations and the landing sequence) *For any reachable state of $SATS$, the following condition holds:*

> *For any aircraft a and b in the landing sequence, if b precedes a in the sequence, and a is on the approach, then b is also on the approach.*

Lemma 3.13 is stated in PVS as follows.

```
order_preserve_lemma: LEMMA
  (FORALL (s:states, a,b:Aircraft):
    reachable(s) =>
    (precedes?(b,a,landing_seq(s)) AND on_approach?(s,a) IMPLIES
     on_approach?(s,b)))
```

Proof. By induction. There is no aircraft in the landing sequence in the initial state, thus the condition trivially holds for the base case. Now we prove the inductive step. Since an aircraft does not precede the same aircraft, we only consider the case a and b are different aircraft.

- *In the case of the transitions for an entry of aircraft (*`VerticalEntry` *and* LateralEntry*):* If neither a nor b is the newly entering aircraft, the condition holds because of the following reason. b precedes a in the pre state Suppose b precedes a, and a is on the approach in the post state. Since the transition does not affect the preceding relation between b and a, or the zone where a is, above two conditions in our assumption also hold in the pre state. It implies that b is on the approach area in the pre state from the induction hypothesis. Because b does not move by the transition, b is on the approach in the post state as required.

Suppose a is the newly entering aircraft. From the uniqueness of aircraft on logical zones stated in Corollary 3.2, a is only on either `holding3` or `lez`, depending on whether the action is `VerticalEntry` or `LateralEntry`. In either case, a is not on the approach, and therefore the condition vacuously holds.

If b is the newly entering aircraft, b is added to the end of the landing sequence. From the uniqueness of aircraft in the landing sequence stated in Corollary 3.9, b cannot precede a. Hence the condition vacuously holds.

- *In the case of* `VerticalApproachInitiation`:

  The transition moves the first aircraft in `holding2` to the approach area (`base` zone). Since the transition does not affect the landing sequence, the preceding relation of any two aircraft in the landing sequence does not change by the transition. Suppose that b precedes a and a is on the approach in the post state.

  We split the case depending on a as follows. If a is not the aircraft initiating the approach (that is, moving by the transition), a must be already on the approach in the pre state. Considering that the preceding relation between a and b does not change by the transition, b is also on the approach in the pre state from the induction hypothesis. Since b does not move by the transition, and thus is still on the approach in the post state, the condition holds.

  Now we consider the case that a is the aircraft initiating the approach (that is, moving by the transition). The precondition of the transition guarantees that in the pre state, either a is the first aircraft in the landing sequence, or the leader of a is already on the approach area. Considering the uniqueness of aircraft in the landing sequence from Corollary 3.9, a cannot be the first aircraft of the landing sequence since it is preceded by b. Again from the uniqueness of aircraft in the landing sequence, the fact that b precedes a implies either $b = leader(a)$, or b precedes $leader(a)$. In the former case, b has to be on the approach from the precondition of the action. Hence the condition holds. In the latter case, as $leader(a)$ is already on the approach in the pre state from the precondition of the transition, we can apply the induction hypothesis to b and $leader(a)$. It follows that b is on the approach in the pre state, and b stays there in the post state. Therefore the condition holds.

- *In the case of* `LateralApproachInitiation`:

  The transition moves an aircraft to either `base` or `holding2` depending on the situation of the current state. If the first aircraft of `lez` is the first aircraft in the landing sequence, or the leader of it is already on the approach (this condition is equivalent to the precondition of `VerticalApproachInitiation`), it directly initiates the approach from `lez`, and thus goes to the approach area. If the above condition is not satisfied as the moment of the transition, the first aircraft of `lez` moves to `holding2`. In the former case, we can have the similar discussion to the case of `VerticalApproachInitiation`. In the latter case, the transition does not affect either the landing sequence or the approach area. Thus the condition immediately follows from the induction hypothesis.

- *In the case of* `Exit` *and* `Landing`:

  The transition removes the first aircraft of the landing sequence. Suppose that b precedes a and a is on the approach area in the post state. Neither a nor b is the first aircraft of the sequence in the pre state, since if so, that aircraft is not in the sequence in the post state from the uniqueness of aircraft, and thus b does not precede a. Hence both a and b are not removed from the landing sequence. Therefore the position of them in the logical zones and the preceding relation between them do not change by the transition. Hence the condition follows from the induction hypothesis.

- *In the case of* `MissedApproach`:
  The aircraft that has missed the approach goes from `final` to `maz` with a new mahf assignment. The same aircraft is also removed from the head of the landing sequence, and is added to the end of the sequence with a new mahf assignment.

  If a is the aircraft that has missed the approach, the uniqueness of aircraft implies that a is not on the approach area in the post state since it is in `maz`. Thus the condition vacuously holds.

  If b is the aircraft that has missed the approach, b is added to the end of the landing sequence. From the uniqueness of aircraft in the landing sequence, b just appear at the end of the landing sequence, and thus does not precede a. Thus the condition holds.

  Now we consider the case in which neither a nor b is the aircraft that has missed the approach. In such a case, the position of a and b in the logical zones and the preceding relation between them do not change by the transition. Thus the condition follows from the induction hypothesis.

The rest of the transitions do not affect either the landing sequence or the approach area. Thus the condition holds from the induction hypothesis. □

From Lemma 3.13, we can guarantee that every aircraft has to preserve the order of the landing sequence when it initiates the approach. To see the reason, consider the following argument. Suppose, for a contradiction, that a violates the order of the landing sequence and initiates the approach. This implies that there is an aircraft b that precedes a but is still in an initiation area. However, from Lemma 3.13, b must also be on the approach. This contradicts the uniqueness of aircraft in the operation area.

3.4 Proving the main properties, Part 1: Properties that can be proved by straightforward induction

Now we start proving the main properties presented in Section 3.2. Some properties can be easily proved by induction using the auxiliary invariants we have proved in Section 3.3. The other properties need to be strengthened using a notion of *blocking* of aircraft in order to make an inductive proof go through.

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 depends only on properties that have been proved. Because of this, the order of proofs in this section does not match the numbering of the properties.[2]

In this section, we start by proving properties that can be proved straightforwardly by induction (Properties 1, 7, and 5). Then, we move on to arguments about a notion of "blocking of aircraft". This notion of blocking is formally introduced in Section 3.5. In Section 3.6, we strengthen Property 6 using a blocking condition in order to prove it inductively. How we strengthen this property and how we prove it gives a good introduction to a more complicated lemma and an argument in Section 3.7. In Section 3.7, we define one key lemma, Lemma 3.26, to prove the rest of the properties. This lemma consists of the conjunction of two of the main properties, Property 3 and 4, and seven conditions each of which represents a different blocking situation. The last property to be proved, Property 2, also follows from this key lemma. A proof for this lemma turns out to be the longest and most complicated in this thesis, mainly due to a substantial number of case analyses and discussions on blocking situations.

Here we start by proving Property 1.

Theorem 3.14. (Property 1) *For any reachable state of $SATS$, $arrival\_op \leq 4$.*

Proof. By induction. In the initial state, there is no aircraft on the operation area, and thus $arrival\_op = 0$. Hence the condition trivially holds for the base case. Now we start proving the

[2] Since we did not know what the order of proofs should be when we define these properties in PVS, we just listed the properties in the order as appear in this thesis. Though we could have re-numbered the properties to match the order of the proof, in order to maintain the consistency with PVS code, we numbered them in the same order as code.

induction step.

- *In the case of* `VerticalEntry(`σ`)`:

  The precondition of the transition guarantees that $virtual(\sigma) < 2$.

  We first claim that from the definition of *virtual*, $virtual(\sigma) \geq assigned2fix(\sigma)$ for any state from the following reason. The value of $assigned2fix$ can be calculated by the sum of the number of aircraft a on σ such that $a.\mathsf{mahf} = \sigma$, plus the number of aircraft b such that $b.\mathsf{mahf} = \sigma$ and b is not on σ. On the other hand, the value of *virtual* can be calculated by the sum of the number of aircraft on σ, plus the number of aircraft b such that $b.\mathsf{mahf} = \sigma$ and b is not on σ. Notice that the second number in both sums are exactly the same. In addition, the first number in the sum of $assigned2fix$ is always less than or equal to the first number in the sum of *virtual*. Thus the above claim holds.

  It follows from the above claim that $assigned2fix(\sigma) < 2$.

  From the induction hypothesis, $arrival\_op \leq 4$ in the pre state. Now we prove the condition by splitting it into two cases. If $arrival\_op < 4$ in the pre state, the condition holds since the transition just adds one aircraft to the operation area. If $arrival\_op = 4$ in the pre state, then it follows from Corollary 3.12 that $assigned2fix(\sigma) = 2$. This contradicts with the above discussion that
  $assigned2fix(\sigma) < 2$.

- *In the case of* `LateralEntry(`σ`)`:

  The precondition of the action guarantees that $virtual(\sigma) = 0$. Thus, it, of course, follows that $virtual(\sigma) < 2$. Thus we can use the same argument as in the case of `VerticalEntry` to prove the condition.

The rest of the transitions do not add an aircraft to the operation area, and thus do not increase *arrival_op*. Hence the condition follows from the induction hypothesis. □

Theorem 3.15. (Property 7) *For any reachable state of* $SATS$ *and side* σ*,*
$assigned2fix(\sigma) \leq 2$.

Proof. The condition immediately follows from Theorem 3.14 and Corollary 3.12: from Theorem 3.14, the number of aircraft on the operation area (*arrival_op*, and thus $|\mathsf{landing\_seq}|$) is at most four, and from Corollary 3.12, the number of aircraft assigned one side
$assigned2fix(\sigma) = 2$ when $arrival\_op = 4$, and $assigned2fix(\sigma) \leq 2$ when $arrival\_op = 3$. If $arrival\_op \leq 2$, the condition is trivial since $assigned2fix(\sigma) \leq arrival\_op$ (the number of aircraft assigned some side cannot exceed the total number of aircraft there). □

Theorem 3.16. (Property 5) *For any reachable state of $SATS$ and side σ, $|\mathsf{lez}(\sigma)| \leq 1$.*

Proof. By induction. There is no aircraft in the `lez` zones in the initial state, thus the condition holds for the base case. We prove the induction step in the following.

- *In the case of* `LateralEntry(`σ`)`:
  The precondition of the transition ensures that $virtual(\sigma) = 0$ in the pre state of the transition. This implies that $actual(\sigma) = 0$. Thus `lez(`σ`)` is empty in the pre state. Since the transition adds just one aircraft to `lez(`σ`)`, there is exactly one aircraft in the post state. Hence the condition holds.

The rest of the transitions do not add an aircraft to `lez` zone. Thus the condition follows from the induction hypothesis. □

3.5 Blocking of aircraft

As we discussed in Section 3.2, we need a notion of *blocking* of aircraft to prove the remaining properties. Recall that, as discussed in Section 3.2, the problem we are concerned about arises from the fact that, once an aircraft initiates the approach, there is no way to prevent the aircraft from missing the approach, and in consequence, it goes to a missed approach zone (`maz`). For instance, suppose an aircraft a with $a.\mathsf{mahf} = \sigma$ initiates the approach from a state in which two aircraft are in `maz(`σ`)`. Since there is no “guard” that prevents `MissedApproach` from being performed, the aircraft can possibly go to `maz(`σ`)`. This scenario would violate Property 3 – $|\mathtt{maz}(\sigma)| \leq 2$. This implies that in order to prove an upper bound on $|\mathtt{maz}(\sigma)|$, we have to guarantee that aircraft that will jeopardize the bound would not initiate the approach.

A key to guaranteeing that such an aircraft would not initiate the approach is a fact that follows from Lemma 3.13: an aircraft has to preserve the order of the landing sequence when it initiates the approach. That is, if aircraft a is preceded by another aircraft b, a has to wait until b initiates the approach before a does so. Thus, this condition works as an implicit “guard” that delays the approach initiation of aircraft that could potentially violate the required bounds. Using this guard, we can assert the following fact: if, for every aircraft a on side σ, $a.\mathsf{mahf} = \sigma$ or a is preceded by some other aircraft b in the landing sequence, then no aircraft on σ would go to $\mathtt{maz}(opposite(\sigma))$ until b initiates its approach.

We define a notion of “blocking” of aircraft as follows, in order to formally capture the above mentioned situation – for every aircraft a on side σ, $a.\mathsf{mahf} = \sigma$ or a is preceded by some other aircraft b in the landing sequence.

First we define a blocking condition for one single aircraft.

Definition 3.17. Let a and b be aircraft, σ be a side. We say that a is (b, σ)*-blocked* if and only if, $a.\mathsf{mahf} = \sigma$ or a is preceded by b in the landing sequence.

If a is (b, σ)-blocked, then a would not go to $\mathtt{maz}(\mathit{opposite}(\sigma))$ until b initiates the approach. Now we define a blocking condition for one whole side.

Definition 3.18. Let b be an aircraft, σ be a side. We say that the side σ is b*-blocked* if and only if, for every aircraft a in σ, a is (b, σ)-blocked. We refer to this aircraft b as a *blocking aircraft* for side σ.

From the above discussion about the guard for the approach initiation, if side σ is b-blocked, then no aircraft in that side would reach $\mathtt{maz}(\mathit{opposite}(\sigma))$ until the aircraft b initiates its approach. In other words, the side σ cannot "send" any aircraft to $\mathit{opposite}(\sigma)$.

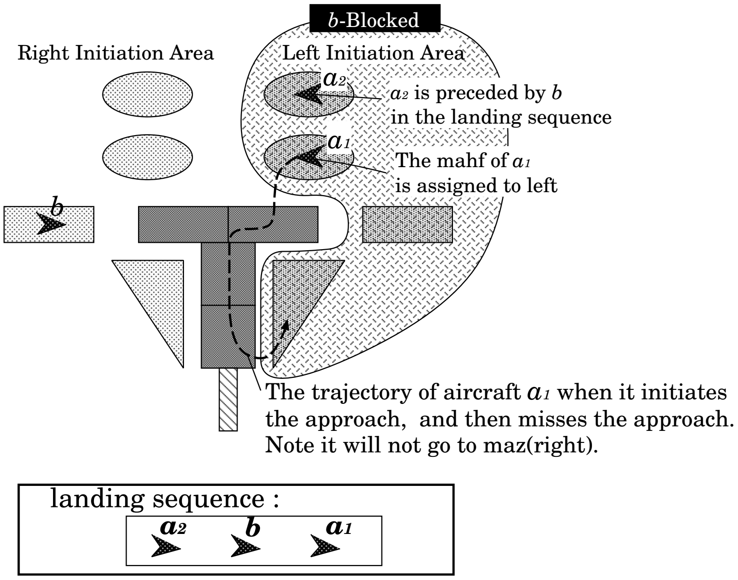
The blocking condition is defined in PVS as follows. The predicate `blocked_by?(a, b, side)` checks if aircraft `a` is (`b`, `side`)-blocked. Using `blocked_by?`, the predicate `blocked_side?` checks whether `side` is `b`-blocked. For an easier understanding of the blocking notion, we defined this notion in terms of σ (and `side` in PVS). However, the actual predicate used to strengthen main properties is the last predicate `blocked_opposite_side?`, defined as `blocked_side?` for `opposite(side)`. Indeed, as discussed in the example in the beginning of this subsection, when we want to strengthen Property 3 $(\forall \sigma : \mathit{Side},\ |\mathtt{maz}(\sigma)| \leq 2)$ for a specific side σ, we need $\mathit{opposite}(\sigma)$ to be blocked, so that an aircraft would never come from $\mathit{opposite}(\sigma)$ to $\mathtt{maz}(\sigma)$.

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

An example of a blocking situation is depicted in Figure 3.2. In this situation, the left side is b-blocked, since every aircraft in the left side is (b,left)-blocked, that is, each aircraft in the left side is either assigned `left` as its mahf, or is preceded by b. Until b initiates the approach, no aircraft in the left side would reach `maz(right)`.

As we will see in the explanation of Lemma 3.26, we also need to consider a slightly different blocking situation than b-blocked. More specifically, we need to define the conditions that capture a situation in which *exactly one* aircraft c with $c.mahf = \sigma$ *can* initiate the approach, but *any other* aircraft have to be blocked.

First we define a notion that an aircraft a is σ*-ready*, for some side σ.



b-Blocked
Right Initiation Area
Left Initiation Area
a_2
a_2 is preceded by b in the landing sequence
a_1
The mahf of a_1 is assigned to left
b
The trajectory of aircraft a_1 when it initiates the approach, and then misses the approach. Note it will not go to maz(right).
landing sequence :
a_2
b
a_1

Figure 3.2: The left side is b-blocked, where b is the first aircraft of `lez(right)`

Definition 3.19. Let σ be a side. We say that aircraft a is *σ-ready* if and only if the following conditions hold

1. a is in *opposite*(σ).

2. a.mahf $= \sigma$.

3. For any aircraft b in σ, a precedes b in the landing sequence.

If aircraft a is σ-ready, then a's approach initiation is not blocked by any aircraft in σ, and thus a can possibly go to `maz(`σ`)` in case it misses the approach.

Example: We now explain an example of σ-ready. In the state depicted in Figure 3.3 (re-depicted from Figure 3.1), aircraft c is *not* right-ready, since it is preceded by b in the right side. However, after b initiates the approach by following a few steps from that state, c is no longer preceded by any aircraft on the right side. Thus in this new state, a *is* right-ready.

The condition that checks whether there is a `side`-ready aircraft in state `s` is defined in PVS as follows.

```
ac_ready_to_approach?(side:Side, s:states): bool =
      (EXISTS (a:Aircraft):
         mahf(a)=side AND
         on?(opposite(side),a,s) AND
         (FORALL (b:Aircraft):
            on?(side,b,s) IMPLIES precedes?(a,b,landing_seq(s))))
```

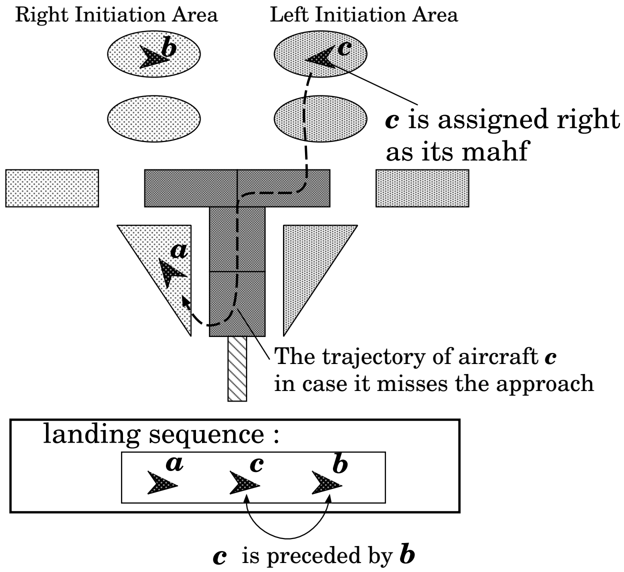

Right Initiation Area
Left Initiation Area
b
c
c is assigned right as its mahf
a
The trajectory of aircraft c in case it misses the approach
landing sequence :
a
c
b
c is preceded by b

Figure 3.3: c is not ready to go to the opposite side until b initiates the approach

Now we define a notion of a blocking except for one aircraft.

Definition 3.20. Let b be an aircraft, σ be a side. We say that the side σ is *b-blocked^{-1}* if and only if there is an aircraft c such that c is in side σ and $c.\mathsf{mahf} = \mathit{opposite}(\sigma)$, and for every aircraft a on σ such that $a \neq c$, a is (b, σ)-blocked.

The condition for b-blocked^{-1} states that except for one aircraft c, the same blocking condition as for b-blocked applies to aircraft in side σ.

This notion is defined in PVS as `blocked_side_minus_1?` as follows. Analogous to `blocked_opposite_side?` for `blocked_side?`, we define `blocked_except_for_one?` as the "opposite side" version of `blocked_side_minus_1?`.[3]

```
blocked_side_minus_1?(b:Aircraft, side:Side, s:states):bool =
   EXISTS (c:Aircraft):
      on?(side,c,s) AND mahf(c) = opposite(side) AND
      FORALL (a:Aircraft):
        on?(side,a,s) AND a /= c IMPLIES blocked_by?(a,b,side,s)
blocked_except_for_one?(b:Aircraft, side:Side, s:states):bool =
   EXISTS (c:Aircraft):
      on?(opposite(side),c,s) AND mahf(c) = side AND
      FORALL (a:Aircraft):
        on?(opposite(side),a,s) AND a /= c IMPLIES
                                  blocked_by?(a,b,opposite(side),s)
```

[3] This `blocked_except_for_one?` is defined in a direct way, as opposed to using `blocked_side_minus_1?`. This is because of a minor technical reason: if we define `blocked_except_for_one?(b,side,s) = blocked_side_minus_1?(b,opposite(side),s)`, then the predicate will have the term `mahf(c) = opposite(opposite(side))` when expanded. Though, of course, we can assert that `opposite(opposite(side)) = side`, in order to ease a mechanical theorem-proving process as much as possible, we chose to define `blocked_except_for_one?` in a expanded form.

The blocking conditions and σ-ready condition presented in this subsection are preserved by some transitions under some conditions of the state in which the transitions are performed. The following Lemmas 3.21, 3.22, and 3.23 state such conditions of the transitions and the states.

Lemma 3.21 states the condition for `blocked_opposite_side?`. Recall that, as we discussed earlier in this subsection, we will use a blocking condition to strengthen main properties, and in doing so for side σ, we have to use the claim *opposite*(σ) is b-blocked. This is why the lemma is defined in terms of `blocked_opposite_side?`, as opposed to `blocked_side?`.

Lemma 3.21. *For any reachable state s of $SATS$, any side σ, and any aircraft b in the operation area, if the side opposite(σ) is b-blocked, then the same side is b-blocked after a transition in any of the following cases.*

1. *The transition is either* `HoldingPatternDescend`*,* `Merging`*,* `FinalSegment`*,* `Taxiing`*, or* `LowestAvailableAltitude`*.*
2. *The transition is for the entry of an aircraft –* `VerticalEntry` *or* `LateralEntry`*.*
3. *The transition is for the approach initiation – * `VerticalApproachInitiation` *or* `LateralApproachInitiation`*.*
4. *The transition is either* `Exit`*,* `Landing`*, or* `MissedApproach`*, and b is on some side.*

Lemma 3.21 is stated as two lemmas in PVS as follows. We split the lemma into two lemmas in PVS, since for the first condition in Lemma 3.21, we have the exact equality for the value of `blocked_opposite_side?`, whereas, we only have an implication for the remaining conditions. The first lemma `blocked_opposite_side?_unchanged` states the first condition of Lemma 3.21, and the second lemma `blocked_opposite_side?_implication` states the remaining cases.

```
blocked_opposite_side?_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
         (HoldingPatternDescend?(a)    OR
          Merging?(a)                  OR
          FinalSegment?(a)             OR
          Taxiing?(a)                  OR
          LowestAvailableAltitude?(a)     ) AND
          enabled(a,s)
            IMPLIES
         blocked_opposite_side?(ac,side,trans(a,s)) =
         blocked_opposite_side?(ac,side,s))
```

```
blocked_opposite_side?_implication: LEMMA
    (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
        (VerticalEntry?(a)                     OR
          LateralEntry?(a)                      OR
          VerticalApproachInitiation?(a)        OR
          LateralApproachInitiation?(a)         OR
          ((Exit?(a) OR Landing?(a) OR MissedApproach?(a)) AND
           (on?(side,ac,s)))) AND
          on_zones?(s,ac) AND
          enabled(a,s) AND
          reachable(s)
            IMPLIES
        (blocked_opposite_side?(ac,side,s) =>
         blocked_opposite_side?(ac,side,trans(a,s))))
```

Proof. Case 1: The transitions in this group does not affect the set of aircraft in the initiation areas or the set of aircraft in the landing sequence, nor the mahf assignment of aircraft. Since all conditions that are related to a blocking argument are above three things, the transition preserves a blocking by b.

Case 2: If a new aircraft enters to *opposite*(σ), the transition does not change the set of the aircraft in σ, and all of them are still either assigned σ as their mahf, or are preceded by b. Thus the condition holds. Now consider the case that an aircraft enters to σ. For the aircraft in σ except for the new aircraft, the blocking condition holds since the transition does not affect their mahf assignments or the preceding relation between them and the blocking aircraft b. For the new aircraft, since it is added to the end of the landing sequence, it is preceded by any aircraft that is already in the landing sequence. From Lemma 3.6, the fact that b is in the operation area implies that it is also in the landing sequence. Thus the new aircraft is preceded by b. Therefore the blocking condition holds for all aircraft in the initiation area of γ.

Case 3: The transition moves an aircraft from one of the initiation areas to the approach area. If an aircraft moves from *opposite*(σ), the condition holds since the transition does not change the set of the aircraft in σ, and all of them are still assigned σ as their mahf, or are preceded by b. If an aircraft moves from σ, the transition just removes one aircraft from σ. Thus all remaining aircraft are still assigned σ as their mahf or are preceded by b. (Note that since we do not have the condition that a blocking aircraft must be in the initiation area, even if b initiates the approach by this action, the condition still holds.)

Case 4: If the transition is either `Exit` or `Landing`, it removes both the first aircraft of the landing sequence and the first aircraft of either `intermediate`, in the case of `Exit`, or `final`, in the case of `Landing`. In either case, the blocking aircraft b would not be removed from the model since we assume that it is in some side, on in the approach area. Considering that removing the first aircraft in the landing sequence does not affect the preceding relation of the remaining aircraft, nor change the mahf assignment of any aircraft, the initiation area of σ is still b-blocked after the transition.

If the transition is `MissedApproach`, it removes both the first aircraft of the landing sequence, and the first aircraft a of `final`, and adds the removed aircraft a, with reassignment of the mahf, to the end of the landing sequence and `maz`(a.`mahf`). The condition for all aircraft in σ except for a follows from the same argument as in the case of `Exit` and `Landing`. For the reassigned aircraft a, since it is added to the end of the landing sequence, it is preceded by any aircraft that has already been in the landing sequence. Thus, from an analogous argument as in the case of `VerticalEntry` or `LateralEntry`, it is preceded by b in the post state of the transition. Hence the blocking condition holds for all aircraft in the initiation area of σ. □

Next we prove a claim analogous to Lemma 3.21 for b-blocked^{-1}.

Lemma 3.22. *For any reachable state of SATS, side σ, and aircraft b in the operation area, if opposite(σ) is b-blocked^{-1}, and b is in σ, then the same side is b-blocked^{-1} after a transition in any of the following cases.*

1. *The transition is either* `HoldingPatternDescend(`*opposite*(σ)`)`*,* `Merging`*,* `FinalSegment`*,* `Taxiing`*, or* `LowestAvailableAltitude(`*opposite*(σ)`)`*.*
2. *The transition is either* `Exit` *or* `Landing`
3. *The transition is* `MissedApproach`*(a) and* a.`mahf` $=$ *opposite*(σ)*.*
4. *The transition is* `VerticalApproachInitiation`(a, *opposite*(σ))*, and* a.`mahf` $=$ *opposite*(σ)*.*
5. *The transition is for the entry of aircraft to opposite*(σ) *–* `VerticalEntry`(*opposite*(σ)) *or* `LateralEntry(`*opposite*(σ)`)`*.*

This lemma is defined in PVS as follows. As in the case of Lemma 3.21, we have two lemmas in PVS to differentiate the cases when we have the exact equality for the value of `blocked_except_for_one?` and the case when we only have an implication.

```
blocked_except_for_one?_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side, b:Aircraft):
         (HoldingPatternDescend?(a)     OR
          Merging?(a)                   OR
          FinalSegment?(a)              OR
          Taxiing?(a)                   OR
          LowestAvailableAltitude?(a)     ) AND
         enabled(a,s)
            IMPLIES
         blocked_except_for_one?(b,side,trans(a,s)) =
         blocked_except_for_one?(b,side,s)          )
```

```
blocked_except_for_one?_implication: LEMMA
      (FORALL (s:states, a:actions, side:Side, b:Aircraft):
        ((Exit?(a)                              OR
           Landing?(a)                          OR
           (MissedApproach?(a) AND
            mahf(ac(a)) = opposite(side))       OR
           (VerticalApproachInitiation?(a) AND
            mahf(ac(a)) = opposite(side)   AND
            side(a) = opposite(side)          ) OR
           ((VerticalEntry?(a) OR LateralEntry?(a)) AND
              side(a)=opposite(side))           )AND
          on?(side,b,s)                          AND
          enabled(a,s)                           AND
          reachable(s)                              )
            IMPLIES
         blocked_except_for_one?(b,side,s) =>
         blocked_except_for_one?(b,side,trans(a,s)))
```

Proof. Case 1: Transitions in this group do not affect the set of aircraft in the initiation areas or the landing sequence, or the mahf assignment of aircraft. Since all conditions that are related to the needed blocking condition are these conditions, the blocking condition holds after the transition.

Case 2: The transition just removes an aircraft from the approach area. Thus it does not affect the initiation areas, or the order of the remaining aircraft in the landing sequence. Thus the condition follows from the assumption on the pre state.

Case 3: The aircraft that misses the approach by the transition is added to the end of the landing sequence. Thus this aircraft is blocked by the blocking aircraft b in the post state. The condition for the rest of the aircraft in $opposite(\sigma)$ holds since the transition does not affect the preceding relation between these aircraft and the blocking aircraft b.

Case 4: The aircraft c that is not blocked in $opposite(\sigma)$ in the pre state is still in that side after the transition, since the aircraft a that initiates the approach by the transition is not c, considering that a is assigned the opposite side of σ. The condition for all the remaining aircraft in $opposite(\sigma)$holds in the post state since the transition does not affect the preceding relation between them and the blocking aircraft b.

Case 5: Since the newly entering aircraft by the transition is added to the end of the landing sequence, it is preceded by the blocking aircraft b in the post state. The transition does not affect either the aircraft c that is not blocked in the pre state, or the preceding relation between the rest of the aircraft in $opposite(\sigma)$ and the blocking aircraft b. Thus the condition holds in the post state from the induction hypothesis. □

Now we prove a lemma for ready aircraft. Note that this lemma states that the existence of a σ-ready aircraft in the post state implies the existence of such an aircraft in the pre state, rather than claiming the existence from the pre state to the post state. This direction is opposite from the lemmas for blocking conditions (Lemmas 3.21 and 3.22). This is because, when strengthening main properties using blocked conditions and ready-aircraft conditions, a blocking condition is stated as a conclusion, whereas, the existence of a ready aircraft is used as an assumption.

Lemma 3.23. *Consider any reachable state s of $SATS$, side σ, and transition π such that (s, π, s') is in the transition relation of the discrete model (s' is the post state of s after the transition π). If there is a σ-ready aircraft in s', then there is a σ-ready aircraft in s in any of the following cases.*

1. *The transition π is either* `HoldingPatternDescend`*,* `Merging`*,* `FinalSegment`*,* `Taxiing`*, or* `LowestAvailableAltitude`*.*
2. *The transition π is either* `Exit` *or* `Landing`*.*
3. *The transition π is* `MissedApproach`*(a), and $a.\mathsf{mahf} = \sigma$ in s.*
4. *The transition π is for an entry of aircraft to opposite(σ) –* `VerticalEntry(`*opposite*(σ)`)` *or* `LateralEntry(`*opposite*(σ)`)`*, and there is an aircraft on σ in s.*
5. *The transition π is* `MissedApproach`*(a), and $a.\mathsf{mahf} =$ opposite(σ) and there is an aircraft on σ in s.*
6. *The transition is* `VerticalApproachInitiation(`*opposite*(σ)`)`*.*

This lemma is defined in PVS as follows. As in Lemma 3.21 and Lemma 3.22, we have two lemmas in PVS to differentiate the case when we have the exact equality for the value of `ac_ready_to_approach?` and the case when we only have an implication.

```
ac_ready_to_approach?_unchanged: LEMMA
   (FORALL (s:states, a:actions, side:Side):
      (HoldingPatternDescend?(a)      OR
       Merging?(a)                    OR
       FinalSegment?(a)               OR
       Taxiing?(a)                    OR
       LowestAvailableAltitude?(a)      ) AND
      enabled(a,s) AND
      reachable(s)
         IMPLIES
      ac_ready_to_approach?(side,trans(a,s)) =
      ac_ready_to_approach?(side,s))
```

```
ac_ready_to_approach?_implication: LEMMA
   (FORALL (s:states, a:actions, side:Side):
      (Exit?(a)                                    OR
       Landing?(a)                                 OR
       (MissedApproach?(a) AND mahf(ac(a)) = side) OR
       ((((VerticalEntry?(a) OR LateralEntry?(a)) AND
           side(a)=opposite(side)) OR
         (MissedApproach?(a) AND mahf(ac(a)) = opposite(side))) AND
        EXISTS (ac:Aircraft): on?(side,ac,s))      OR
       (VerticalApproachInitiation?(a) AND
        side(a) = opposite(side))                   ) AND
      enabled(a,s) AND
      reachable(s)
        IMPLIES
      (ac_ready_to_approach?(side,trans(a,s)) =>
       ac_ready_to_approach?(side,s)))
```

Proof. Case 1: Transitions in this group do not affect the set of aircraft in the initiation areas or the landing sequence, or the mahf assignment of aircraft. Since all conditions that are related to the condition for the aircraft to be ready to approach from the opposite side of σ to σ, the condition holds in the pre state of the transition.

Case 2: The transition just removes an aircraft from the approach area. Thus it does not affect the initiation areas, or the order of the remaining aircraft in the landing sequence. Thus the condition follows from the assumption on the post state.

Case 3: The aircraft that misses the approach moves to `maz(`σ`)`, and thus the transition does not affect *opposite*(σ). Therefore a σ-ready aircraft c in the post state is already in that area in the pre state. the aircraft c precedes all aircraft in σ since it does so in the post state, and the only difference in σ between the pre and post states is that the aircraft that misses the approach by the transition is not in σ in the pre state. Thus this aircraft c satisfied the condition for a σ-ready aircraft in the pre state.

Case 4: The newly added aircraft a by the transition is not a σ-ready aircraft c since a is added to the end of the landing sequence, and thus is preceded by an aircraft in σ. This implies that c is already in *opposite*(σ) in the pre state. This aircraft c precedes all aircraft in the initiation area of σ since it does so in the post state, and the transition does not affect the side σ or the preceding relation of the aircraft in the landing sequence. Thus this aircraft c satisfied the condition for a σ-ready aircraft in the pre state.

Case 5: We can prove this case by a discussion analogous to Case 4. That is, the aircraft a that misses the approach by the transition is not σ-ready since a is added to the end of the landing sequence after the transition. Thus a σ-ready aircraft c in the post state is already in *opposite*(σ) in the pre state. From the same discussion as in Case 4, we conclude that this aircraft c satisfied the condition for a σ-ready aircraft in the pre state.

Case 6: The transition moves one aircraft in *opposite*(σ) to the approach area. A σ-ready aircraft c in the post state is already in *opposite*(σ) in the pre state, since if the transition moves c, it is on the approach in the post state, and thus is not σ-ready. This aircraft c precedes all aircraft in σ in the pre state since it does so in the post state, and the transition does not affect the side σ or the preceding relation of the aircraft in the landing sequence. □

3.6 Proving the main properties, Part 2: strengthening Property 6

In this section, we prove Property 6, which we need to strengthen by using a blocking condition defined in Section 3.5. The definition of the original Property 6 as an invariant is as follows.

Theorem 3.24. (Property 6) *For any reachable state of SATS and side σ, if* $\texttt{lez}(\sigma)$ *is non-empty, then* $\texttt{holding2}(\sigma)$, $\texttt{holding3}(\sigma)$, *and* $\texttt{maz}(\sigma)$ *are all empty.*

The property is defined in PVS as a predicate over states as follows.

```
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))
```

As we have mentioned, we cannot directly prove this property – we need to strengthen it to make an inductive proof go through. To see the reason, consider proving Property 6 by induction for an arbitrary side σ. We have to ensure that there is no aircraft a with $a.\texttt{mahf} = \sigma$ in the approach area when $\texttt{lez}(\sigma)$ is non-empty, since otherwise, one missed approach would violate the property. Now in turn, to prove this condition, we have to guarantee that no aircraft a with $a.\texttt{mahf} = \sigma$ will initiate the approach when $\texttt{lez}(\sigma)$ is non-empty. For this purpose, we need a blocking condition to guarantee that such a problematic approach initiation would not occur.

A possible blocking aircraft is only in $\texttt{lez}(\sigma)$ since we are proving that other zones in side σ are all empty. Furthermore, from Property 5, there is only one aircraft in that zone. Thus only the first aircraft of $\texttt{lez}(\sigma)$ can possibly be a blocking aircraft.

This discussion leads us to the following strengthened Property 6.

Lemma 3.25. (Strengthened Property 6) *For any reachable state of SATS and side σ, if* $\texttt{lez}(\sigma)$ *is non-empty, then the following conditions hold.*

(i) $\texttt{holding2}(\sigma)$, $\texttt{holding3}(\sigma)$, *and* $\texttt{maz}(\sigma)$ *are all empty.*

(ii) *No aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *opposite*(σ) *is b-blocked, where b is the first aircraft of* $\texttt{lez}(\sigma)$.

This strengthened property is defined in PVS 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)
```

The first condition is from the original Property 6; the second condition is to guarantee that a missed aircraft would not go to `maz(`σ`)` when `lez(`σ`)` is non-empty; and the third condition states a blocking condition. The state depicted in Figure 3.2 is actually an example of the states that satisfy these three conditions with respect to the right side. Namely, the first condition is satisfied since there is no aircraft in the right initiation area except for `lez(right)`, the second condition is vacuously true since there is no aircraft on the approach, and the third condition holds because the left initiation area is b-blocked by the first aircraft b of `lez(right)`.

Proof. By induction. There is no aircraft in any zone in the initial state. Thus the condition vacuously holds for the base case. Now we consider the inductive case. Let b be the first aircraft of `lez(`σ`)`, the blocking aircraft of our interest.

- *In the case of* `VerticalEntry`:

  In the case that a new aircraft enters `holding3`(σ), the precondition of the transition ensures that `lez(`σ`)` is empty in the pre state. Since it remains empty in the post state, the condition is vacuously true. In the case that a new aircraft enters `holding3`($\mathit{opposite}(\sigma)$), the first and second conditions of the lemma follow from the induction hypothesis and the fact that the transition does not affect the zones referred in the first condition or the approach area. The third condition follows from Lemma 3.21, and the fact that the transition does not affect `lez(`σ`)`, and thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

- *In the case of* `LateralEntry`:

  In the case that a new aircraft enters `lez(`σ`)`, the precondition of the transition ensures that $\mathit{virtual}(\sigma) = 0$. This implies that there is no aircraft in σ, and no aircraft a with $a.\mathsf{mahf} = \sigma$ is in $\mathit{opposite}(\sigma)$ or in the approach area. These facts immediately follow the three required conditions. In the case that a new aircraft enters `lez`($\mathit{opposite}(\sigma)$), the condition follows from the reason analogous to the case of `VerticalEntry` when an aircraft enters $\mathit{opposite}(\sigma)$.

- *In the case of* `VerticalApproachInitiation`:

  In the case that an aircraft initiates the approach from `holding2`(σ) by the transition:

The induction hypothesis implies that no aircraft is in `lez(`σ`)`, since otherwise, no aircraft is in `holding2(`σ`)`. Since the transition does not affect `lez(`σ`)`, the zone remains empty in the post state. Thus the condition is vacuously true. Now we consider the case that an aircraft initiates the approach from `holding2`(*opposite*(σ)). If `lez(`σ`)` is empty in the pre state, the condition holds from the same reason as the above case. If there is an aircraft in `lez(`σ`)` in the pre state, then from the induction hypothesis, the three conditions of the lemma hold in that state. Hence *opposite*(σ) is blocked by the first aircraft of `lez(`σ`)`. Thus only aircraft a with $a.\mathsf{mahf} = \textit{opposite}(\sigma)$ can initiate the approach from there. This fact and the induction hypothesis follow the second condition. The first condition immediately follows from the induction hypothesis since the transition does not affect any of zones referred in the condition. The third condition follows from Lemma 3.21, and the fact that the transition does not affect `lez(`σ`)`, and thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

- *In the case of* `LateralApproachInitiation`:

  In the case that an aircraft initiates the approach from `lez(`σ`)`, it follows that no aircraft is in `lez(`σ`)` in the post state, since, from Property 5, only one aircraft is there in the pre state. Thus the condition vacuously holds. In the case that an aircraft initiates the approach from `lez`(*opposite*(σ)), we can use an argument analogous to the case of `VerticalApproachInitiation` when an aircraft initiates from *opposite*(σ).

- *In the case of* `MissedApproach`:

  Suppose `lez(`σ`)` is non-empty in the post state. Since the transition does not affect `lez(`σ`)`, it is non-empty in the pre state, and thus from the induction hypothesis, the three conditions in the lemma hold in that state. It follows, from the second condition, that no aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach in the pre state. Thus an aircraft that misses the approach goes to `maz`(*opposite*(σ)). Hence the emptiness of the three zones referred in the first condition follows from this fact and the induction hypothesis. The second condition follows since, the transition just removes an aircraft from the approach area. The third condition follows from Lemma 3.21 and the fact that the transition does not affect `lez(`σ`)`, and thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

- *In the case of* `Exit` *or* `Landing`:

  Suppose `lez(`σ`)` is non-empty in the post state. Since the transition does not affect `lez(`σ`)`, it is non-empty in the pre state, and thus from the induction hypothesis, the three conditions in the lemma hold in that state. The first condition immediately follows from the induction hypothesis since the transition does not affect the initiation areas. The second condition follows again from the induction hypothesis since it just removes an aircraft from the approach area. The third condition follows from Lemma 3.21 and the fact that the transition does not affect `lez(`σ`)` thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

- *In the case of* `HoldingPatternDescend` *or* `LowestAvailableAltitude`
  Suppose `lez(`σ`)` is non-empty in the post state. Since the transition does not affect `lez(`σ`)`, it is non-empty in the pre state, and thus from the induction hypothesis, the three conditions in the lemma hold in that state. The transition for side σ is disabled since no aircraft is in either `holding3(`σ`)` or `maz(`σ`)`. If the transition is performed on *opposite*(σ), then, since it does not affect the side σ or the approach area, the first and second conditions follow from the induction hypothesis. The third condition follows from Lemma 3.21 and that the fact that the transition does not affect `lez(`σ`)`, and thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

- *In the case of* `Merging`*,* `FinalSegment`*, or* `Taxiing`:
  Suppose `lez(`σ`)` is non-empty in the post state. Since the transition does not affect `lez(`σ`)`, the zone is non-empty in the pre state, and thus from the induction hypothesis, the three conditions in the lemma hold in that state. The transition does not affect the initiation areas, or the set of aircraft on the approach. Thus the first and second conditions follow from the induction hypothesis. The third condition follows from Lemma 3.21 and the fact that the transition does not affect `lez(`σ`)`, and thus the blocking aircraft b remains being the first aircraft of `lez(`σ`)`.

□

3.7 Proving the main properties, Part 3: the key lemma, and the remaining properties

In this subsection, we present a key lemma to prove the rest of the main properties. As we mentioned in Section 3.4, this lemma has the longest and most complex statement, and its proof is also complicated because of the substantial number of case analyses and discussions on blocking conditions. The lemma consists of nine conditions, where two of them state main properties, Properties 3 ($\forall\sigma : Side, |\texttt{holding3}(\sigma)| \leq 1 \wedge |\texttt{holding2}(\sigma)| \leq 1$) and 4 ($\forall\sigma : Side, |\texttt{maz}(\sigma)| \leq 1$), and the remaining seven conditions describe case analyses for seven different blocking situations.

Each of these seven conditions has a form analogous to the strengthened Property 6 proved in Section 3.6.

Property 2 ($\forall \sigma : Side, actual(\sigma) \leq 2$), the last main property to be proved, easily follows from this key lemma, as discussed in the end of this section.

3.7.1 Intuition behind the lemma

Since the statement of the key lemma is complicated in itself, we present some intuition behind how the lemma is constructed in this subsection.

First, we examine the reason why we need multiple conditions describing different blocking situations to be proved together, rather than just a single blocking situation as in Lemma 3.25.

Consider proving Property 4 ($\forall \sigma : Side, |\texttt{maz}(\sigma)| \leq 1$) by induction. Analogous to the case of Property 6 strengthened and proved in Section 3.6, when two aircraft are in $\texttt{maz}(\sigma)$, we have to guarantee that no aircraft a such that $a.\mathsf{mahf} = \sigma$ is on the approach, since otherwise one missed approach would violate the bound. Now, to ensure the above fact, we need $opposite(\sigma)$ to be blocked by some aircraft. From this discussion, in order to prove Property 4, one may come up with the following claim stated as Claim 1, which describes one blocking situation. This claim has a form analogous to the strengthened Property 6 (Lemma 3.25).

Claim 1. *For any reachable state of $SATS$ and side σ, if $|\texttt{maz}(\sigma)| = 2$, then the following conditions hold.*

(i) $\texttt{holding2}(\sigma)$ *and* $\texttt{holding3}(\sigma)$ *are empty*

(ii) *No aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *$opposite(\sigma)$ is b-blocked, where b is some aircraft in side σ.*

As in Lemma 3.25, this Claim 1 has three conditions: a condition on the emptiness of zones for side σ; a condition on the mahf assignments of aircraft in the approach area; and a blocking condition for $opposite(\sigma)$. Analogous to Lemma 3.25, the second condition guarantees that a missed aircraft goes to $opposite(\sigma)$, and thus a missed approach would not cause any violation of Property 4. As in Lemma 3.25, the third condition, a blocking condition, is used to guarantee the second condition, by guaranteeing that every aircraft initiating from $opposite(\sigma)$ is assigned $opposite(\sigma)$. The first condition guarantees that there is no approach initiation from side σ.[4]

[4] One might think that this condition is strong since we could possibly use the blocking argument for side σ as well as the opposite side. However, considering that we prove Property 2 later, which states that the total number of aircraft in one initiation area is at most two, we can see that this emptiness condition is actually a necessary condition for Property 2

Even though describing one specific blocking situation works well in the case of Lemma 3.25, it turns out that, as we see in the following, Claim 1, which describes one blocking situation, is not strong enough to be proved by induction.

The reason why Lemma 3.25 can be proved without having any other case, whereas Claim 1 is not strong enough to be proved by induction, comes from the assumptions of these two results. In Lemma 3.25, we assume $|\texttt{lez}(\sigma)| = 1$. The value of $|\texttt{lez}(\sigma)|$ increases just by $\texttt{LateralEntry}(\sigma)$, which has a strict examination of the safety separation in its precondition: $\mathit{virtual}(\sigma) = 0$. As we saw in the proof of the strengthened Property 6 (Lemma 3.25), this precondition directly implies the required blocking condition.

In contrast, in Claim 1, we assume $|\texttt{maz}(\sigma)| = 2$. The value of $|\texttt{maz}(\sigma)|$ increases by `MissedApproach`, which, as we have discussed several times, has no "guard" in its precondition to examine the current situation of the system (it is enabled whenever `final` is non-empty). This implies that there is no way we can guarantee the required blocking condition by the precondition of the transition, and thus we need an analogous blocking condition to hold in the pre state before `MissedApproach` is performed.

For this purpose, we need the following Claim 2, which has a form analogous to Claim 1, but represents the "pre situation" just before $|\texttt{maz}(\sigma)|$ gets two by `MissedApproach`.

Claim 2. *For any reachable state of SATS and side σ, if $|\texttt{maz}(\sigma)| = 1$ and some aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach, then the following conditions hold.*

(i) $\texttt{holding2}(\sigma)$ *and* $\texttt{holding3}(\sigma)$ *are empty*

(ii) *At most one aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *$\mathit{opposite}(\sigma)$ is b-blocked, where b is some aircraft in side σ.*

In this claim, we have the exact same conditions for the first and third conclusions as in Claim 1. The difference is in the assumption and the second conclusion. We assume that $|\texttt{maz}(\sigma)| = 1$, and some aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach. This reflects the fact that the Claim 2 represents the "pre case" just before $|\texttt{maz}(\sigma)|$ gets set to 2 by `MissedApproach`. The second conclusion, the only conclusion that differs from Claim 1, states what should be true in the "pre case" of Claim 1 in order to prove the second conclusion of Claim 1. Namely, at most one aircraft a with $a.\mathsf{mahf} = \sigma$ must be on the approach, since no such aircraft must be there after `MissedApproach`.

Now, consider proving Claim 2 by induction. First, as in the case of Claim 1, let us examine the case of the `MissedApproach` transition. As opposed to Claim 1, in this case, we *can* guarantee the blocking condition without depending on another claim. Indeed, we can guarantee the blocking condition from an main property proved so far, as follows. In Claim 2, we assume that $|\mathtt{maz}(\sigma)| = 1$ and some aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach. This implies that if the post state of `MissedApproach` satisfies these assumptions, then at least two aircraft a with $a.\mathsf{mahf} = \sigma$ are on the approach in the pre state. It follows from Theorem 3.15 (Property 7; $assigned2fix(\sigma) \leq 2$) no aircraft a with $a.\mathsf{mahf} = \sigma$ is outside of the approach area. This implies that no aircraft a with $a.\mathsf{mahf} = \sigma$ is in *opposite*(σ). Thus, from the definition of b-blocked, *opposite*(σ) is b-blocked by any aircraft b in the operation area. From the above discussion, we do not need any additional pre cases for Claim 2 in the case of `MissedApproach`. Note that we could not use the same argument for Claim 1 using Property 7 because, even though aircraft in $\mathtt{maz(}\sigma\mathtt{)}$ must have been assigned σ for its mahf before they missed the approach, it by no means guarantee that their mahf's are still assigned σ after the re-assignment of mahf by `MissedApproach`.

A problem in proving Claim 2 occurs in the case of $\mathtt{VerticalApproachInitiation(}\sigma\mathtt{)}$. Even though this transition has a precondition that represents a guard that delays the approach initiation, it does not guarantee any blocking condition. Indeed, the precondition checks if the aircraft initiating the approach follows the order in the landing sequence, and if $|\mathsf{base}(\mathsf{right})| + |\mathsf{base}(\mathsf{left})| \leq 3$. Nevertheless, it does not examine any condition about the initiation areas. This implies that we need another claim that represents the "pre situation" just before the assumption of Claim 2 is satisfied by $\mathtt{VerticalApproachInitiation(}\sigma\mathtt{)}$. The assumption of Claim 3 in the following represents the situation just before the assumption of Claim 2 get satisfied by $\mathtt{VerticalApproachInitiation(}\sigma\mathtt{)}$, that is, the states in which $|\mathtt{maz}(\sigma)| = 1$, and either $\mathtt{holding2(}\sigma\mathtt{)}$ or $\mathtt{holding3(}\sigma\mathtt{)}$ is non-empty.

Claim 3. *For any reachable state of $SATS$ and side σ, if $|\mathtt{maz}(\sigma)| = 1$ and either* $\mathtt{holding2}(\sigma)$ *or* $\mathtt{holding3}(\sigma)$ *is non-empty, then the following conditions hold.*

(i) $|\mathtt{holding2}(\sigma)| + |\mathtt{holding3}(\sigma)| \leq 1$.

(ii) *No aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *opposite(σ) is b-blocked, where b is some aircraft in side σ.*

Analogous to the previous two claims, there are three conclusions in this claim. First, $|\mathtt{holding2}(\sigma)| + |\mathtt{holding3}(\sigma)| \leq 1$ must be true since after $\mathtt{VerticalApproachInitiation(}\sigma\mathtt{)}$ is performed, there must be no aircraft in $\mathtt{holding2}(\sigma)$ and $\mathtt{holding3}(\sigma)$, in order to prove the first conclusion of Claim 2. Second, no aircraft a with $a.\mathsf{mahf} = \sigma$ must be on the approach, since one aircraft assigned σ as its mahf initiates approach by $\mathtt{VerticalApproachInitiation(}\sigma\mathtt{)}$, and at most one aircraft assigned σ as its mahf must be on the approach after the transition, in order to prove the second conclusion in Claim 3. Third, we need a blocking condition corresponding to the third conclusion of Claim 3.

In this way, we can explore more cases until we construct sufficiently many cases so that we can prove the required blocking condition for each case either from those cases used as induction hypotheses, or from main properties that have already been proved. We already saw one situation where we can use Property 7 to prove the required blocking condition in the discussion for the "pre case" of Claim 2 just before `MissedApproach` is performed.

An important point we want to know is whether these cases can be proved individually in some order, or need to be proved together in an inductive proof. It turns out that even the three claims above have a dependency on each other. As we discussed above, we need Claim 2 to prove Claim 1 in the case of `MissedApproach`, and need Claim 3 to prove Claim 2 in the case of $\texttt{VerticalApproachInitiation}(\sigma)$. In addition, we actually need Claim 1 to prove Claim 3 from the following reason. In Claim 3, we assume that $|\texttt{maz}(\sigma)| = 1$, and either $\texttt{holding2}(\sigma)$ or $\texttt{holding3}(\sigma)$ is non-empty. Consider proving Claim 3 by induction. In the case of the $\texttt{LowestAvailableAltitude}(\sigma)$ transition, since its precondition does not guarantee anything about the required blocking condition (the transition is always enabled if $\texttt{maz}(\sigma)$ is non-empty), we have to obtain the blocking condition using some "pre situation" analogous to the cases of Claim 1 and Claim 2. The pre situation of Claim 3 for $\texttt{LowestAvailableAltitude}(\sigma)$ is represented by the states in which $|\texttt{maz}(\sigma)| = 2$ This is exactly what Claim 1 assumes. Furthermore, Claim 1 is actually sufficient to prove Claim 3 in the case of $\texttt{LowestAvailableAltitude}(\sigma)$. Thus we need Claim 1 to prove Claim 3.

To see the dependency of the claims discussed above, see Figure 3.4, which depicts "transitions between cases." An arrow between claims represents that the claim the arrow points at needs the claim the arrow starts from as an induction hypothesis when we analyze the transition that labels the arrow. For example, Claim 2 is needed to prove Claim 1, more specifically in the case of the `MissedApproach` transition. Analogously, we need Claim 3 when we prove Claim 2 by induction, more specifically in the case of the $\texttt{VerticalApproachInitiation}(\sigma)$ transition. Finally, we need Claim 1 when we prove Claim 3 by induction, more specifically in the case of $\texttt{LowestAvailableAltitude}(\sigma)$. This observation closes a cycle of dependency of claims by adding an arrow from Claim 1 to Claim 3. This implies that we have to prove these claims (and actually more cases) together in an inductive proof.

It turns out that the three claims stated above are not strong enough to be proved inductively, and thus we have to revise them slightly. In these claims, we do not specify which aircraft blocks *opposite*(σ); instead, we just state that *some* aircraft blocks it. As we will see in the

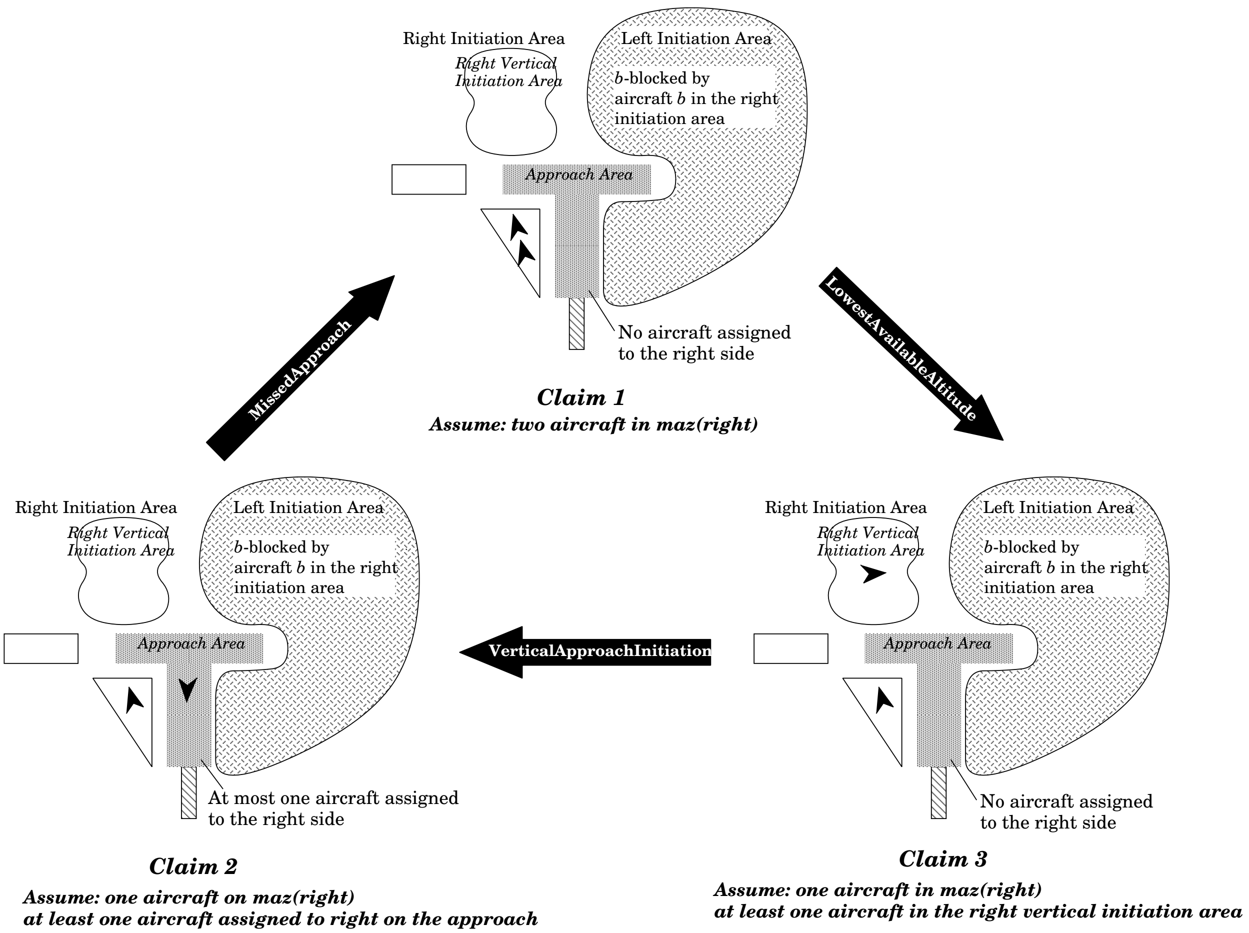

Right Initiation Area
Right Vertical Initiation Area
Left Initiation Area
b-blocked by aircraft b in the right initiation area
Approach Area
No aircraft assigned to the right side
Claim 1
Assume: two aircraft in maz(right)
MissedApproach
LowestAvailableAltitude
Right Initiation Area
Right Vertical Initiation Area
Left Initiation Area
b-blocked by aircraft b in the right initiation area
Approach Area
At most one aircraft assigned to the right side
Claim 2
Assume: one aircraft on maz(right)
at least one aircraft assigned to right on the approach
VerticalApproachInitiation
Right Initiation Area
Right Vertical Initiation Area
Left Initiation Area
b-blocked by aircraft b in the right initiation area
Approach Area
No aircraft assigned to the right side
Claim 3
Assume: one aircraft in maz(right)
at least one aircraft in the right vertical initiation area

Figure 3.4: Transitions between cases creating a circle

following, this ambiguity causes a problem when we prove Claim 2 using Claim 3 as an induction hypothesis. We revise the three claims in two steps. First, we revise the claims by specifying a blocking aircraft in each claim. It turns out that after this simple refinement, Claim 1 contains a contradictory statement. We resolve this problem in the second revision by introducing a finer case analysis of a blocking aircraft.

First Revision: As we discussed before, when we prove Claim 2 by induction, we need Claim 3 as an induction hypothesis in the case of `VerticalApproachInitiation(`σ`)`. It turns out that assuming that *some* aircraft blocks the side σ is not sufficient to prove Claim 2. This is because, without specifying the position of a blocking aircraft, the blocking aircraft can possibly be the aircraft a that initiates the approach by
`VerticalApproachInitiation(`a, σ`)`. In such a case, we cannot use this blocking aircraft a to prove Claim 2, since the blocking aircraft specified in Claim 2 must be in side σ, and a is on the approach after the transition. This implies that we have to clearly specify in which position the

blocking aircraft for Claim 3 is.

A more careful examination of possible positions of the blocking aircraft resolves this problem. Considering the assumption of Claim 2, we actually have only one possible blocking aircraft for that claim, namely the first aircraft of $\texttt{maz}(\sigma)$. This is because of the following reason. First, $\texttt{holding2}(\sigma)$ and $\texttt{holding3}(\sigma)$ are empty (this is what we prove for the claim), and $\texttt{lez}(\sigma)$ is also empty from Theorem 3.24 (Property 6). Thus the only possible zone at which the blocking aircraft can be located is $\texttt{maz}(\sigma)$. Furthermore, since we assume in Claim 2 that $|\texttt{maz}(\sigma)| = 1$, the blocking aircraft must be the first aircraft of $\texttt{maz}(\sigma)$. This discussion leads us to the following revised version of Claim 2. The only change from the original Claim 2 is that now we specify the position of the blocking aircraft in Conclusion (iii).

Claim 2. (First Revision) *For any reachable state of SATS and side σ, if $|\texttt{maz}(\sigma)| = 1$ and some aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach, then the following conditions hold.*

(i) $\texttt{holding2}(\sigma)$ *and* $\texttt{holding3}(\sigma)$ *are empty*

(ii) *At most one aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *opposite(σ) is b-blocked, where b is the first aircraft of* $\texttt{maz}(\sigma)$.

To reflect this change of Claim 2 to Claim 3 and Claim 1, we also specify the position of the blocking aircraft in these two claims as well. Here we have to be careful. The blocking aircraft should match up in these three claims, since otherwise we cannot use the blocking condition of one claim to prove the blocking condition of another claim.

We designate the first aircraft of $\texttt{maz}(\sigma)$ as the blocking aircraft in Claim 3. This is because Claim 3 is used in the proof of Claim 2 in the case of $\texttt{VerticalApproachInitiation}(\sigma)$. Since this transition does not affect $\texttt{maz}(\sigma)$, the first aircraft of $\texttt{maz}(\sigma)$ must be the blocking aircraft for Claim 3 as well as Claim 2.

Claim 3. (First Revision) *For any reachable state of SATS and side σ, if $|\texttt{maz}(\sigma)| = 1$ and either* $\texttt{holding2}(\sigma)$ *or* $\texttt{holding3}(\sigma)$ *is non-empty, then the following conditions hold.*

(i) $|\texttt{holding2}(\sigma)| + |\texttt{holding3}(\sigma)| \leq 1$.

(ii) *No aircraft a with $a.\mathsf{mahf} = \sigma$ is on the approach.*

(iii) *opposite(σ) is b-blocked, where b is the first aircraft of* $\texttt{maz}(\sigma)$.

Analogously, we have to revise Claim 1. Recall that Claim 1 is used to prove Claim 3 in the case of the $\texttt{LowestAvailableAltitude}(\sigma)$ transition. We designate in Claim 1 the *second* aircraft of `maz(`σ`)` to be the blocking aircraft. This is because when there are two aircraft in `maz(`σ`)` (the assumption of Claim 1), the second aircraft of `maz(`σ`)` becomes the first aircraft of that zone after the `LowestAvailableAltitude(`σ`)` transition. Since we have to match up the blocking aircraft in Claim 1 with the blocking aircraft in Claim 3 (the first aircraft of `maz(`σ`)`), we need the second aircraft of `maz(`σ`)` to be the blocking aircraft in Claim 1.

Claim 1. (First Revision) *For any reachable state of $SATS$ and side σ, if* $|\mathtt{maz}(\sigma)| = 2$*, then the following conditions hold.*

(i) `holding2(`σ`)` *and* `holding3(`σ`)` *are empty*

(ii) *No aircraft a with a.*$\mathsf{mahf} = \sigma$ *is on the approach.*

(iii) *opposite*(σ) *is b-blocked, where b b is the second aircraft of* `maz(`σ`)`.

Second Revision: We revised the three claims by specifying the actual position of the blocking aircraft. However, the new statement in Claim 1 – that the *second* aircraft of `maz(`σ`)` blocks the initiation area – seems contradictory. This is because the second aircraft of `maz(`σ`)` may have just entered `maz`(σ) by `MissedApproach`. In such a case, the aircraft has been added to the end of the landing sequence, and hence cannot be a blocking aircraft except for some special situation explained in the following. The only case that an aircraft that has just missed the approach can be a blocking aircraft is the case when all aircraft in *opposite*(σ) are assigned *opposite*(σ) as their mahf. In this case, from the definition of b-blocked, we can conclude that the required blocking condition holds no matter what aircraft is specified as a blocking aircraft. However, there *are* some reachable states in which some aircraft a with a.`mahf` $= \sigma$ is in *opposite*(σ) and, at the same time, the assumption of Claim 1 is satisfied. This implies that Claim 1 need another revision.

Adding a finer case analysis resolves this contradictory statement in Claim 1. We use two different blocking aircraft in Claim 1, depending on the mahf assignment of the first aircraft m of `maz(`σ`)`. If m.`mahf` $= \sigma$, then *opposite*(σ) is blocked by the *second* aircraft of `maz(`σ`)`, as originally stated in the claim. If m.`mahf` $=$ *opposite*(σ), then *opposite*(σ) is blocked by the *first* aircraft m of `maz(`σ`)`. Note that the blocking aircraft in the second case, the first aircraft of `maz(`σ`)`, is not chosen arbitrarily. Indeed, the possible positions of the blocking aircraft are just the above mentioned two, since we prove in this claim that `holding2(`σ`)` and `holding3(`σ`)` are empty, and from Property 6 proved in Theorem 3.24, `lez(`σ`)` is also empty.

Claim 1. (Second Revision) *For any reachable state of $SATS$ and side σ, if* $|\mathtt{maz}(\sigma)| = 2$*, then the following conditions hold.*

(i) `holding2(`σ`)` *and* `holding3(`σ`)` *are empty*

(ii) *No aircraft a with a.*mahf $= \sigma$ *is on the approach.*

(iii) *Let b be the first aircraft m of* `maz(`σ`)` *if m.*`mahf` $=$ *opposite*(σ)*; otherwise let b be the second aircraft of* `maz(`σ`)`*. opposite*(σ) *is b-blocked.*

To reflect this change in Claim 1, we also modify Claim 3. Recall that Claim 1 is needed to prove Claim 3 by induction, namely in the case of `LowestAvailableAltitude(`σ`)`. When there are two aircraft in `maz(`σ`)`, `LowestAvailableAltitude(`σ`)` transition makes the following changes: the first aircraft of `maz(`σ`)` goes to `holding2`(σ) or `holding3`(σ), and the second aircraft of `maz(`σ`)` becomes the first aircraft of that zone. Thus in order to match up blocking aircraft for Claim 1 and Claim 3, we modify the blocking condition for Claim 3 as follows.

Claim 3. (Second Revision) *For any reachable state of $SATS$ and side σ, if* $|$`maz`$(\sigma)| = 1$ *and either* `holding2`(σ) *or* `holding3`(σ) *is non-empty, then the following conditions hold.*

(i) $|$`holding2`$(\sigma)| + |$`holding3`$(\sigma)| \leq 1$.

(ii) *There is no aircraft a with a.*mahf $= \sigma$ *on the approach.*

(iii) *Let b be the first aircraft h of the concatenation* holding2$(\sigma) \circ$ holding3(σ) *if h.*`mahf` $=$ *opposite*(σ)*; otherwise let b be the first aircraft of* `maz`*(σ). opposite*(σ) *is b-blocked.*

We have modified the blocking condition in Claim 3 and Claim 1. Now see in the following how the contradictory statement in Claim 1 (the second aircraft of `maz(`σ`)` blocks *opposite*(σ) has been resolved with this modification. In the revised Claim 1, we designate the second aircraft of `maz(`σ`)` as the blocking aircraft only when the mahf of the first aircraft of `maz(`σ`)` is assigned σ. We assume the mahf of the first aircraft of `maz(`σ`)` is assigned σ in the following. A key is that under the assumption of Claim 1, if the mahf of the first aircraft of `maz(`σ`)` is assigned σ, then we have the special case mentioned earlier in this subsection, where any aircraft is a blocking aircraft. This follows from the discussion analogous to the case we saw when Claim 2 is introduced: Before the second aircraft of `maz(`σ`)` missed the approach, it had been assigned σ. Considering that we assume the first aircraft of `maz(`σ`)` is assigned σ as its mahf, two aircraft a with a.`mahf` $= \sigma$ are on the approach before `MissedApproach` occurs. From Property 7 proved in Lemma 3.15, $assigned2fix(\sigma) \leq 2$. It follows that *no aircraft a* with a.mahf $= \sigma$ is in *opposite*(σ). Since this fact also holds after `MissedApproach`, the second aircraft of `maz(`σ`)` *is* a blocking aircraft, regardless of whether or not it precedes aircraft in *opposite*(σ).

For simplicity, we explained how the lemma is constructed by focusing on the three claims each of which describes a different blocking situation. However, we need more cases than the three cases described in the above stated three claims in order to prove them together inductively. For example, we need a "pre case" of the situation assumed in Claim 3 before the `LowestAvailableAltitude(`σ`)` is performed. By adding these missing cases to the three claims and two main properties, we obtain the complete statement of Lemma 3.26 presented in the following subsection.

3.7.2 The key lemma

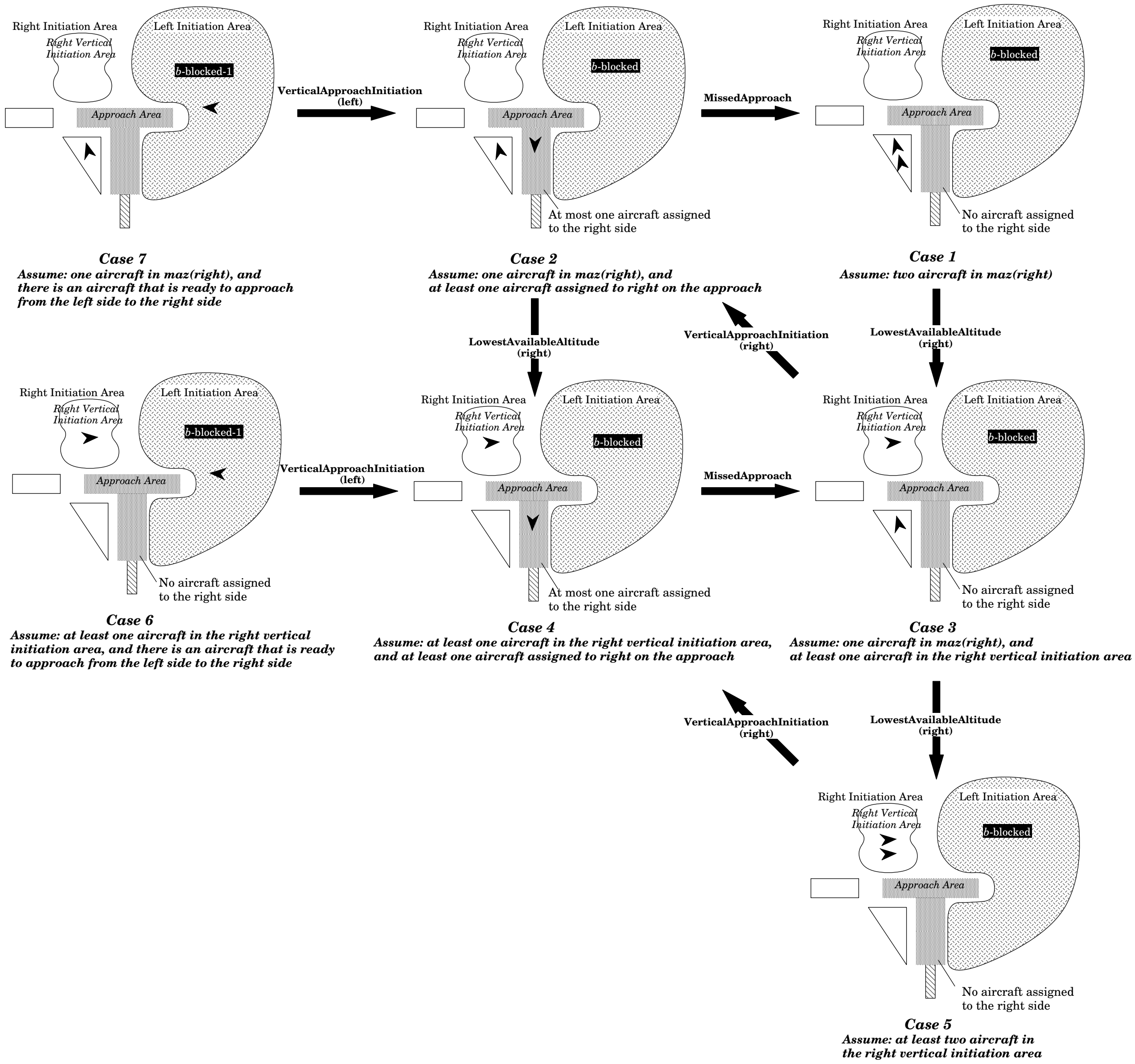
In this subsection, we present the complete lemma statement. It consists of a conjunction of two main properties, Properties 3 and 4, and seven case conditions each of which describes a different blocking situation.

In Figure 3.5, we show a high level picture of the seven different blocking situations described as Cases 1 - 7 in the lemma: the picture represents abstract "transitions" between seven cases, following the same philosophy of Figure 3.4. An arrow between two cases represents that the case at which an arrow starts become a "pre case" of the case at which an arrow ends (a "post case"). To prove the conditions of a case, we need a pre case of that case as an induction hypothesis, specifically when analyzing the transition that labels the arrow for that case and its pre case.

Now we explain how these seven cases are constructed. The first three cases, Cases 1, 2, and 3, of the seven correspond to the three claims, (revised) Claim 1, 2, and 3, respectively, discussed in Section 3.7.1.

The following two cases, Cases 4 and 5, also share the same construction scheme as Cases 1 - 3. That is, these two cases are constructed in the following way.[5] The first conclusion of a case states an upper bound on the number of aircraft in specific zones of side σ; more specifically, the conclusion is defined in a way that the bound on one specific zone specified in the conclusion can be used to prove the corresponding bound on the same zone for the "post case", as we discussed when constructing Claims 2 and 3. The second conclusion of a case describes the bound on the number of aircraft a on the approach with $a.\texttt{mahf} = \sigma$. This bound is also determined in a way that the bound can be used to prove the corresponding bound for the "post case". The third conclusion of a case describes a blocking condition, which states that some specific aircraft in side σ blocks *opposite*(σ). This condition guarantees a scenario in which the model violates the second conclusion would never occur. We specified the blocking aircraft in the third conclusion in a way that the blocking aircraft specified in this case becomes the blocking aircraft specified

[5] The reader might consider that the fact that these cases share the same scheme may imply that there may possibly be a way to describe the lemma in a more concise way, rather than having seven different cases. However, as we discussed in the previous subsection, we have to specify the blocking aircraft in each case, and there is not quite a good way to describe the position of the blocking aircraft in a uniform way that applies to all seven cases.



Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked-1
Approach Area
Case 7
Assume: one aircraft in maz(right), and there is an aircraft that is ready to approach from the left side to the right side
VerticalApproachInitiation (left)
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked
Approach Area
At most one aircraft assigned to the right side
Case 2
Assume: one aircraft in maz(right), and at least one aircraft assigned to right on the approach
MissedApproach
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked
Approach Area
No aircraft assigned to the right side
Case 1
Assume: two aircraft in maz(right)
LowestAvailableAltitude (right)
VerticalApproachInitiation (right)
LowestAvailableAltitude (right)
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked-1
Approach Area
No aircraft assigned to the right side
Case 6
Assume: at least one aircraft in the right vertical initiation area, and there is an aircraft that is ready to approach from the left side to the right side
VerticalApproachInitiation (left)
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked
Approach Area
At most one aircraft assigned to the right side
Case 4
Assume: at least one aircraft in the right vertical initiation area, and at least one aircraft assigned to right on the approach
MissedApproach
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked
Approach Area
No aircraft assigned to the right side
Case 3
Assume: one aircraft in maz(right), and at least one aircraft in the right vertical initiation area
VerticalApproachInitiation (right)
LowestAvailableAltitude (right)
Right Initiation Area
Left Initiation Area
Right Vertical Initiation Area
b-blocked
Approach Area
No aircraft assigned to the right side
Case 5
Assume: at least two aircraft in the right vertical initiation area

Figure 3.5: Transitions between seven cases in the lemma

in the "post case" after the transition.

We construct the remaining cases, Cases 6 and 7, based the same philosophy as the former five cases, but these two cases consider a slightly different blocking situation. In these cases, we consider a situation in which one aircraft c with c.`mahf` $= \sigma$ in *opposite*(σ) *can* initiate the approach, but *any other* aircraft have to be blocked. This is because, for instance, Case 6 represents the "pre case" of Case 2 before the `VerticalApproachInitiation(opposite(`σ`))` is performed. In such a case, we allow exactly one aircraft c with c.`mahf` $= \sigma$ to initiate the approach, but any other aircraft have to be blocked, since, after the transition, σ has to be blocked, as described in Case 2. We express this kind of situations using σ-*ready* and b-*blocked*$^{-1}$ defined in Section 3.5.

Now we present the complete lemma. It turns out that Case 2 does not need the condition on the emptiness of holding zones (`holding3(`σ`)` and `holding2(`σ`)`), since we can prove this fact using some other cases. Thus the condition is removed from the case.[6] From the same reason, Case 7 needs only the blocking condition.

Lemma 3.26. *For any reachable state of SATS and side σ, the following two properties and the seven case conditions hold.*

Property 3: $|$`holding3`$(\sigma)| \leq 1 \wedge |$`holding2`$(\sigma)| \leq 1$.

Property 4: $|$`maz`$(\sigma)| \leq 1$.

Case 1: *If* $|$`maz`$(\sigma)| = 2$*, then the following conditions hold.*

(i) `holding2(`σ`)` *and* `holding3(`σ`)` *are empty*

(ii) *No aircraft a with a.*`mahf` $= \sigma$ *is on the approach.*

(iii) *Let b be the first aircraft m of* `maz(`σ`)` *if m.*`mahf` $=$ *opposite*(σ)*; otherwise let b be the second aircraft of* `maz(`σ`)`*. opposite(σ) is b-blocked.*

Case 2: *If* $|$`maz`$(\sigma)| = 1$ *and some aircraft a with a.*`mahf` $= \sigma$ *is on the approach, then the following conditions hold.*

(i) `holding2(`σ`)` *and* `holding3(`σ`)` *are empty*

(ii) *At most one aircraft a with a.*`mahf` $= \sigma$ *on the approach.*

(iii) *opposite(σ) is b-blocked, where b is the first aircraft of* `maz(`σ`)`*.*

[6] We could have chosen to retain these conditions to preserve the uniformity of the form of the case statements. However, in order to shorten the proof length (especially in PVS) by removing the conditions that we do not actually need to prove, we chose to remove this condition from the case statement.

Case 3: *If* $|\texttt{maz}(\sigma)| = 1$ *and either* $\texttt{holding2}(\sigma)$ *or* $\texttt{holding3}(\sigma)$ *is non-empty, then the following conditions hold.*

(i) $|\texttt{holding2}(\sigma)| + |\texttt{holding3}(\sigma)| \leq 1$.

(ii) *No aircraft* a *with* $a.\mathsf{mahf} = \sigma$ *is on the approach.*

(iii) *Let* b *be the first aircraft* h *of the concatenation* $\mathsf{holding2}(\sigma) \circ \mathsf{holding3}(\sigma)$ *if* $h.\texttt{mahf} =$ *opposite*(σ)*; otherwise let* b *be the first aircraft of* $\texttt{maz}(\sigma)$*. opposite*(σ) *is* b*-blocked.*

Case 4: *If either* $\texttt{holding2}(\sigma)$ *or* $\texttt{holding3}(\sigma)$ *is non-empty, and some aircraft* a *with* $a.\mathsf{mahf} = \sigma$ *is on the approach, then the following conditions hold.*

(i) $|\texttt{holding2}(\sigma)| + |\texttt{holding3}(\sigma)| \leq 1$.

(ii) *At most one aircraft* a *with* $a.\mathsf{mahf} = \sigma$ *on the approach.*

(iii) *opposite*(σ) *is* b*-blocked, where* b *is the first aircraft of the concatenation* $\mathsf{holding2}(\sigma) \circ \mathsf{holding3}(\sigma)$.

Case 5: *If both* $\texttt{holding2}(\sigma)$ *and* $\texttt{holding3}(\sigma)$ *are non-empty, then the following conditions hold.*

(i) $\texttt{maz}(\sigma)$ *is empty.*

(ii) *No aircraft* a *with* $a.\mathsf{mahf} = \sigma$ *is on the approach.*

(iii) *Let* b *be the first aircraft* h *of the concatenation* $\mathsf{holding2}(\sigma) \circ \mathsf{holding3}(\sigma)$ *if* $h.\texttt{mahf} =$ *opposite*(σ)*; otherwise let* b *be the second aircraft of* $\mathsf{holding2}(\sigma) \circ \mathsf{holding3}(\sigma)$*. opposite*$(\sigma)$ *is* b*-blocked.*

Case 6: *If either* $\texttt{holding2}(\sigma)$ *or* $\texttt{holding3}(\sigma)$ *is non-empty, and there is a* σ*-ready aircraft, then the following conditions hold.*

(i) $|\texttt{holding2}(\sigma)| + |\texttt{holding3}(\sigma)| \leq 1$ *and* $\texttt{maz}(\sigma)$ *is empty.*

(ii) *No aircraft* a *with* $a.\mathsf{mahf} = \sigma$ *is on the approach.*

(iii) *opposite*(σ) *is* b*-blocked*$^{-1}$*, where* b *is the first aircraft of the concatenation* $\mathsf{holding2}(\sigma) \circ \mathsf{holding3}(\sigma)$.

Case 7: *If* $|\texttt{maz}(\sigma)| = 1$ *and there is a* σ*-ready aircraft, then the following condition holds.*

opposite(σ) *is* b*-blocked*$^{-1}$*, where* b *is the first aircraft of* $\texttt{maz}(\sigma)$

The lemma is defined in PVS as follows.

```
%% case 1: |maz(side)| = 2
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: |maz(side)| = 1 and on_approach?(side).
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: |maz(side)| = 1 and either h2(side) or h3(side) is not empty
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: on_approach?(side) and either h2(side) or h3(side) is not empty
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 h2(side) and h3(side) are non-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 a side-ready aircraft and
%% either h2(side) or h3(side) is not empty
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 a side-ready aircraft  and |maz(side)| = 1
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)
```

3.7.3 Proof of Lemma 3.26

Now we start proving Lemma 3.26. Before going into the detailed proof for the lemma, we first present some common strategies used to prove the conditions for each of Cases 1 to 7.

The easiest case is that the assumption of the case in the post state and the effect of the transition imply the assumption of the same case in the pre state. For example, when we prove Case 1 in the case of `Landing`, the transition does not affect `maz(`σ`)`. Thus, if $|$`maz`$(\sigma)| = 2$ in the post state, then $|$`maz`$(\sigma)| = 2$ in the pre state as well. Thus the assumption of Case 1 is satisfied in the pre state. In such a case, we can make use of the conditions that hold in the pre state to prove the corresponding conditions in the post state, by using the same argument as in the proof of Lemma 3.25 (the strengthened Property 6). In these cases, as in Lemma 3.25, we use Lemma 3.21 or 3.22 to obtain the required blocking condition for the case from the corresponding blocking condition that holds for the pre state.

In order to prove some specific cases in the seven cases, we need another case as an induction hypothesis. For example, when we prove Case 1 in the case of `MissedApproach`, we use Case 2 as the "pre case," as discussed in Section 3.7.1. In this case, the assumption of the "pre case" follows from the assumption of the "post case" and the effects and the precondition of the transition. Thus, we can prove the conditions of the post case using the corresponding conditions of the pre case. We again use Lemma 3.21 or 3.22 to obtain the required blocking condition for the case from the from the corresponding blocking condition for the "pre case". When applying these lemmas, we have to make sure that the blocking aircraft match up between the pre state and the post state.

As we saw in Section 3.7.1, in some cases, we obtain the required blocking condition by main properties that have been proved. A property mainly used for this purpose is Property 7 (Theorem 3.15) that states $assigned2fix(\sigma) \leq 2$. We demonstrated how this property is used to obtain the required blocking condition in Section 3.7.1 when we consider proving Claim 2, namely in the case of MissedApproach. In that case, the assumption in the post state and the effects of the transition imply that two aircraft a with $a.\texttt{mahf} = \sigma$ are on the approach. It follows from Property 7 that there is no other aircraft b with $b.\texttt{mahf} = \sigma$ outside of the approach area. This fact gave us the required blocking condition.

In the proof of Cases 6 and 7 in the case of VerticalApproachInitiation, we have a different discussion from the discussions stated above, in order to obtain the required blocking condition, as we will see in the proof.

The actual proof of Lemma 3.26 is as follows. The proof is more than ten pages long, due to the the substantial number of case analyses for the seven cases and the twelve transitions.

Proof. By induction. There is no aircraft on any zone in the initial state. Thus the assumptions of all cases are not satisfied, and also the upper bounds for Properties 2 and 3 hold. Now we consider the inductive step.

- *In the case of* VerticalEntry:
  - *Properties 3 and 4*: The precondition of the transition guarantees that there is no aircraft in holding3 of the side where the new aircraft enters. Thus the number of aircraft in holding3 of that side is one in the post state. The bound on holding3 of the opposite side and the bounds on holding2 zones and maz zones hold from the induction hypothesis, since the transition does not affect these zones.
  - *Case 1*: Suppose there are two aircraft in maz(σ) in the post state. Since the transition does not effect maz(σ), there are already two aircraft in maz(σ) in the pre state. The transition for the side σ cannot have been performed since the precondition of the transition is not satisfied.

    If a new aircraft enters into the opposite side of σ, since there are already two aircraft in maz(σ) in the pre state, the three conditions of Case 1 hold in the pre state from the induction hypothesis. The emptiness of holding2(σ) and holding3(σ) follows from the facts that the same emptiness holds in the pre state and that the transition does not affect these zones. The condition of the assignments in the approach area analogously follows since it holds in the pre state and the transition does not affect the approach area. The blocking condition follows from Lemma 3.21, and the fact that the transition does not affect `maz`, thus the specified blocking aircraft in the pre state remains being in the same position in the same zone.

- *Case 2*: Suppose there are one aircraft in `maz(`σ`)` and at least one aircraft assigned σ on the approach in the post state. The transition for the side σ cannot have been performed from a reason analogous to Case 1: there is already one aircraft in `maz(`σ`)`, and thus the transition is disabled.

  If a new aircraft enters into the opposite side of σ, since the transition does not effect `maz(`σ`)` or the approach area, the pre state of the transition also satisfies the assumption of Case 2. Thus we can use an argument analogous to Case 1 to prove two conditions using Lemma 3.21 and the fact that the transition does not affect `maz(`σ`)` or the approach area.

- *Case 3*: We can prove this case using a discussion analogous to the previous two cases. That is, when we assume the assumption of the case in the post state, the transition for side σ has been disabled, and the transition for the opposite side does not affect the initiation area of σ or the approach area, and thus preserves the emptiness of the vertical initiation area, and the number of assignments to one side in the approach area. The blocking condition follows from Lemma 3.21.

- *Case 4*: Suppose that in the post state of the transition, there are at least one aircraft in the vertical initiation area of side σ and at least one aircraft assigned σ in the approach area. First consider the case that the transition is performed for side σ. Since the transition does not affect the approach area, there are already at least one aircraft assigned σ in the area in the pre state. Thus the transition for the side σ is disabled in the pre state.

  In the case of the transition for the opposite side of σ, we can use a discussion analogous to the previous cases.

- *Case 5*: Suppose that in the post state of the transition, there are at least two aircraft in the vertical initiation area of side σ. First consider the case that the transition is performed for side σ. The emptiness of `maz(`σ`)` in the pre condition holds since the precondition of the transition guarantees that there are no aircraft in `maz(`σ`)` in the pre state and the transition does not affect that zone. Analogously, the condition on the assignments in the approach area follows from the pre condition of the transition and the fact that the transition does not affect that area. There must be at lest one aircraft in the vertical initiation area of side σ in the pre state, since there are at least two in the post state and the transition adds just one aircraft into that area. From this fact and the condition on the potential number of aircraft in the initiation area of side σ stated in the precondition – the number is less than two, it follows that there are no aircraft assigned σ outside of the initiation area of σ. It implies the remaining condition to be proved since from the definition of the blocking of aircraft, the initiation area is blocked by any aircraft with respect to side σ if there is no aircraft assigned σ in the area.

In the case of the transition for the opposite side of σ, we can use a discussion analogous to the previous cases.

- *Case 6*: Suppose that in the post state of the transition, there are at least one aircraft on the vertical approach area of side σ, and there is an aircraft that is ready to approach from the opposite side of σ to side σ. First consider the case that the transition is performed for side σ. The emptiness of `maz(`σ`)` and the condition on the assignments in the approach area follows analogously to Case 5 from the precondition of the transition and the fact that the transition does not affect these zone and area. The blocking condition also follows from a discussion analogous to Case 5: since the transition does not affect the initiation area of the opposite side of σ, the aircraft that is ready to approach from that area to side σ have already been there in the pre state. Since this aircraft has the mahf assignment of side σ, it is counted as the potential aircraft that possibly goes to side σ. In addition, from the precondition of the transition, the potential number of aircraft on the initiation area of σ is less than two. It implies that there are no aircraft assigned σ in the initiation area of the opposite side of σ, other than the aircraft mentioned above – the aircraft that is ready to approach. Thus the blocking condition holds.

  In the case of the transition for the opposite side of σ, we can use a discussion analogous to the previous cases. Note that in this case, we use Lemma 3.23 to obtain the fact that the conditions of Case 6 hold in the pre state, and use Lemma 3.22, instead of Lemma 3.21, to prove the blocking condition.

- *Case 7*: Suppose that in the post state of the transition, there are one aircraft in `maz(`σ`)` and there is an aircraft that is ready to approach from the opposite side of σ to side σ. The transition for side σ is disabled in the pre state since there is one aircraft in `maz(`σ`)` in the pre state considering that the transition does not affect that zone.

  In the case of the transition for the opposite side of σ, we can use a discussion analogous to Case 6.

- *In the case of* `LateralEntry`:
  - *Properties 3 and 4*: The transition does not affect either `holding3(`σ`)`, `holding2(`σ`)`, or `maz(`σ`)`. Thus the conditions hold from the induction hypothesis.
  - *Cases 1 to 6 in the case that the transition is performed for the opposite side of* σ: We can prove the six cases by a discussion analogous to the corresponding cases of `VerticalEntry` using Lemmas 3.21, 3.23, and 3.22, and the fact that the transition does not affect the initiation area of side σ or the approach area.

    We consider the case that the transition is performed for side σ in the following.
  - *Cases 1, 2, 3, and 7*: Suppose there are at least one aircraft in `maz(`σ`)` in the post state as we suppose in these cases, and also suppose that the transition for side σ is performed. It follows that these aircraft are already in that zone in the pre state, since the transition does not affect `maz(`σ`)`. It implies that the potential number of aircraft in the initiation area of side σ is at least one in the pre state. This is a contradiction since the precondition of the action ensures that the potential number is zero in the pre state.
  - *Case 4*: We can prove this case by contradiction in a way analogous to the above cases. For this case, we assume that there are at least one aircraft assigned σ in the approach area in the post state. Since the transition does not affect the approach area, these aircraft assigned σ are already in the area in the pre state. This leads to a contradiction with the precondition of the transition that states that the potential number of aircraft in the initiation area of side σ is zero in the pre state.
  - *Cases 5 and 6*: In these cases, we assume that there are at least one aircraft in the vertical initiation area of side σ in the post state. Since the transition does not affect that area, these aircraft are already in the area in the pre state. Now we have a contradiction analogous to above cases concerning the potential number of aircraft.

- *In the case of* `VerticalApproachInitiation`:
  - *Properties 3 and 4*: The transition removes an aircraft from `holding2`. Thus the transition does not increase the number of aircraft in `holding2`, `holding3`, and `maz`. Thus the conditions hold from the induction hypothesis.
  - *Case 1*: Suppose there are two aircraft in `maz(`σ`)` in the post state. Since the transition does not affect `maz` zones, there are exactly two aircraft in `maz(`σ`)` in the pre

state as well. It implies that the pre state satisfies the assumption of Case 1, and thus from the induction hypothesis, the three conditions of Case 1 hold in the pre state. I follows that there is no aircraft in the vertical approach initiation area of side σ, and hence the transition for side σ is disabled in the pre state.

Now we consider the case that the transition is performed for the opposite side of σ. Since the transition does not affect the vertical approach initiation of side σ, the emptiness conditions on `holding3` and `holding2` hold from the induction hypothesis. The bound on the assignments of aircraft in the approach area follows from the facts that the same bound holds in the pre state, and that the number of assignments to σ in the approach area does not change by the transition since the initiation area of the side where an aircraft initiates the approach is blocked. The blocking condition follows from Lemma 3.21 and the facts that the same blocking condition holds in the pre state, and that the transition does not move the blocking aircraft in the pre state.

– *Case 2*: Suppose in the post state, there is one aircraft in `maz(`σ`)` and there are at least one aircraft assigned σ in the approach zone. Since the transition does not affect `maz` zones, there is exactly one aircraft in `maz(`σ`)` in the pre state as well.

  First we consider the case that the transition is performed for side σ. The precondition for the transition ensures that there are at least one aircraft in `holding2(`σ`)` in the pre state. It implies that the pre state satisfies the assumption of Case 3, and thus the three conditions of the case hold in the pre state. The bound on the number of the assignments of aircraft in the approach area, which is one, holds since there are no aircraft assigned σ in the approach from the conditions of Case 3, and the transition just adds one aircraft to the approach area. In addition, considering that there are at least one aircraft assigned σ on the approach in the post state, the aircraft that initiates the approach by this transition should be assigned σ. It implies from the blocking condition of Case 3 that the initiation are of the opposite side of σ is blocked by the first aircraft of `maz(`σ`)`. Since the transition does not move this aircraft, the blocking condition in the post state follows from Lemma 3.21.

  Next we consider the case that the transition is performed for the opposite side of σ. We split the case depending on whether there is already an aircraft assigned σ on the approach in the pre state.

  ∗ If there is an aircraft assigned σ on the approach in the pre state, it implies that the pre state satisfies the assumption of Case 2, and thus the conditions of Case 2 hold in the pre state. It follows that the initiation area of the opposite side is blocked, and there are at most one aircraft assigned σ on the approach. The same bound on the assignments of aircraft in the approach area follows from the above two facts, and the blocking condition of Case 2 follows from the same blocking condition that hold in the pre state and Lemma 3.21.

  - Suppose there is no aircraft assigned to σ in the approach area in the pre state. Considering that from the assumption of the case, there are at least such aircraft in the approach area in the post state, the aircraft that initiates the approach by this transition must be assigned σ. In addition, the fact that this aircraft can initiate the approach implies that this aircraft satisfies the condition for an aircraft that is ready to go approach from the opposite side of σ to side σ. It implies that the pre state satisfies the assumption of Case 7, and thus the two conditions of Case 7 hold in the pre state. Since this aircraft initiates the approach by the transition, and except for this aircraft, the initiation area of the opposite side of σ is blocked by the first aircraft of `maz(`σ`)`, that area is blocked by the same blocking aircraft in the post state. The condition on the assignments of aircraft in the approach area follows from the facts that there is no aircraft assigned σ in the approach area in the pre state, and that the transition adds just one aircraft to the approach area.

- *Case 3*: Suppose there is one aircraft in `maz(`σ`)` and there are at least one aircraft in the vertical initiation area of side σ. Since the transition does not affect `maz` zones, there is one aircraft in `maz(`σ`)` in the post state as well.

  First we consider the case that the transition is performed for side σ. The precondition for the transition ensures that there are at least one aircraft in `holding2(`σ`)` in the pre state. Since the transition removes one aircraft from `holding2(`σ`)`, there must be another aircraft in the vertical initiation area in the pre state since there are at least one aircraft in the area in the post state. It implies that the pre state satisfies the assumption of Case 5, and thus there is no aircraft in `maz(`σ`)`. This is a contradiction.

  Next we consider the case that the transition is performed for the opposite side of σ. Since the transition does not affect the vertical initiation area of side σ, there are at least one aircraft in that area in the pre state. It follows that the pre state satisfies the assumption of Case 3, and thus the three conditions of Case 3 hold in the pre state. The bound on the number of aircraft in the vertical initiation area follows from the same bound that holds in the pre state, and the fact that the transition does not affect the area. The condition on the number of assignments in the approach area follows from the same condition that holds in the pre state, and the fact that the initiation area of the side where an aircraft initiates the approach by this transition is blocked. The blocking condition holds from the same blocking condition that holds in the pre state, Lemma 3.21, and the fact that the transition does not move the blocking aircraft.

- *Case 4*: Suppose that in the post state there are at least one aircraft in the vertical initiation area of side σ, and at least one aircraft assigned σ on the approach.

  First we consider the case that the transition is performed for side σ. The precondition of the transition ensures that `holding2(`σ`)` is not empty. Considering that there are at least one aircraft in the vertical initiation area of side σ even after the transition that removes one aircraft from that area, there must be at least two aircraft in the area in the pre state. It follows that the pre state satisfies the assumption of Case 5, and thus the three conditions of Case 5 hold in the pre state. From the second condition, there is no aircraft assigned σ in the approach area in the pre state. Considering that there are at least such aircraft in the same area after the transition, the aircraft that initiates the approach by this transition must be assigned σ. It follows that the initiation are of the opposite side of σ is blocked by the second aircraft of the vertical initiation area of side σ. Since this aircraft becomes the first aircraft of the same area, the blocking condition that we need holds from Lemma 3.21. The condition of the assignments follows from the facts that there is no aircraft assigned σ on the approach in the pre state, and that the transition just adds one aircraft to the approach area. The bound on the number of aircraft in the vertical initiation area of σ follows since there is at most one aircraft in `holding2` and `holding3`, respectively, from Property 3 that holds in the pre state, and the transition removes one aircraft from `holding2(`σ`)`.

  Next we consider the case that the transition is performed for the opposite side of σ. We can prove the conditions using a discussion analogous to Case 2 as follows. If there already is an aircraft assigned σ in the approach in the pre state, then the pre state satisfies the assumption of Case 2. We can use a discussion analogous to that in Case 2 to prove the three conditions using the same three conditions that hold in the pre state. If there is no aircraft assigned σ in the approach area in the pre state, then considering the fact that there are at least one such aircraft in the same area after the transition, the aircraft that initiates the approach by this transition must be assigned σ. From this and the fact that this aircraft can initiate the approach, it satisfy the condition for an aircraft that is ready to approach from the opposite side of σ to side σ. It follows that the pre state satisfies the assumption of Case 6, and thus the three conditions of the case hold in the pre state. We can prove the conditions for case 4 using the three conditions above in a way analogous to Case 2.

- *Case 5*: Suppose that in the post state, there are at lest two aircraft in the vertical approach area of side σ. First we consider the case that the transition is performed for side σ. From Property 3 that holds in the pre state, there is at most one aircraft in both `holding2` and `holding3`, respectively. Since the transition removes one aircraft from `holding2(`σ`)`, that zone becomes empty after the transition. It follows that there is at most one aircraft in the vertical approach area of side σ in the post state. This is a contradiction.

  Next we consider the case that the transition is performed for the opposite side of σ. Since the transition does not affect the vertical initiation area of side σ, there are at least two aircraft in the vertical approach area of side σ in the pre state. It follows that the pre state satisfies the assumption of Case 5, and thus the three conditions of the case hold in the pre state. We can prove the three conditions of Case 5 using the three conditions above and Lemma 3.21 in a way analogous to the corresponding case in Case 1 or Case 3.

- *Case 6*: Suppose that in the post state there are at least one aircraft in the vertical initiation area of side σ, and there is an aircraft that is ready to approach from the opposite side of σ to side *sigma*.

  First we consider the case that the transition is performed for side σ. From the precondition of the transition, `holding2(`σ`)` is not empty in the pre state. Considering that the transition removes one aircraft from that zone, there must be two aircraft in the vertical initiation area of side σ in the pre state since there is still one aircraft after the transition. It implies that the pre state satisfies the assumption of Case 5, and thus the three conditions of the case hold in the pre state. It implies that the initiation area of side σ is blocked by some specific aircraft with respect to σ. We split the case into following two cases depending on the blocking aircraft.

  * If the first aircraft of `holding2(`σ`)` in the pre state is assigned σ, the area is blocked by the first aircraft of `holding3(`σ`)`. From Lemma 3.21 and the fact that the transition does not move the specified blocking aircraft, the area is still blocked by the same aircraft after the transition. It contradicts the assumption in the post state that there is an aircraft that is ready to approach from that initiation area to side σ.

* If the first aircraft of `holding2(`σ`)` in the pre state is assigned the opposite side of σ, the area is blocked by that aircraft. The emptiness of `maz(`σ`)` follows from the facts that it is empty in the pre state and that the transition does not affect the zone. The bound on the number of aircraft in the vertical initiation area of σ, which is one, follows from Property 3 that holds in the pre state and the fact that the transition removes one aircraft from `holding2(`σ`)`. The condition on the number of assignments to σ in the approach area follows from the same condition that holds in the pre state and the fact that the aircraft that initiates the approach by the transition is assigned the opposite side of σ, and thus the number of assignments to σ in the approach area does not change by the transition.

  Now we prove the blocking condition. In this case, we cannot use the same discussion as the above case to prove the blocking condition since the blocking aircraft initiates the approach by the transition, and thus does not block any aircraft after the transition. As we show in the following, we have to use the blocking condition for side σ, as opposed to the opposite side of σ like we have done so far, in order to lead to a contradiction. For a sake of contradiction, suppose that there are more than one aircraft, not necessarily ready to approach, in the initiation area of the opposite side of σ. From Theorem 3.16 (Property 5), two aircraft cannot be in `lez(`$\mathit{opposite}(\sigma)$`)`. Furthermore, from Theorem 3.24 (Property 6), there cannot be any aircraft in `lez(`$\mathit{opposite}(\sigma)$`)` since otherwise aircraft exist in both `lez(`$\mathit{opposite}(\sigma)$`)` and other initiation area of $\mathit{opposite}(\sigma)$, which contradicts Property 6. Thus, two aircraft are in the initiation area of the opposite side of σ excluding `lez(`$\mathit{opposite}(\sigma)$`)`. It implies that by any possible position of two aircraft in the area, the assumption of either Case 1, 3, or 5 for the opposite side of σ is satisfied. It implies that in that area, all aircraft assigned the opposite side of σ are blocked by some specific aircraft in side σ. This contradicts the fact that the first aircraft of `holding2(`σ`)`, which is assigned the opposite side of σ, initiates the approach by this transition. Thus there is at most one aircraft in the initiation area of the opposite side of σ, and hence the blocking condition for this case holds.

Next we consider the case that the transition is performed for the opposite side of σ. Since the transition does not affect the vertical initiation are of side σ, there are at least one aircraft in that area in the pre state. In addition, from Lemma 3.23, there is an aircraft that is ready to approach from the opposite side of σ to side σ in the pre state. It follows that the pre state satisfies the assumption of Case 6, and thus the three conditions of the case hold in the pre state. We split the case depending on the mahf assignment of the aircraft that initiates the approach by this transition.

  - $*$ If the mahf assignment of the aircraft that initiates the approach is σ, then it implies that after the transition, there is no aircraft assigned to σ since the initiation area where this aircraft initiates the approach is blocked with respect to σ except for the aircraft. This contradicts with the fact that in the post state, there is an aircraft that is ready to approach from this initiation area to side σ.
  - $*$ Suppose the mahf assignment of the aircraft that initiates the approach is the opposite side of σ. The emptiness condition of $\texttt{maz}(\sigma)$ and the bound on the number of aircraft in the vertical initiation area of σ follows from the facts that the same conditions hold in the pre state, and that the transition does not affect these zone and area. The bound on the number of aircraft assigned σ in the approach holds since the same bound holds in the pre state, and the aircraft that initiates the approach is assigned the opposite side of σ, and thus does not affect the number of the assignments to σ. The blocking condition follows from Lemma 3.22 and the facts that the same blocking condition holds in the pre state and that the transition does not move the blocking aircraft.

- *Case 7*: Suppose that in the post state there is exactly one aircraft in $\texttt{maz}(\sigma)$, and there is an aircraft that is ready to approach from the opposite side of σ to side *sigma*. Since the transition does not affect $\texttt{maz}(\sigma)$, there is exactly one aircraft in that zone in the pre state.

  First we consider the case that the transition is performed for side σ. From the precondition of the transition, $\texttt{holding2}(\sigma)$ is not empty in the pre state. It follows that the pre state satisfies the assumption of Case 3, and thus the three conditions of the case hold in that state. We split the case into two depending on the mahf assignment of the first aircraft of $\texttt{holding2}(\sigma)$.

  - $*$ If the first aircraft of $\texttt{holding2}(\sigma)$ is assigned σ, then from the blocking condition holds in the pre state from Case 3, the initiation area of the opposite side of σ is blocked by the first aircraft of $\texttt{maz}(\sigma)$ with respect to σ. Since the transition does not affect $\texttt{maz}(\sigma)$, this blocking aircraft stays in that zone after the transition, and thus the initiation area of the opposite side of σ is still blocked by the aircraft. This contradicts with the fact that there is an aircraft that is ready to approach from that area to side *sigma*.

  - ∗ If the first aircraft of `holding2(`σ`)` is assigned the opposite side of σ, we can use the same discussion as in the corresponding case in the proof of Case 6. That is we prove that there must be just one aircraft in the initiation area of the opposite side of σ by applying the blocking condition to side σ.

Next we consider the case that the transition is performed for the opposite side of σ. From Lemma 3.23, there is an aircraft that is ready to approach from the opposite side of σ to side *sigma* in the pre state as well. It implies that the pre state satisfies the assumption of Case 7, and thus the conditions of the case hold in that state. We can use a discussion analogous to the corresponding case in the proof of Case 6. That is, if the aircraft that initiates the approach by the transition is assigned σ, then the initiation area of the opposite side of σ is blocked in the pre state, and thus it is a contradiction; and if the aircraft is assigned the opposite side of σ, then we can use Lemma 3.22 to obtain the blocking condition.

- *In the case of* `LateralApproachInitiation`:

  – *Property 3 and 4*: The transition does not affect either `holding2`, `holding3`, or `maz`. Thus the bounds immediately follows from the corresponding bounds that hold in the pre state.

  – *Cases 1 to 6 in the case that the transition is performed for side σ*: All of these cases assume that there are at least one aircraft in either the vertical initiation area or `maz` of side σ. Since the transition does not affect these area and zone, if we assume the case assumption holds in the post state, it implies that there are at least one aircraft in these area and zone in the pre state. It follows from Theorem 3.24 (Property 6) that there is no aircraft in `lez(`σ`)` in the pre state. This contradicts with the precondition of the transition that ensures that the zone is not empty.

  – *Cases 1 to 6 in the case that the transition is performed for the opposite side of σ*: The transition moves an aircraft to the different space depending on the pre state. In the case that the aircraft moves to the approach area, we can prove the conditions using a discussion analogous to the corresponding case in the proof of `VerticalApproachInitiation`. In the case that the aircraft moves to `holding2(`*opposite*(σ)`)`, we can prove the conditions using a discussion analogous to the corresponding case in the proof of `VerticalEntry`.

- *In the case of* `MissedApproach`:

- *Properties 3*: The transition does not affect the vertical initiation area. Thus the bound immediately follows from the induction hypothesis.
- *Properties 4*: We prove the bound on the number of `maz(`γ`)` for an arbitrary side γ. In the case that the mahf of the aircraft that misses the approach is assigned γ: If the number of aircraft in `maz(`γ`)` in the pre state is strictly less than two, then since the transition just adds one aircraft, the number of aircraft in `maz(`γ`)` in the post state is at most two. If the number of aircraft in `maz(`γ`)` in the pre state is exactly two, it follows that the pre state satisfies the assumption of Case 1. It implies that there is no aircraft assigned γ on the approach. This contradicts the fact that the aircraft that misses the approach has γ as its mahf.

  In the case that the mahf of the aircraft that misses the approach is assigned the opposite side of γ: the aircraft that misses the approach goes to `maz` of the opposite side of γ. Thus the transition does not affect `maz(`γ`)`. Hence the condition follows from the induction hypothesis.
- *Case 1*: Suppose there are two aircraft in `maz(`σ`)` in the pre state.

  We first consider the case that the mahf of the aircraft that misses the approach is assigned σ. Since one aircraft enters `maz(`σ`)` by the transition, the number of aircraft in `maz(`σ`)` in the pre state is exactly one, and there are at least one aircraft assigned to σ on the approach. It follows that the pre state satisfies the assumption of Case 2, and thus the three conditions of the case hold in the pre state. Since the transition removes one aircraft from the approach area, the bound on the number of assignments of aircraft immediately holds from the same bound that holds in the pre state. The emptiness of the vertical initiation area of side σ follows from the following reason. If there is an aircraft in that area in the post state, it is already in the area in the pre state since the transition does not affect the area. It implies that the pre state satisfies the assumption of Case 3. It follows that there is no aircraft assigned σ on the approach in the pre state. This is a contradiction. Thus the vertical initiation area of σ must be empty. Next we prove the blocking condition. We have to consider two cases depending on the mahf assignment of the first aircraft of `maz(`σ`)` in the post state.

  * If the mahf assignment of the aircraft is σ in the post state, then we have to prove that the second aircraft of `maz(`σ`)` blocks the initiation area of the opposite side of σ in that state. In the pre state, we have two aircraft assigned to σ: one is the first aircraft of `maz(`σ`)`, and the other is in the approach area. Thus from

Theorem 3.15, there is no other aircraft assigned to σ in the operation area in that state. Since the transition does not add any aircraft to the initiation area of the opposite side of σ, there is no aircraft assigned to σ in that area after the transition, and thus the area is blocked by any aircraft in the system, especially by the second aircraft of `maz(`σ`)`, in the post state, as needed.

  * If the mahf assignment of the aircraft is the opposite side of σ, we have to prove that this aircraft blocks the initiation area of the opposite side of σ. The condition follows from the same blocking condition that holds in the pre state and Lemma 3.21.

  Next we consider the case that the mahf of the aircraft that misses the approach is assigned the opposite side of σ. The aircraft that misses the approach goes to `maz` of the opposite side of σ. It implies that there are already two aircraft in `maz(`σ`)` in the pre state. Thus the pre state satisfies the assumption of Case 1, and hence the three conditions of the case hold in the pre state. The emptiness of the vertical initiation area follows immediately from the fact that the area is empty in the pre state, since the transition does not affect the area. The condition on the number of aircraft assigned σ in the approach area follows from the same condition that holds in the pre state, since the transition removes one aircraft from the area, thus does not increase the number of aircraft. The blocking condition follows from Lemma 3.21 and the facts that the same blocking condition holds in the pre state, and that the transition does not move the blocking aircraft.

- *Case 2*: Suppose that in the post state, there is one aircraft in `maz(`σ`)` and there are at least one one aircraft assigned σ on the approach.

  We first consider the case that the mahf of the aircraft that misses the approach is assigned σ. Considering that the transition removes one aircraft from the approach area, and this aircraft is assigned σ, there must be at lest two aircraft assigned σ in the approach area in the pre state. Furthermore, from Theorem 3.15 (Property 7), there are exactly two aircraft assigned σ in the area, and there is no other aircraft assigned σ in other areas. It implies that there is exactly one aircraft assigned σ in the approach area in the post state, and the initiation are of the opposite side of σ is blocked with respect to σ by any aircraft, and thus especially the blocking aircraft specified in the case, as needed.

  In the case that the mahf of the aircraft that misses the approach is assigned the opposite side of σ, we can use a discussion analogous to the corresponding case in the proof of Case 1. That is, we first assert that the assumption of Case 1 holds in the pre state as well, and then prove the conditions using the corresponding conditions that hold in the pre state and Lemma 3.21.

- *Case 3*: Suppose that in the post state, there is one aircraft in `maz(`σ`)` and there are at least one aircraft in the vertical initiation area of side σ.

  We first consider the case that the mahf of the aircraft that misses the approach is assigned σ. The assumption of the case implies that there are at lest one aircraft assigned σ in the approach area in the pre state. In addition, considering that the transition does not affect the vertical approach areas, there are at least one aircraft in the vertical initiation area of side σ. It follows that the pre state satisfies the assumption of Case 4, and thus the three conditions of the case hold in the pre state. The bound on the number of aircraft in the vertical approach area immediately follows from the same bound that holds in the pre state, since the transition does not affect the area. There are no aircraft assigned σ in the approach area after the transition since there is at least one such aircraft in the pre state from a condition of Case 4, and we identify this aircraft with the aircraft that misses the approach by the transition because it is assigned σ. To prove the blocking condition, we consider the following two cases.

  - Suppose the first aircraft of the vertical initiation area of side σ is assigned σ in the pre state. It follows that there are two aircraft assigned σ in the pre state: one is the aircraft above, and the other is the aircraft that misses the approach by the transition. Thus we can prove the blocking condition in the same way as Case 2 using Theorem 3.15 (Property 7).

  - Suppose the first aircraft of the vertical initiation area of side σ is assigned the opposite side of σ in the pre state. From the blocking condition of Case 4 that holds in that state, the initiation area of the opposite side of σ is blocked by the first aircraft of the vertical initiation area of σ. From Lemma 3.21, the area is blocked by the same aircraft in the same position, as needed.

  In the case that the mahf of the aircraft that misses the approach is assigned the opposite side of σ, we can use a discussion analogous to the corresponding situation in the previous cases.

- *Case 4*: Suppose that in the post state, there are at least one aircraft in the vertical initiation area of side σ and at least one aircraft assigned σ in the approach area. Considering that the transition does not affect the vertical initiation areas, there are at least one aircraft in that area in the pre state as well.

We first consider the case that the mahf of the aircraft that misses the approach is assigned σ. We lead to a contradiction to this assumption in the following. There are at least one aircraft assigned σ in the approach area after the transition removes one aircraft assigned σ from that area. It follows that in the pre state, there must be at least two aircraft assigned σ in the area. Thus the pre state satisfies the assumption of Case 4, and hence the three conditions of the case hold in the pre state. It implies that there is at most one aircraft assigned σ in the approach area. This is a contradiction.

In the case that the mahf of the aircraft that misses the approach is assigned the opposite side of σ, we can use a discussion analogous to the corresponding situation in the previous cases.

- *Case 5*: Suppose that in the post state, there are at least two aircraft in the vertical initiation area of side $sigma$. Since the transition does not affect the vertical initiation areas, there are at least two aircraft in that area in the pre state as well. It follows that the pre state satisfies the assumption of Case 5, and thus the three conditions of the case hold in the pre state. The second condition in Case 5 implies that there is no aircraft assigned σ in the approach in the pre state. Therefore the aircraft that misses the approach by this transition is assigned the opposite side of σ before the transition. We can use a discussion analogous to the corresponding situation in the previous cases in order prove each condition using the corresponding condition that holds in the pre state.

- *Case 6*: Suppose that in the post state, there are at least one aircraft in the vertical initiation area of side $sigma$, and there is an aircraft that is ready to approach from the opposite side of σ to side σ. Since the transition does not affect the vertical initiation areas, there are at least one aircraft in the vertical initiation area of σ in the pre state as well. In addition, from Lemma 3.23, there is an aircraft that is ready to approach from the opposite side of σ to side σ in the pre state. It implies that the pre state satisfies the assumption of Case 6, and thus the three condition of the case hold in that state. The second condition of Case 6 implies that there is no aircraft assigned σ in the approach in the pre state. Thus the aircraft that misses the approach by this transition is assigned the opposite side of σ before the transition. We can use a discussion analogous to the corresponding situation in the previous cases in order prove each condition using the corresponding condition that holds in the pre state. Note, however, that in this case, we have to use Lemma 3.22 instead of Lemma 3.21 to obtain the blocking condition.

- *Case 7*: Suppose that in the post state, there is one aircraft in `maz(`σ`)`, and there is an aircraft that is ready to approach from the opposite side of σ to side σ.

  We first consider the case that the mahf of the aircraft that misses the approach is assigned σ. The assumption of the case implies that there are two aircraft assigned σ in the pre state: one is the aircraft above, and the other is the one that is ready to approach to σ. Thus, from Theorem 3.15 (Property 7), there is no other aircraft assigned σ in the pre state. The condition on the number of assignments in the approach area in the post state follows from the above fact and the fact that the transition removes one aircraft assigned σ from the approach area, since this aircraft is the only one that is assigned σ in that area. The blocking condition follows from the following reason. In the pre state, there is no aircraft assigned σ in the initiation area of the opposite side of σ, except for the one mentioned above. It implies that this area is blocked, except for one aircraft, by any aircraft in the pre state with respect to σ. Using Lemma 3.22, we obtain the blocking condition we need in the post state.

- *In the case of* `LowestAvailableAltitude`:

  - *Property 3*: We will prove the condition for an arbitrary side γ. First we consider the case that the transition is performed for side γ. The precondition of the transition ensures that there are at least one aircraft in `maz(`γ`)` in the pre state. We split the case depending on the number of aircraft in the zone in that state.

    * Suppose there is exactly one aircraft on `maz(`γ`)` in the pre state. If there is at most one aircraft in the vertical approach zone after the transition, the bound vacuously holds. If there are more than one aircraft in the vertical approach zone in the post state, it implies that there are at least one aircraft in the vertical initiation area in the pre state. It follows that the pre state satisfies the assumption of Case 3, and thus there is at most one aircraft in the vertical initiation area of γ in that state. If there is no aircraft in the area, the bound holds since the transition just add one aircraft to `holding2(`γ`)`.

      Now consider the case that there is exactly one aircraft in the vertical initiation area of γ in the pre state. If one aircraft is on `holding2(`γ`)` and no aircraft is on `holding3(`γ`)`, the transition moves an aircraft to `holding3(`γ`)`. Thus both of these zones have exactly one aircraft after the transition. If one aircraft is on `holding3(`γ`)` and no aircraft is on `holding2(`γ`)`, the transition performs a simultaneous movement of aircraft: it moves the aircraft on holding3(γ) to holding2(γ), and the first aircraft of maz(γ) to holding3(γ). Thus both holding2(γ) and holding3(γ) have exactly one aircraft after the transition.

  - $*$ Suppose there are more than one aircraft on maz(γ) in the pre state. Property 4 that holds in the pre state implies that there are exactly two aircraft in that zone in the pre state. Hence, the pre state satisfies the assumption of Case 1. It follows that there is no aircraft in the vertical approach area. Thus there is exactly one aircraft in holding2(γ) and no aircraft in holding3(γ) in the post state.

- *Property 4*: The transition does not affect maz zones. Thus the condition immediately follows from the induction hypothesis.

- *Case 1*: Suppose there are two aircraft in maz(σ) in the post state. If the transition is performed for side σ, it implies that there are three aircraft in maz(σ) in the pre state. This contradicts with Property 4 that holds in the pre state.

  Now we consider the case that the transition is performed for the opposite side of σ. In this case, the transition does not affect the initiation area of side *sigma* or the approach area. In addition, the blocking condition in the pre state immediately implies the same blocking condition in the post state from Lemma 3.21. Thus the pre state satisfies the assumption of Case 1, and the conditions we have to prove immediately follows from the corresponding conditions that hold in the pre state. This discussion apply for the rest of the cases when the transition is performed for the opposite side of σ. Thus we will only prove the case the transition is performed for side σ in the following. Note, however, that as before, in order to prove Cases 6 and 7, we have to use Lemma 3.23 to assert that the pre state satisfies the assumption of the corresponding case, and have to use Lemma 3.22, instead of Lemma 3.21 to prove the blocking condition. The proof structure is the same as the rest of the cases.

- *Case 2*: Suppose that in the post state, there is exactly one aircraft in maz(σ), and there are at least one aircraft assigned σ in the approach area. As stated above, we only consider the case that the transition is performed for side σ. In the following, we lead to a contradiction to the fact that the transition is preformed for that side. The assumption of the case implies that there are exactly two aircraft on maz(σ) in the pre state. Thus the pre state satisfies the assumption of Case 1. It follows that there is no aircraft assigned σ on the approach in the pre state. This is a contradiction.

- *Case 3*: Suppose that in the post state, there is exactly one aircraft on maz(σ) and there are at least one aircraft in the vertical initiation area of side σ. We only consider the case that the transition is performed for side σ. It implies that there are exactly two aircraft on `maz(`σ`)` in the pre state. Thus the pre state satisfies the assumption of Case 1. It follows that the three conditions of the case hold in the pre state. The bound on the number of aircraft in the vertical initiation area in the post state holds since there is no aircraft in the area in the pre state from the first condition of Case 1, and the transition just adds one aircraft to the area. The condition on the number of assignments in the approach area hold since the same condition holds in the pre state from Case 1 and the transition does not affect the area. Now we prove the blocking condition. Basically, we can prove it using the blocking condition that holds in the pre state, and Lemma 3.21. However, we have to be careful about the matching of the blocking aircraft in the pre state and the post state. First suppose that in the pre state, the first aircraft of `maz(`σ`)` is assigned σ. In this case, the initiation area of the opposite side is blocked by the second aircraft of `maz(`σ`)`, from the blocking condition stated in Case 1. Since the first aircraft of `maz(`σ`)` moves to the vertical initiation area of side σ by the transition, and the area is empty in the pre state from the first condition of Case 1, this aircraft becomes the first aircraft of the vertical initiation area of σ. In the case that the first aircraft of that area is assigned σ, we have to prove for Case 3 that in the post state, the initiation area of the opposite side of σ is blocked by the first aircraft of `maz(`σ`)`. Since the same area is blocked by the second aircraft of `maz(`σ`)` in the pre state, and this aircraft becomes the first aircraft of the zone after the transition, we obtained the required blocking condition. We can analogously prove the blocking condition in the case that the first aircraft of `maz(`σ`)` is assigned the opposite side of σ.

- *Case 4*: Suppose that in the post state, there are at least one aircraft in the vertical initiation area of side σ, and at least one aircraft assigned σ in the approach area. Since the transition does not affect the approach area, there are at least one aircraft assigned σ in the approach area in the pre state as well. We only consider the case that the transition is performed for side σ. We first prove that the vertical initiation area of σ is empty in the pre state. Suppose, for a contradiction, it is not empty. It implies that the pre state satisfies the assumption of Case 3. It in turn follows that there is no aircraft assigned to σ on the approach in the pre state. This is a contradiction. Now we prove the rest of the conditions. The precondition of the transition ensures that `maz(`σ`)` is not empty in the pre state.

First we consider the case that there is exactly one aircraft in `maz(`σ`)` in the pre state. It follows that the pre state satisfies the assumption of Case 2, and thus the conditions of Case 2 hold in that state. The condition of the number of assignments in the approach area hold since the same condition holds in the pre state and the transition does not affect the area. The blocking condition follows from Lemma 3.21, the blocking condition that holds in the pre state, and the fact that the first aircraft of `maz(`σ`)` in the pre state – the specified blocking aircraft in Case 2 – becomes the first aircraft of the vertical initiation area of σ – the specified blocking aircraft in Case 4.

Next we consider the case that there is more than one aircraft in `maz(`σ`)` in the pre state. We prove that this case cannot happen by contradiction. Property 4 that holds in the pre state implies that there are exactly two aircraft in the zone in that state. It follows that the pre state satisfies the assumption of Case 1, and thus there is no aircraft assigned σ on the approach in that state. This is a contradiction.

- *Case 5*: Suppose that in the post state, there are at least two aircraft in the vertical initiation area of σ. We only consider the case that the transition is performed for side σ. It follows that in the pre state, there are at least one aircraft in `maz(`σ`)`, and at least one aircraft in the vertical initiation area

  First we consider the case that there is exactly one aircraft in `maz(`σ`)` in the pre state. It follows that the pre state satisfies the assumption of Case 3, and thus the conditions of Case 3 hold in that state. There are at most two aircraft in the vertical initiation area in the post state, since there are at most one aircraft in the area in the pre state and the transition just adds one aircraft to the area. The condition of the number of assignments in the approach area hold since the same condition holds in the pre state and the transition does not affect the area. We can prove the blocking condition using the matching of the blocking aircraft between the pre state and the post state, and Lemma 3.21 in a way analogous to the proof of Case 4.

  Next we consider the case that there is more than one aircraft in `maz(`σ`)` in the pre state. We can lead to a contradiction using Property 4 and Case 1 in a way analogous to the proof of Case 4.

- *Case 6*: Suppose that in the post state, there are at least one aircraft in the vertical initiation area of side σ, and there is an aircraft that is ready to approach from the opposite side of σ to side σ. From Lemma 3.23, there is an aircraft that is ready to approach from the opposite side of σ to side σ in the pre state as well. We only consider the case that the transition is performed for side σ. The precondition of the transition ensures that there are at least one aircraft on `maz(`σ`)` in the pre state.

We first consider the case that there is exactly one aircraft in `maz(`σ`)` in the pre state. First we prove that the vertical initiation area is empty in the post state. Suppose, for a contradiction, that the area is not empty in the post state. It follows that the pre state satisfies the assumption of Case 3, and thus the initiation area of the opposite side of σ is blocked with respect to σ. This contradicts that there is an aircraft that is ready to approach from the opposite side of σ to side σ in that state. There is no aircraft in `maz(`σ`)` in the post state since there is exactly one in the pre state, and the transition removes it from the zone. Next we prove, by contradiction, that there is no aircraft assigned σ on the approach area in the post state. Suppose, for a contradiction, there are at least one aircraft assigned σ on the approach area in the post state. It follows that the pre state satisfies the assumption of Case 2, and thus the initiation area of the opposite side of σ is blocked with respect to σ. This is a contradiction from the same reason as above. Finally, we prove the blocking condition. The pre state satisfies the assumption of Case 7, and thus the blocking condition stated in Case 7 hold in the pre state The blocking condition follows from this condition and the fact that the first aircraft of `maz(`σ`)` in the pre state becomes the first aircraft in the vertical initiation area of σ in the post state, and thus the specified blocking aircraft matches between the pre state and the post state.

Next we consider the case that there is more than one aircraft in `maz(`σ`)` in the pre state. We prove, by a contradiction, this case cannot happen. Property 4 that holds in the pre state implies that there are exactly two aircraft in `maz(`σ`)` in that state. It implies that the pre state satisfies the assumption of Case 1, and thus the the initiation area of the opposite side of σ is blocked with respect to σ in that state. This contradicts that there is an aircraft that is ready to approach from the opposite side of σ to side σ in the state.

– *Case 7*: Suppose that in the post state, there is one aircraft in `maz(`σ`)`, and there is an aircraft that is ready to approach from the opposite side of σ to side σ. From Lemma 3.23, there is an aircraft that is ready to approach from the opposite side of σ to side σ in the pre state as well. We only consider the case that the transition is performed for side σ. We prove, by contradiction, that the transition cannot be performed for the side in the following. Since the transition removes one aircraft from `maz(`σ`)`, there must be exactly two aircraft in `maz(`σ`)` in the pre state. It follows that the pre state satisfies the assumption of Case 1, and thus the the initiation area of the opposite side of σ is blocked with respect to σ in that state. This contradicts that there is an aircraft that is ready to approach from the opposite side of σ to side σ in the state.

- *In the case of* `HoldingPatternDescend`:
  - *Property 3*: The transition moves an aircraft from `holding3` of one side to `holding2` of the same side. The bound on the number of aircraft in `holding3` holds from the same bound that holds in the pre state. The bound on the number of aircraft in `holding2` follows from the fact that the precondition of the transition ensures that there is no aircraft in `holding3` of the side that an aircraft joins by the transition.
  - *Property 4*: The bound immediately follows from the induction hypothesis since the transition does not affect `maz` zones.
  - *Cases 1 to 7*: The transition does not affect either `maz` zones, or the number of aircraft in the vertical initiation areas. In addition, Lemma 3.23 implies that if there is an aircraft that is ready to approach, then the same condition holds in the pre state. It follows that when we assume the assumption of one of the cases in the pre state, the assumption of the same case is satisfied in the pre state. The conditions of the each case follows from the same condition that hold in the pre state, the fact that the transition does not affect either `maz` zones, the approach initiation area, or the number of aircraft in the vertical initiation areas, and the combination of Lemma 3.21 or 3.22 and the matching of the blocking aircraft between the pre state and the post state.
- *In the case of* `Exit` *or* `Landing`:
  - *Properties 3 and 4*: The bounds immediately follows from the induction hypothesis since the transition does not affect the vertical initiation areas or `maz` zones.
  - *Cases 1 to 7*: Analogous to the case of `HoldingPatternDescend`, if we assume the assumption of one of the cases in the post state, the assumption of the same case is satisfied in the pre state. This is because the transition just removes one aircraft from the approach area and does not affect the other areas. Note that even though the transition does affect the approach zone, the assumption of each case in the post state implies the assumption of the same case in the pre state, since the condition in each assumption that refers to the approach area states that there are at least one

aircraft assigned σ in that area, and there can be more aircraft assigned σ in the pre state, but not less.

The bounds on the number of aircraft in a zone of the initiation area immediately follows from the same bounds that holds in the pre state and the fact that the transition does not affect that area.

The conditions on the number of assignments in the approach area holds since the transition removes one aircraft from that area, but does not add any aircraft to the area.

The blocking condition can be proved using Lemma 3.21 or 3.22.

- *In the case of* `FinalSegment` *or* `Taxiing`: The transition does not affect the initiation areas or the membership of aircraft in the approach area. Thus Properties 3 and 4 immediately follows from the induction hypothesis, and each case of Cases 1 to 6 can be proved in a way analogous to the case of `HoldingPatternDescend`.

□

3.7.4 Proof of Property 2

Now we prove Property 2, $\forall \sigma : \mathit{Side}, \mathit{actual}(\sigma) \leq 2$. We use Lemma 3.26 as well as the properties we have proved so far in order to conduct a case analysis in the proof. The formal definition of the property is as follows.

Theorem 3.27. (Property 2) *For any reachable state of SATS and side σ, $\mathit{actual}(\sigma) \leq 2$.*

Proof. We prove the bound for the initiation area of an arbitrarily chosen side σ. First, if there is an aircraft in `lez(`σ`)`, Properties 5 and 6 imply that there is only one aircraft in the initiation area of σ – the first aircraft of `lez(`σ`)`. Thus $\mathit{actual}(\sigma) = 1$ in this case.

Next we consider the case that `lez(`σ`)` is empty. We consider three cases:

split the case depending on the number of aircraft in `maz(`σ`)`. There are at most two aircraft in the zone from Property 4.

- $|\mathtt{maz}(\sigma)| = 2$: The state satisfies the assumption of Case 1 of Lemma 3.26. It implies that both `holding3`(σ) and `holding2`(σ) are empty. Thus the bound holds.

- $|\mathtt{maz}(\sigma)| = 1$: If both `holding3`($\sigma$) and `holding2`($\sigma$) are empty, then the bound trivially holds. If either `holding3`(σ) or `holding2`(σ) is non-empty, then the state satisfies the assumption of Case 3 of Lemma 3.26. It implies that $|\mathtt{holding2}(\sigma)| + |\mathtt{holding3}(\sigma)| \leq 1$. Thus the bound holds.

- $|\mathsf{maz}(\sigma)| = 0$: Since both `maz(`$\sigma$`)` and `lez(`$\sigma$`)` are empty in the state, the only position at which aircraft can be located is the vertical approach area. From Property 3, $|\mathtt{holding2}(\sigma)| \leq 1$ and $|\mathtt{holding3}(\sigma)| \leq 1$. Thus $actual(side) \leq 2$.

□

3.8 Proof using the PVS theorem prover

In this section, we explain how we conducted our mechanical theorem-proving process for the proofs presented in this Chapter. We also discuss how we could improve the production speed and also ease the difficulties of such mechanical proof verification using theorem-provers, based on the experience we obtained through the process.

3.8.1 Steps we have taken to use a theorem-prover

We took the following process to conduct a machine-supported verification of the proofs presented in this Chapter.

1. First, we translated TIOA code to PVS code by using an automatic translator [8], which is written by Hongping Lim in our research group. As we discussed in Section 2.3, we also input some supplemental PVS files that define auxiliary recursive functions and types, and functions that use those. The translator looks up the vocabulary file, and find the functions whose types are declared, but whose definitions are not specified, in the vocabulary file. When the translator finds such functions, it complements the definitions of them by inserting the actual definitions of those functions from the supplemental PVS files. The translator's output uses a framework primarily developed for a library of *theorem-proving strategies* for PVS, called TAME, written by Myla Archer [1]. Theorem-proving strategies are tools that can be used in an interactive theorem-proving: they combine basic commands of a theorem-prover by executing these commands in a specific order using some control flow mechanics such as `if then`. TAME is specifically designed for supporting theorem-proving of invariants of timed I/O automata and the abstraction relations between automata. This framework includes definitions for the basic structures of timed I/O automata, and it also has basic theories for establishing an induction over the number of transitions of an automaton, and for proving an abstraction relation between automata.

   (you may consider them as some type of macros that combine basic PVS commands)

2. Next, we stated the properties we want to prove, in PVS code. As we have seen in Section

3.3, we need to state not only main properties we want to prove, but also some auxiliary lemmas needed to prove the main properties.

3. Finally, we conducted a mechanical verification of the hand-written proofs using the interactive theorem-prover of PVS. During this theorem-proving process, we found additional auxiliary lemmas that could be used to avoid repetitive arguments in the theorem-proving. In such cases, we added those lemmas to the PVS code, proved them separately from the proof of main properties or other auxiliary lemmas, and used them directly in the proof.

3.8.2 Theorem-proving process in PVS

Here we informally explain how theorem-proving is conducted in PVS. It is important to (at least roughly) understand how PVS works in order to understand the suggestion for improving the verification process discussed in Section 3.8.3.

PVS has a "growing tree structure" for the proof process: A proof start with just the top of the tree, which will grow during the verification process. This top node contains *the assumptions*, the logical formulas we assume, and *the conclusion*, the logical formula we want to prove using the assumptions. The proof process progresses by manipulating these given formulas, for example, by applying the *modus pones* rule to formulas, or by using substitution for a specific term that appear in one formula, using the equality specified in another formula. These basic manipulations are given by interactions with the prover using *prover commands*. The result of such a prover command creates a child of the node for which the command is executed. This child contains the formulas after the manipulation by that command for the parent's formulas. In this way, the "proof tree" grows downward.

When we have a formula $A \wedge B$ in the conclusion, we can split the verification process into two "branches", where in one branch, the new proof goal is to prove A, and in the other branch, the new goal is B. In PVS, this split creates a branching in the proof tree: after a split in some node with the goal $A \wedge B$, that node has two children, where one child has the new goal A, and the other has B. By this branching, the proof tree grows both vertically and horizontally.

All formulas in the assumptions and the conclusion are numbered simply from 1 in the order they appear (See Fig. 3.6 to see one typical node in a PVS proof tree. Here, we have two formulas for the assumptions, and one formula for the conclusion. Formulas in the assumptions are numbered from -1.) We may also label the formulas using some prover commands, and use those labels to refer to formulas, instead of formula numbers. These labels can be some arbitrary text, such as "Assumption 1", or "Induction hypothesis 3" (see Fig. 3.7). The advantage of using labels over using formula numbers to refer to formulas is as follows. The formula numbers are

```
{-1}  in_queue?(ac!1, q!1)
{-2}  ac!1`mahf = side!1
  |-------
{1}   assigned(q!1, side!1) >= 1
```

Figure 3.6: A typical node in a PVS proof tree

```
[-1,(Assumption 1)]
      in_queue?(ac!1, q!1)
[-2,(Assumption 2)]
      ac!1`mahf = side!1
  |-------
{1,(Conclusion)}
      assigned(q!1, side!1) >= 1
```

Figure 3.7: Use of labels

determined just by the order in which the formulas appear in a specific node. Thus, for instance, the second formula in one specific node N may become the third formula in its child node, since, for example, the command executed in N introduces one new formula to the child node. Fig. 3.8 shows the formulas in a child of the node depicted in Fig. 3.7, where one new formula (which appear at the top) is inserted by the `lemma` command that inserts a formula stated as a lemma. In this child node, the formula labeled sf Assumption 2 is now have the formula number -3. Labels are useful in that, once a formula is labeled at some branch, we can refer to the same formula by its label regardless of insertion of other formulas.

Some of the prover commands take formula numbers as arguments. For example, the `replace` command that is used for substitution of a particular term using an equality is used in the form `(replace i j)`. When the command `(replace i j)` is executed in some node of the prove tree, the prover first checks if `i` and `j` are valid formula numbers for the formulas in that node, and also checks if the formula numbered `i` has form `x = y`. If the command does not satisfy these conditions, a prover simply skips this command. Otherwise, it creates a child of that node,

```
{-1}  FORALL (side: Side, z: Zone): length(z) >= assigned(z, side)
[-2,(Assumption 1)]
      in_queue?(ac!1, q!1)
[-3,(Assumption 2)]
      ac!1`mahf = side!1
  |-------
[1,(Conclusion)]
      assigned(q!1, side!1) >= 1
```

Figure 3.8: Use of labels (when another formula is inserted to the assumptions)

and the child contains the formulas in which every occurrence of x in the parent's formulas is replaced by y. We may also use labels to refer to the formulas, instead of using formula numbers like i and j in this example.

PVS considers the proof obligation for one specific node is verified in any of the following cases: 1. the conclusion formula becomes true. 2. the assumptions contains false (since we are assuming that all assumptions are true, one false that appear in an assumption gives a false implication), and 3. the same formula appears in an assumption and the conclusion. When the proof obligations of all leaves of the tree are verified, the theorem-proving for the theorem that is stated in the top of the tree is successfully finished.

PVS also has some commands that make use of decision procedures, rather than just doing simple logical or algebraic manipulations. Those commands combine simple manipulation commands by executing these simple commands in a order determined by some decision procedure. Among the commands that use decision procedures, a grind command has a decision procedure that basically "tries everything that PVS can" to finish the verification of the node in which grind is executed. Since the problem that grind deals with is generally undecidable, this command is not guaranteed to terminate.

The collection of prover commands executed during a proof process constructs a tree of commands that represents a proof tree for that proof. This tree of commands is saved as a text file, called a *proof script*, using an S-expression representation for that tree.

In addition, PVS has a graphical feature that generates a picture of the proof tree of the current theorem-proving process. This depicted tree is useful in that we can easily recognize which proof branch we are currently verifying.

3.8.3 Our experience obtained by this case study and possible improvement for mechanical proof processes using theorem-provers

In this section, we describe our experience in using PVS for a mechanical verification of our proof, and discuss how we could have improved the mechanical proof process using theorem-provers based on that experience.

Our experience using PVS

In our case study, the whole verification of the proofs presented in this chapter took about four months. Though the author, who conducted the mechanical verification, was doing course work during the same time period, more than 20 hours were devoted to theorem-proving using PVS in each week. We consider that the following are the three main reasons this large amount of time had to be devoted to conducting this project. First, some theorems and lemmas (most notably

Lemma 3.26) have a lengthy proof, even in the hand-written version. For this respect, we cannot really improve the production speed of the machine-verified proof, unless we can improve the hand-written proofs.

Second, the author was learning the PVS theorem-prover along with the verification process. Though I have some experience in using PVS at the point when this theorem-proving project for the SATS protocol was started, I was still leaning PVS at that point, especially in terms of an efficient theorem-proving; for example, which prover command works best (in terms of both result of the command and time to execute it) for what case. Even if a choice of the command does not change the execution time of one command, a collection of them in a large proof does make a drastic difference in the efficiency, for example, the running time of the whole proof process.

Third, verifying a large proof, like the one for Lemma 3.26, in PVS slows down the verification process in several aspects:

1. For a small, simple proof obligation, a decision procedure command, such as grind, was so powerful that we could basically finish the verification of that small obligation by using just one grind command. Using the grind command in an earlier stage of the proof took longer time (since grind presumably made some unnecessary attempts because of the lack of supplemental information) than in the later stage where supplemental information (formulas) needed to prove the obligation had been provided by human interaction. Using grind in an early stage was acceptable (with respect to time-efficiency) for small proofs since the places in which we used grind were few in such proofs. In a large proof, we had many branches whose proof obligations were trivial to a human. We wanted to use grind to finish those branches. The problem here was that since we had many branches that we wanted to use grind, if we had used grind commands without adding a supplemental information that might have helped a decision procedure, the running time of the whole proof would become unacceptably long. This was a serious problem since we often needed to re-run the whole proof. For example, we found some auxiliary lemmas that helped reducing the redundancy in the proof tree. In such a case, we had to exit the current proof process once and stated those lemmas before using them. To reduce the re-running time of a large proof, we basically had to add sufficient information for a decision procedure (or sometimes "hide" unnecessary information that may confuse a decision procedure) before executing grind. This process of adding supplemental information would reduced the running time of the proof, but required extra human interaction.

2. Theorem-proving process for a large proof made a proof tree large. When dealing with

such a large tree, we experienced that each execution of a prover command, especially for one that uses a decision procedure (such as `grind`), became slower (it sometimes took more than twice as long as in the case when the command was executed in a smaller proof tree). It might be because PVS has to store more internal information for a large proof than for a small proof. This slow-down reduced possible places where commands with decision procedures could be used in an earlier stage of the proof, in addition to the reason discussed in the above Reason 1.

3. Since we had to deal with a large proof tree for a large proof, a graphical representation of that tree made by PVS became large. This construction of the graphical tree took long time (up to 5 minutes depending on the size of the tree), and the tree was updated every time we executed a new command (it sometimes took around 20 seconds). This made it infeasible to use the graphical feature. Thus, we had to rely on only the text information to comprehend what branch of the proof tree we were verifying now. We sometimes lost a big picture of the proof in such situations, and thus took long time to grasp what assumed formulas were really needed and what were unnecessary information in order to prove the conclusion of that branch.

How we could have improved the production speed of the machine-verified proofs

As we have discussed earlier in this subsection, in our case study, the main difficulty to conduct a theorem-proving came up when verifying large proofs. An easy and effective way to resolve the above mentioned problems for verifying a large proof is to decompose the proof into small sub-proofs using lemmas. For example, for mechanical verification of the proof for Lemma 3.26, we could state several lemmas each of which represents an inductive step of the proof for one specific transition. Since there are thirteen transitions, we would state thirteen lemmas. We could also have even smaller sub-proofs: For instance, the proof of the inductive step for `VerticalApproachInitiation` is lengthy. Thus we could have the auxiliary lemmas `VAI.Case1` - `VAI.Case7`, each of which represents the proof obligation for one specific case of the inductive step for `VerticalApproachInitiation`. Then, the verification of the original large proof can be done by applying these lemmas to verify the corresponding branches in the original proof.

By verifying one large proof as a collection of sub-proofs, we can make the size of a PVS proof tree for one sub-proof small. In this way, we can increase the number of cases when we can use the `grind`-type of command in an earlier stage of the verification, and also increase the accessibility to the graphical feature for the proof tree.

How we could have improved the stability of machine verified proofs for modification

We sometimes came across the situation in which we proved a theorem using some auxiliary lemmas that have not yet been proved, and found that the statement of the lemmas need some minor changes or additional minor assumptions. In such a case, we modified the lemmas as required, and proved it. Then, of course, we had to re-run the proof verification for the theorem that used these lemmas, in order to check the theorem-proving was still sound. This process sometimes required some additional verification steps that did not appear in the original verification, in order to verify the modified part.

To gain the maximum stability of machine verified proofs for such modifications, we could make use of labels when we refer to formulas. This is because the modification of the lemma statement may result in inserting some additional formulas in a node for which the lemma is applied. For example, suppose we have three formulas A_1, A_2, B in the assumption before the modification of the lemma, and we have four formulas A_1, A_2, A_3, B after the modification. Since the formulas are indexed simply from the beginning of a sequence of formulas, formula B has the formula number *three* before the modification, but the number for B becomes *four* after the modification. If we are referring to the formulas by number, and, for example, we are using B for the substitution of formulas `(replace 3 1)`, such substitution command in the original proof script may result in an invalid command (since the formula number 3 now refers to A_3.) In such a case, PVS simply skips executing that command, and keep executing the remaining commands in a given proof script. Of course, the proof verification may fail to finish rerunning the original proof script successfully since one command that is needed, `(replace 3 1)`, has been skipped.

To avoid such situations, we can make use of labels for formulas. If we had given a label `formula_B` for formula B, and `formula_A1` for A_1 in the above situation, and we had referred B by that label (`(replace formula_B formula_A1)`), the problem would have been avoided.

The TAME strategies [1] could be used to help a user to label formulas automatically. These strategies help the user by setting up the proof obligation of an induction proof, with automated labeling. The strategies also have features with which, whenever new formulas are introduced by a command defined in the strategies, these formulas are automatically labeled.

Strategies for PVS that could have been used to support theorem-proving of large proofs

The TAME strategies [1] could be used to support theorem-proving of large proofs in various ways. First, as we explained above, TAME helps the user to label the formulas. TAME also has strategies for applying lemmas that combine basic steps (inserting a formula stated as a lemma and instantiating this formula for the specific case we are currently dealing with). By reducing the interaction with PVS required for the user, these strategies can help the user to have sub-proofs we have discussed above.

A new type of strategies that could reduce the interaction with PVS are strategies for date-type specific simplifications. In our case study, we had many cases in which we had to do some simplification for queue structures that are trivial to a human. We stated lemmas that represent the equalities between queues before and after some simplifications, and used these lemmas to simplify queues. However, if we have had some simplification strategies specific to queues, then we would have required less amount of human interaction with PVS. Myla Archer, the developer of TAME, is currently writing these simplification strategies for major data structures, such as queues. However, we may sometimes find other simplifications that would come up often in the proof, than the simplifications supported by the existing strategies. This implies that the close connection between the user of the strategies and the developer of them is required (for example, we should avoid the situation that we have to wait for one week for the strategy writer to write a new simplification strategy). A straightforward way to resolve this problem is for the user to become also the writer, though this requires an additional learning effort for the user. To do so, a more experienced writer of the strategies can give some "template" for simplification strategies so that efforts for a new writer are minimized.

Chapter 4

Timed I/O automata framework

In this chapter, we explain preliminaries for the timed I/O automata framework. We also introduce simulation relation and refinement proof techniques for the framework.

4.1 Timed I/O automata

In this section, we introduce the mathematical framework we use in Chapter 5, *timed I/O automata*. All material of this section is taken from [6].

4.1.1 Functions

In this section, we present basic notations and operations for functions that are used in the rest of this chapter.

If f is a function, then we denote the domain and range of f by $dom(f)$ and $range(f)$, respectively. If S is a set, then we write $f \lceil S$ for the restriction of f to S, that is, the function g with $dom(g) = dom(f) \cap S$ such that $g(c) = f(c)$ for each $c \in dom(g)$.

If f is a function whose range is a set of functions and S is a set, then we write $f \downarrow S$ for the function g with $dom(g) = dom(f)$ such that $g(c) = f(c) \lceil S$ for each $c \in dom(g)$. The restriction operation $\rightarrow$ is extended to sets of functions by pointwise extension.

4.1.2 Time

In this thesis, a *time axis* T is the set R of real numbers. We define $\mathsf{T}^{\geq 0} = \{t \in \mathsf{T} | t \geq 0\}$

An *interval* J is a nonempty, convex subset of T. We denote intervals as usual: $[t_1, t_2] = \{t \in T | t_1 \leq t \leq t_2\}$, $[t_1, t_2) = \{t \in T | t_1 \leq t < t_2\}$, etc. An interval J is *left-closed* (*right-closed*) if it has a minimum (resp., maximum) element and is *left-open* (*right-open*) otherwise. It is *closed* if it is both left-closed and right-closed. For $K \subseteq \mathsf{T}$ and $t \in \mathsf{T}$, we define $K + t = \{t' + t | t' \in K\}$.

Similarly, for a function f with domain K, we define $f+t$ to be the function with domain $K+t$ satisfying, for each $t' \in K+t, (f+t)(t') = f(t'-t)$.

In some definitions and theorems in this chapter we assume that the relation $\leq$ on R extends to a relation on $\mathsf{R} \cup \{\infty\}$ such that $\infty \leq \infty$ and for all $t \in \mathsf{R}, t < \infty$.

4.1.3 Static and dynamic types

We assume a universal set V of *variables.* A variable represents a location within the state of a system. For each variable v, we assume both a *(static) type*, which gives the set of values it may take on, and a *dynamic type*, which gives the set of trajectories it may follow. Formally, for each variable v we assume the following:

1. $\mathit{type}(v)$, the *(static) type* of v. This is a nonempty set of values.

2. $\mathit{dtype}(v)$, the *dynamic type* of v. This is a set of functions from left-closed intervals of T to $\mathit{type}(v)$ that satisfies the following properties.

   (a) *Closure under time shift*: For each $f \in \mathit{dtype}(v)$ and $t \in \mathsf{T}$, $f+t \in \mathit{dtype}(v)$.

   (b) *Closure under subinterval*: For each $f \in \mathit{dtype}(v)$ and each left-closed interval $J \subseteq \mathit{dom}(f)$, $f \lceil J \in \mathit{dtype}(v)$.

   (c) *Closure under pasting*: Let $f_0 f_1 \cdots$ be a sequence of function in $\mathit{dtype}(v)$ such that, for each nonfinal index i, $\mathit{dom}(f_i)$ is right-closed and $\max(\mathit{dom}(f_i)) = \min(\mathit{dom}(f_{i+1}))$. Then the function f defined by $f(t) = f_i(t)$, where i is the smallest index such that $t \in \mathit{dom}(f_i)$, is in $\mathit{dtype}(v)$.

4.1.4 Trajectories

In this subsection, we define the notion of a *trajectory*, define operations on trajectories, and prove simple properties of trajectories and their operations. A trajectory is used to model the evolution of a collection of variables over an interval of time.

Basic definitions

Let V be a set of variables. A *valuation* v for V is a function that associates with each variable $v \in V$ a value in $\mathit{type}(v)$. We write $\mathit{val}(V)$ for the set of valuations for V. Let J be a left-closed interval of T with left endpoint equal to 0. Then a *J-trajectory* for V is a function $\tau : J \rightarrow \mathit{val}(V)$, such that for each $v \in V$, $\tau \downarrow v \in \mathit{dtype}(v)$. A *trajectory* for V is a J-trajectory for V, for any J. We write $\mathit{trajs}(V)$ for the set of all trajectories for V. If Q is a set of valuations for some set V of variables, we write $\mathit{trajes}(Q)$ for the set of all trajectories whose range is a subset of Q.

A trajectory for V where $V = \emptyset$ is simply a function from a time interval to the special function with the empty domain. Thus, the only interesting information represented by such a trajectories is the length of the time interval that constitutes the domain of the trajectory. We use trajectories over the empty set of variables when we wish to capture the amount of time-passage. but abstract away the evolution of variables.

A trajectory for V with domain $[0, 0]$ is called a *point* trajectory for V. If v is a valuation for V then $\wp(\mathsf{v})$ denotes the point trajectory for V that maps 0 to v. We say that a J-trajectory is *finite* if J is a finite interval, *closed* is J is a (finite) closed internal, *open* if J is a right-open interval, and *full* if $J = \mathsf{T}^{\geq 0}$.

If τ is a trajectory then $\tau.ltime$, the *limit time* of τ, is the supremum of $dom(\tau)$. We define $\tau.fval$, the *first evaluation* of τ, to be $\tau(0)$, and if τ is closed, we define $\tau.lval$, the *last valuation* of τ, to be $\tau(\tau.ltime)$. For τ a trajectory and $t \in \mathsf{T}^{\geq 0}$, we define

$$\begin{aligned} \tau \trianglelefteq t &= \tau\lceil[0, t], \\ \tau \triangleleft t &= \tau\lceil[0, t), \\ \tau \trianglerighteq t &= (\tau\lceil[t, \infty)) - t. \end{aligned}$$

By convention, we also write $\tau \trianglelefteq \infty = \tau$ and $\tau \triangleleft \infty = \tau$.

Prefix ordering

Trajectory τ is a *prefix* of trajectory υ, denoted by $\tau \leq \upsilon$, if τ can be obtained by restricting υ to a subset of its domain. Formally, it τ and υ are trajectories for V, then $\tau \leq \upsilon$ iff $\tau = \upsilon\lceil dom(\tau)$. Alternatively, $\tau \leq \upsilon$ iff there exists a $t \in \mathsf{T}^{\geq 0} \cup \{\infty\}$ such that $\tau = \upsilon \trianglelefteq t$ or $\tau = \upsilon \triangleleft t$. If T is a set of trajectories for V, then $pref(T)$ denotes the *prefix closure* of T, defined by

$$pref(T) = \{\tau \in trajs(V) | \exists \upsilon \in T : \tau \leq \upsilon\}$$

The following theorem gives a simple domain-theoretic characterization of the set of trajectories over a given set V of variables:

Lemma 4.1. *(Lemma 3.4 of [6]) Let V be a set of variables. The set $trajs(V)$ of trajectories for V, together with the prefix ordering $\leq$, is an algebraic cpo. Its compact elements are the closed trajectories.*

Concatenation

The concatenation of two trajectories is obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Formally, suppose τ and τ' are trajectories for V, with τ closed. Then the *concatenation* $\tau \frown \tau'$ is the function given by

$$\tau \frown \tau' = \tau \cup (\tau' \lceil (0, \infty) + \tau.ltime).$$

The following lemma shows the close connection between concatenation and the prefix ordering.

Lemma 4.2. *(Lemma 3.5 of [6]) Let τ and υ be trajectories for V with τ closed. Then*

$$\tau \leq \upsilon \Leftrightarrow \exists \tau' : \upsilon = \tau \frown \tau'.$$

We extend the definition of concatenation to any (finite or countably infinite) number of arguments. Let $\tau_0 \tau_1 \cdots$ be a (finite or infinite) sequence of trajectories such that τ_i is closed for each nonfinal index i. Define trajectories $\tau'_0, \tau'_1, \cdots$ inductively by

$$\begin{aligned} \tau'_0 &= \tau_0, \\ \tau'_{i+1} &= \tau'_i \frown \tau_{i+1} \text{ for nonfinal } i \end{aligned}$$

Lemma 4.2 implies that for each nonfinal $\tau'_i \leq \tau'_{i+1}$. We define the *concatenation* $\tau_0 \frown \tau_1 \frown \cdots$ to be the limit of the chain $\tau'_0 tau'_1 \cdots$; existence of this limit follows from Lemma 4.1.

4.1.5 Hybrid Sequences

In this subsection, we introduce the notion of a *hybrid sequence*, which is used to model a combination of changes that occur instantaneously and changes that occur over the intervals of time. Our definition is parameterized by a set A of *actions*, which are used to model instantaneous changes and instantaneous synchronizations with the environment, and a set V of variables, which are used to model changes over intervals of time. We also define some special kinds of hybrid sequences and some operations on hybrid sequences, and give basic properties.

Basic Definitions

Fix a set A of actions and a set V of variables. An (A, V)*-sequence* is a finite or infinite alternating sequence $\alpha = \tau_0 a_1 \tau_1 \cdots$, where

1. each τ_i is a trajectory in $trajs(V)$,

2. each a_i is an action in A,

3. if α is a finite sequence, then it ends with a trajectory, and

4. if τ_i is not the last trajectory in α, then τ_i is closed.

A *hybrid sequence* is an (A, V)-sequence for some A and V.

If α is a hybrid sequence, with notation as above, then we define the *limit time* of α, $\alpha.ltime$, to be $\sum_i \tau_i.ltime$. A hybrid sequence α is defined to be *closed* if α is a finite sequence and its final trajectory is closed.

For any hybrid sequence α, we define the *first valuation* of α, $\alpha.fval$ to be $head(\alpha).fval$. Also, if α is closed, we define the *last valuation* of α, $\alpha.lval$, to be $last(\alpha).lval$, that is, the last valuation in the final trajectory of α.

If α is a closed (A, V)-sequence, where $V = \emptyset$ and $\beta \in trajs(\emptyset)$, we call $\alpha \frown \beta$ a *time-extension* of α.

Prefix Ordering

We say that (A, V)-sequence $\alpha = \tau_0 a_1 \tau_1 \cdots$ is a *prefix* of (A, V) sequence $\beta = \upsilon_0 b_1 \upsilon_1 \cdots$, denoted by $\alpha \leq \beta$, provided that (at least) one of the following holds:

1. $\alpha = \beta$.

2. α is a finite sequence ending in some τ_k; $\tau_i = \upsilon_i$ and $a_{i+1} = bi + 1$ for every i, $0 \leq i \leq k$; and $\tau_k \leq \upsilon_k$.

Similar to the set of trajectories over V, the set of (A, V)-sequence is also as algebraic cpo.

Lemma 4.3. *(Lemma 3.6 of [6]) Let V be the set of variables and A a set of actions. The set of (A, V)-sequences, together with the prefix ordering $\leq$, is an algebraic cpo. Its compact elements are the closed (A, V)-sequences.*

Concatenation

Suppose α and α' are (A, V)-sequences with α closed. Then the *concatenation* $\alpha \frown \alpha'$ is the (A, V)-sequence given by

$$\alpha \frown \alpha' = init(\alpha)(last(\alpha) \frown head(\alpha'))tail(\alpha').$$

(Here, *init*, *last*, *head*, and *tail* are ordinary sequence operations.)

Lemma 4.4. *(Lemma 3.7 of [6]) Let α and β be (A, V)-sequence with α closed. Then*

$$\alpha \leq \beta \Leftrightarrow \exists \alpha' : \beta = \alpha \frown \alpha'$$

As we did for trajectories, we extend the concatenation definition for (A, V)-sequences to any finite or infinite number of arguments. Let $\alpha_0\alpha_1\cdots$ be a finite or infinite sequence of (A, V)-sequence such that α_i is closed for each nonfinal index i. Define (A, V)-sequences α'_0, α'_1, $\cdots$ inductively by

$$\alpha'_0 = \alpha_0,$$
$$\alpha'_{i+1} = \alpha'_i \frown \alpha_{i+1} \text{ for nonfinal } i$$

Lemma 4.4 implies that for each nonfinal i, $\alpha'_i \leq \alpha'_{i+1}$. We define the *concatenation* $\alpha_0 \frown \alpha_1 \cdots$ to be the limit of the chain $\alpha'_0\alpha'_1\cdots$; existence of this limit is ensured by Lemma 4.3.

Restriction

Let A and A' be sets of actions and let V and V' be sets of variables. The (A', V')-*restriction* of an (A, V)-sequence α, denoted by $\alpha\lceil(A', V')$, is obtained by first projecting all trajectories of α on the variables in V', then removing the actions not in A', and finally concatenating all adjacent trajectories. Formally, we define the (A', V')-restriction first for closed (A, V)-sequences and then extend the definition to arbitrary (A, V)-sequences using a limit construction. The definition for closed (A, V)-sequence is by induction on the length of those sequences:

$$\tau\lceil(A', V') = \tau \downarrow V' \text{ if } \tau \text{ is a single trajectory,}$$
$$\alpha a \tau\lceil(A', V') = \begin{cases} (\alpha\lceil(A', V'))a(\tau \downarrow V') & \text{if } a \in A' \\ (\alpha\lceil(A', V')) \frown (\tau \downarrow V') & \text{otherwise} \end{cases}$$

It is easy to see that the restriction operator is monotone on the set of closed (A, V)-sequences. Hence, if we apply this operation to a directed set, the result is again a directed set. Together with Lemma 4.3, this allows us to extend the definition of restriction to arbitrary (A, V)-sequences by

$$\alpha\lceil(A', V') = \sqcup\{\beta\lceil(A', V')|\beta \text{ is a closed prefix of } \alpha\}.$$

4.1.6 Timed Automata

A timed automaton is a state machine whose states are divided into *variables* and that has a set of discrete *actions*, some of which may be internal and some external. The state of a timed automation may change in two ways: by *discrete transitions*, which change the state atomically, and by *trajectories*, which describe the evolution of the state over intervals of time. The evolution described by a trajectory may be described by continuous or discontinuous functions. Formally, a timed automaton (TA) $A = (X, Q, \Theta, E, H, D, T)$ consists of the following:

- A set X of *internal variables*.

- A set $Q \subseteq val(X)$ of *states*.

- A nonempty set $\Theta \subseteq Q$ of *start states*.

- A set E of *external actions* and a set H of *internal actions*, disjoint from each other.

- A set $D \subseteq Q \times A \times Q$ of *discrete transitions*. We say that a is *enabled* in x if $(x, a, x') \in D$ for some x'.

- A set $T \subseteq trajs(Q)$ of trajectories. Given a trajectory $\tau \in T$, we denote $\tau.fval$ by $\tau.fstate$ and, if τ is closed, we denote $\tau.lval$ by $\tau.lstate$.
  we require that the following axioms hold:
  T0 (Existence of point trajectories). If $x \in Q$, then $\wp(x) \in T$.
  T1 (Prefix closure). For every $\tau \in T$ and every $\tau' \leq \tau$, $\tau' \in T$.
  T2 (Suffix closure). For every $\tau \in T$ and every $t \in dom(\tau)$, $\tau \trianglerighteq t \in T$.
  T3 (Concatenation closure). Let $\tau_0\tau_1\cdots$ be a sequence of trajectories in T such that, for each nonfinal index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. Then $\tau_0 \frown \tau_1 \frown \cdots \in T$.

Notation: We denote the components of a TA A by X_A, Q_A, Θ_A, E_A, etc. We sometimes omit these subscripts, where no confusion seems likely.

In this thesis, we specify sets of trajectories using differential equations and inclusions. In doing so, we use the following notations. Suppose the time domain T is R, τ is a (fixed) trajectory over some set of variables V, $v \in V$, and e is an integrable function containing variables from V. Then we say that τ satisfies

$$d(v) = e$$

if, for every $t1, t_2 \in dom(\tau)$ such that $t_1 \leq t_2$, $v(t_2) = v(t_1) + \int_{t_1}^{t_2} e(t')dt'$.

We generalize this notation to handle inequalities as well as equalities. We say that τ satisfies

$$e \leq d(v)$$

if, for every $t1, t_2 \in dom(\tau)$ such that $t_1 \leq t_2$, $v(t_1) + \int_{t_1}^{t_2} e(t')dt' lev(t_2)$. $t \in dom(\tau)$, $v(0) + \int_0^t e(t')dt' \leq v(t)$, and τ satisfies

$$d(v) \leq e$$

if, for every $t1, t_2 \in dom(\tau)$ such that $t_1 \leq t_2$, $v(t_2) lev(t_1) + \int_{t_1}^{t_2} e(t')dt'$.

Conventions for automata specifications: In the examples of this thesis, we assume the time axis T to be R and specify timed automata by using the TIOA language presented in [3].

In this thesis, the set of states of an automaton equals the set of all valuations of its state variables. A type $\mathsf{AugmentedReal}$ denotes $\mathsf{R} \cup \{\infty\}$.

The transitions are specified in precondition-effect style, as in usual I/O automata.

In this thesis, the trajectories are specified using a combination of differential equations and inclusions, and stopping conditions. A trajectory belongs to the set of legal trajectories of an automaton if it satisfies the stopping condition expressed by the **stop when** clause and the equations or inequalities in the **evolve** clause. The stopping condition is satisfied by a trajectory if the only state in which the condition holds is the last state of that trajectory. That is, time cannot advance beyond the point where the stopping condition is true. The **evolve** clause specifies the differential equations and inclusions that must be satisfied by the trajectories. we write $\mathtt{d}(v) = e$ for $d(v) = e$, $\mathtt{d}(v) \leq e$ for $d(v) \leq e$, and $e \leq \mathtt{d}(v)$ for $e \leq d(v)$. We assume that the evolution of each variable follows a continuous function throughout a trajectory. This implies that the value of a discrete variable is constant throughout a trajectory: time-passage does not change the value of discrete variables.

Execution and traces

We now define execution fragments, executions, trace fragments, and traces, which are used to describe automaton behavior. An *execution fragment* of a timed automaton A is an (A, V)-sequence $\alpha = \tau_0 a_1 tau_1 a_2 \cdots$, where (1) each τ_i is a trajectory in T and (2) if τ_i is not the last trajectory in α, then $(\tau_i.lstate, a_{i+1}, \tau_{i+1}.fstate) \in D_A$. An execution fragment records what happens during a particular run of a system, including all the instantaneous, discrete state changes and all the changes to the state that occur while time advances. We write $frags_A$ for the set of all execution fragments of A.

If α is an execution fragment, with notation as above, then we define the *first state* of α, $\alpha.fstate$, to be $\alpha.fval$. An *execution fragment* of a timed automaton A *from* a state x of A is an execution fragment of A whose start state is x. We write $frags_A(x)$ for the set of execution fragments of A from x. An execution fragment α is defined to be an *execution* if $\alpha.fstate$ is a start state, that is $\alpha.fstate \in \Theta_A$. We write $execs_A$ for the set of all executions of A. If α is a closed (A, V)-sequence, then we define the *last state* of α, $\alpha.lstate$, to be $\alpha.lval$. A state of A is *reachable* if it is the last state of some closed execution of A. We write $reachable(A)$ for the set of all reachable states of A. A property that is true for all reachable state of an automaton is called an *invariant* of the automaton.

Execution fragments are closed under countable concatenation (Lemma 4.7 of [6]).

Lemma 4.5. *Let $\alpha_0 \alpha_1 \cdots$ be a finite or infinite sequence of execution fragments of A such that,* *for each nonfinal index, i, α_i is closed and $\alpha_i.lstate = \alpha_{i+1}.fstate$. Then $\alpha_0 \frown \alpha_1 \cdots$ is an execution fragment of A.*

The external behavior of a timed automaton is captured by the set of "traces" of its execution fragments, which record external actions and the trajectories that describe the intervening passage of time. A trace consists of alternating external actions and trajectories over the empty set of variables, $\emptyset$; the only interesting information contained in these trajectories in the amount of time that elapses.

Formally, if α is an execution fragment, then the *trace* of α, denoted by $trace(\alpha)$, is the $(E, \emptyset)$-restriction of α, $\alpha\lceil(E, \emptyset)$. A *trace fragment* of a timed automaton A *from* a state x of A is the trace of an execution fragment of A whose first state is x. We write $tracefrags_A(x)$ for the set of trace fragment of A from x. Also, we define a *trace* of A to be a trace fragment from a start state, that is, the trace of an execution of A, and write $traces_A$ for the set of traces of A.

4.1.7 Timed I/O Automata

Timed I/O automata is a refined version of timed automata of Section 4.1.6 by distinguishing between input and output actions.

A *timed I/O automaton* (TIOA) A is a tuple (B, I, O) where

- $B = (X, Q, \Theta, E, H, D, T)$ is a timed automaton.
- I and O partition E into *input* and *output actions*, respectively.
- The following additional axioms are satisfied:
  E1 (Input action enabling). For every $x \in Q$ and every $a \in I$, there exists $x' \in Q$ such that $(x, a, x') \in D$.
  E2 (Time-passage enabling.) For every $x \in Q$, there exists $\tau \in T$ such that $\tau.fstate = x$ and either
  1. $\tau.ltime = \infty$ or
  2. τ is closed and some $l \in L$ is enabled in $\tau.lstate$.

Notation: We denote the components of a TA A by B_A, I_A, O_A, X_A, Q_A, Θ_A, E_A, etc. We sometimes omit these subscripts, where no confusion seems likely. We abuse notation slightly by referring to a TIOA A as a TA when we intend to refer B_A.

An *execution fragment*, *execution*, *trace fragment*, or *trace* of a TIOA A is defined to be an execution fragment, execution, trace fragment, or trace of the underlying TA B_A, respectively.

Definition 4.6. A *step of automaton A starting with state s*, or simply a *step starting with s* when A is obvious from the context, is an execution fragment of A starting with s that consists of either one discrete transition surrounded by two point trajectories, or one closed trajectory with no discrete transition.

4.2 Simulation relation and refinement proof techniques for timed I/O automata

In this section, we introduce simulation relation and refinement proof techniques for timed I/O automata. In Section 4.2.1, we introduce a forward simulation relation from automaton A to automaton B, and a refinement from automaton A to automaton B. These techniques can be used to show trace inclusion between two automata A and B ($traces_A \subseteq traces_B$). The soundness theorems of these techniques are also given. In Section 4.2.2, we introduce a new refinement that has a slightly different definition from an ordinary refinement presented in Section 4.2.1. This refinement captures an idea of using invariants of automata in a proof of a refinement. A simulation relation or a refinement actually give us a stronger claim than just trace inclusion. In Section 4.2.3, we introduce the notion of *samples* for an automaton execution. Using this notion, we prove the close correspondence between the executions of two automata that are related by a simulation relation. In Section 4.2.4, as an application of this close correspondence, we discuss invariants that can be deduced from the existence of a simulation relation.

4.2.1 Forward simulation and refinement for timed I/O automata

A *forward simulation* and a *refinement* are proof techniques that can be used to show trace inclusion between two automata A and B ($traces_A \subseteq traces_B$) [6, 9].[1] We often consider B as a specification of a system, and consider A as an implementation of that specification. Informally, the trace inclusion $traces_A \subseteq traces_B$ tells us that the external behavior of the implementation A does not go beyond what we expect from the specification B. All definitions and proofs for forward simulations and ordinary refinements in this subsection are from [6]. *Weak refinements* are introduced in [9], but are not discussed in [6]. Thus, we define weak refinements in terms of the TIOA framework of [6] (since, as we mentioned, the paper [9] uses slightly different framework from the one of [6]), and prove the soundness theorem for them.

[1] Many kinds of simulation relation and refinement proof techniques and their soundness are studied in [9]. However, the authors use slightly different definitions of timed I/O automata from [6]. In this thesis, we follow the definitions used in [6].

Forward simulation (materials from [6])

To prove that there is a forward simulation from automaton A to automaton B, we define a relation between Q_A to Q_B, and show that this relation satisfies the following conditions for a forward simulation.

Definition 4.7. *(Forward Simulation)* Let A and B be comparable timed I/O automata. A *forward simulation* from A to B is a relation $R \subseteq Q_A \times Q_B$ satisfying the following conditions, for all states x_A and x_B of A and B, respectively:

1. If $x_A \in \Theta_A$, then there exists a state $x_B \in \Theta_B$ such that $x_A R x_B$.

2. If $x_A R x_B$ and α is an execution fragment of A consisting of one action surrounded by two point trajectories, with $\alpha.fstate = x_A$, then B has a closed execution fragment β with $\beta.fstate = x_B$, $trace(\beta) = trace(\alpha)$, and $\alpha.lstate\ R\ \beta.lstate$.

3. If $x_A R x_B$ and α is an execution fragment of A consisting of a single closed trajectory, with $\alpha.fstate = x_A$, then B has a closed execution fragment β with $\beta.fstate = x_B$, $trace(\beta) = trace(\alpha)$, and $\alpha.lstate\ R\ \beta.lstate$.

Condition 1 states that for each start state of A there exists a related start state of B. Conditions 2 and 3 assert that each discrete transition and trajectory of A, respectively, can be simulated by a corresponding execution fragment of B with the same trace.

We need a soundness theorem for the forward simulation technique in order to verify that the technique is sound, that is, the existence of a refinement from A to B indeed implies the trace inclusion $traces_A \subseteq traces_B$.

In order to prove the soundness theorem, we need the following lemma that states that a forward simulation yields a correspondence for open trajectories (this is stated as Lemma 4.21 of [6]).

Lemma 4.8. *Let A and B be comparable timed I/O automata and R be a forward simulation from A to B. Let x_A and x_B be states of A and B, respectively, such that $x_A R x_B$. Let α be an execution fragment of A from state x_A consisting of a single open trajectory. Then B has an execution fragment β with $\beta.fstate = x_B$ and $trace(\beta) = trace(\alpha)$.*

The soundness of a forward simulation immediately follows as a corollary from the following theorem and Condition 1 of the definition of a forward simulation.

Theorem 4.9. *Let A and B be comparable timed I/O automata, and let R be a forward simulation from A to B. Let x_A and x_B be be states of A and B, respectively, such that $x_A R x_B$. Then $tracefrags_A(x_A) \subseteq tracefrags_B(x_B)$.*

A proof appears in [6]. Nevertheless, since, in order to prove Theorem 4.21, we need the same construction of an execution fragment of B presented in the proof of Theorem 4.9 (Theorem 4.22 of [6]), we will repeat the proof here.

Proof. Suppose that δ is the trace of an execution fragment of A that starts from x_A. We prove that δ is also a trace of an execution fragment of B that starts from x_B. Let $\alpha = \tau_0 \alpha_1 \tau_1 \cdots$ be an execution fragment of A such that $\alpha.fstate = x_A$ and $\delta = trace(\alpha)$.

We consider the following cases:

1. α is an infinite sequence.

   Using Axioms T_1 and T_2, we can write α as an infinite concatenation of execution fragments $\alpha_0 \frown \alpha_1 \frown \cdots$, in which α_i with i even consist of a trajectory only, and α_i with i odd consist of a single discrete step surrounded by two point trajectories.

   We define inductively a sequence $\beta_0 \beta_1 \cdots$ of closed execution fragments of B, such that $\beta_0.fstate = x_B$ and, for all i, $\beta_i.lstate = \beta_{i+1}.fstate$, $\alpha_i.lstate \; R \; \beta_i.lstate$, and $trace(\beta_i) = trace(\alpha_i)$. We use Condition 2 of the definition of a forward simulation to construct b_i's with i even, and use Condition 3 to construct b_i's with i odd. Let $\beta = \beta_0 \frown \beta_1 \frown \cdots$. By Lemma 4.7 of [6], β is an execution fragment of B. Clearly, $\beta.fstate = x_B$. By Lemma 3.9 of [6], $trace(\beta) = trace(\alpha)$. Thus β has the required properties.

2. α is a finite sequence ending with a closed trajectory.

   Similar to the first case.

3. α is a finite sequence ending with an open trajectory.

   Similar to the first case, using Lemma 4.8.

□

Corollary 4.10. *Let A and B be comparable timed I/O automata and let R be a simulation relation from A to B. Then $traces(A) \subseteq traces(B)$.*

Refinement (materials from [6])

A *refinement* is a simple, special case of a forward simulation, often used in practice, in which the relation between Q_A and Q_B is a partial function.

Definition 4.11. *(Refinement)* Let A and B be comparable timed I/O automata. Let r be a partial function from Q_A to Q_B.

We say that r is a *refinement* from A to B if it satisfies the following three conditions, for all states x_A and x_B of A and B, respectively.

1. If $x_A \in \Theta_A$, then $x_A \in dom(r)$ and $r(x_A) \in \Theta_B$.

2. If α is an execution fragment of A consisting of one action surrounded by two point trajectories and $\alpha.fstate \in dom(r)$, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

3. If α is an execution fragment of A consisting of a single closed trajectory and $\alpha.fstate \in dom(r)$, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

The following theorem (Theorem 4.27 of [6]) gives us the soundness of a refinement.

Theorem 4.12. *Let A and B be two timed I/O automata and suppose $R \subseteq Q_A \times Q_B$. Then R is a refinement from A to B iff R is a forward simulation from A to B and R is a partial function.*

Weak refinement

In some cases, we want to use invariants of automata in a proof of a refinement. A *weak refinement*[2] can be used for such cases. Weak refinements are introduced in [9], but are not discussed in [6]. Thus, we define weak refinements here in terms of the TIOA framework of [6] (since, as we mentioned, the paper [9] uses slightly different framework from the one of [6]), and prove the soundness theorem for them.

Definition 4.13. *(Weak Refinement)* Let A and B be comparable timed I/O automata. Let P_A be an invariant of A, and P_B be an invariant of B. Let r be a partial function from Q_A to Q_B.

We say that r is a *weak refinement with respect to P_A and P_B* if it satisfies the following two conditions for all states x_A and x_B of A and B, respectively.

1. If $x_A \in \Theta_A$, then $x_A \in dom(r)$ and $r(x_A) \in \Theta_B$.

2. If α is an execution fragment of A consisting of one action surrounded by two point trajectories, and $\alpha.fstate \in dom(r)$, and

$$P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate))$$

holds, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

[2] This usage of the term "weak" here comes from [9], and has nothing to do with Milner's usage [11]; he uses it to indicate whether or not internal steps are abstracted away. In contrast, we use the term "weak" here since we have more assumptions (namely, invariants of automata) in Conditions 2 and 3 in the definition of this refinement, than an ordinary refinement.

3. If α is an execution fragment of A consisting of a single closed trajectory, and $\alpha.fstate \in dom(r)$, and
$$P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate))$$
holds, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

The following theorem states a reduction from the existence of a weak refinement (Definition 4.13) to the existence of an ordinary refinement (Definition 4.11).

Theorem 4.14. *Let A and B be comparable timed I/O automata, and let r be a weak refinement from A to B with respect to P_A and P_B. Then the following mapping r^* is a refinement from A to B.*
$$r^* = r\lceil(reachable(A) \cap \{x | r(x) \in reachable(B)\})$$

Proof. Condition 1 of a refinement follows from Condition 1 of the weak refinement r and the fact that start states of A and B are reachable states of A and B, respectively.

Condition 2 of a refinement follows from the following argument: Suppose α is an execution fragment of A consisting of one discrete transition surrounded by two point trajectories, and $\alpha.fstate \in dom(r^*)$.

From this, we assert the following facts:

1. From $\alpha.fstate \in dom(r^*)$ and the definition of r^*, we have $\alpha.fstate \in dom(r)$, $\alpha.fstate \in reachable(A)$, and $r(\alpha.fstate) \in reachable(B)$.

2. Since $\alpha.fstate$ is a reachable state of A from the above Fact 1, and α is a valid execution of A, $\alpha.lstate$ is also a reachable state of A.

3. Since P_A and P_B are invariants of A and B, respectively, $P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate))$ hold.

4. From the above Fact 3, the assumptions of Condition 2 of the definition of the weak refinement are satisfied, with respect to α and r. From this, $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β_1 with $\beta_1.fstate = r(\alpha.fstate)$, $trace(\beta_1) = trace(\alpha)$, and $\beta_1.lstate = r(\alpha.lstate)$.

5. For β_1 in the above Fact 4, since $\beta_1.fstate = r(\alpha.fstate)$ is a reachable state of B (by Fact 1), and β_1 is a valid execution fragment of B, $\beta_1.lstate = r(\alpha.lstate)$ is a reachable state of B.

Using the above facts, we show that $\alpha.lstate \in dom(r^*)$ and B has a closed execution fragment β with $\beta.fstate = r^*(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r^*(\alpha.lstate)$.

We first prove $\alpha.lstate \in dom(r^*)$. It is sufficient to prove $\alpha.lstate \in dom(r)$, $\alpha.lstate \in reachable(A)$, and $r(\alpha.lstate) \in reachable(B)$. These conditions follow from Fact 1, Fact 2, and Fact 5, respectively.

Now we prove that B has a closed execution fragment β that satisfies the required conditions. For β_1 in Fact 4, from the definition of r^*, $\beta_1.fstate = r(\alpha.lstate) = r^*(\alpha.lstate)$ and $\beta_1.lstate = r(\alpha.lstate) = r^*(\alpha.lstate)$ (note that $r^*(\alpha.fstate)$ and $r^*(\alpha.lstate)$ are well defined since $\alpha.fstate \in dom(r^*)$ from our assumption, and $\alpha.lstate \in dom(r^*)$ as proved). Thus, this β_1 satisfies the required conditions.

We can prove Condition 3 of a refinement similarly to the case of Condition 2. □

4.2.2 Weak refinement using step invariants

An ordinary refinement or a weak refinement from A to B works fine in many cases to show trace inclusion between two automata A and B. To prove a weak refinement from A to B, we can assume invariants of A and B hold when proving Conditions 2 and 3 (the step conditions) of the refinement. This is useful since we often need some invariants of two automata to prove the step conditions of the refinement. However, there are some cases when we need to make use of invariants in a slightly different way. As we will see in Chapter 5, in some cases, we actually need *invariants of* B in order to prove some *invariants of* A needed in the proof of a refinement from A to B. Since we can assert the fact that invariants of B also hold for A *only after* proving a refinement from A to B, without some modification to the refinement definition, this reasoning is circular.

Informally, our solution to this problem is to prove the inductive case of the invariant proof for such invariants of A, assuming additional conditions – invariants of B. In the following, we present a new definition of invariants that captures the above informal discussion

Definition 4.15. Let A be a timed I/O automaton. Let P_1 and P_2 be predicates over Q_A. We say that P_1 is a *step invariant of A using* P_2, or simply *a step invariant using* P_2 when A is obvious from the context, if, for any reachable state s of A and any step α of A starting with s, the following condition holds.

$$P_1(\alpha.fstate) \wedge P_2(\alpha.fstate) \Rightarrow P_1(\alpha.lstate)$$

The following lemma easily follows from the definition of a step invariant.

Lemma 4.16. *$P_1 \wedge P_2 \wedge ... \wedge P_n$ is a step invariant for automaton A using condition Q if P_1 is a step invariant of A using Q, and P_i, $2 \leq i \leq n$, is a step invariant of A using Q and $P_1 \wedge ... \wedge P_{i-1}$.*

Now we are ready to define the new refinement. The main difference from the definition of an ordinary weak refinement (Definition 4.13) is that we assume an additional predicate P^* over Q_A in the step conditions (Conditions 2 and 3) of the refinement. This P^* must be a step invariant using $\lambda s.P_B(r(s))$, where P_B is an invariant of B. This captures the above informal discussion: since we need invariant P_B of B in order to prove that P^* is an invariant of A, we just require P^* to be a step invariant using $\lambda s.P_B(r(s))$, invariant P_B "adapted" to A using mapping r.

Definition 4.17. Let A be a timed I/O automaton. Let P_A be an invariant of A, and P_B be an invariant of B. Let r be a partial function from Q_A to Q_B. Let P^* be a step invariant of A using $\lambda s.P_B(r(s))$.

We say that r is a *weak refinement using P_A, P_B, and P^** if it satisfies the following three conditions for all states x_A and x_B of A and B, respectively.

1. If $x_A \in \Theta_A$ then $x_A \in dom(r)$, $r(x_A) \in \Theta_B$, and $P^*(x_A)$ hold.

2. If α is an execution fragment of A consisting of one action surrounded by two point trajectories, and $\alpha.fstate \in dom(r)$, and

$$P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate)) \wedge P^*(\alpha.fstate)$$

holds, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

3. If α is an execution fragment of A consisting of a single closed trajectory, and $\alpha.fstate \in dom(r)$, and

$$P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate)) \wedge P^*(\alpha.fstate)$$

holds, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

We now prove the soundness of the above refinement. Analogously to the case of an ordinary weak refinement (Theorem 4.14), we prove the soundness of this new refinement by proving that the existence of a new refinement implies the existence of an ordinary weak refinement.

Theorem 4.18. *Let A and B be comparable timed I/O automata, and let r be a weak refinement from A to B using P_A, P_B, and P^*. Then the following mapping r^* is a weak refinement from A to B, with respect to $\lambda s.(s \in reachable(A))$ and $\lambda t.(t \in reachable(B))$.*

$$r^* = r\lceil\{x|P^*(x)\}$$

Proof. Condition 1 of a weak refinement for r^* follows from Condition 1 of the new refinement r and the definition of r^*.

Condition 2 of a weak refinement for r^* follows from the following argument: Suppose α is an execution fragment of A consisting of one discrete transition surrounded by two point trajectories, and $\alpha.fstate \in dom(r^*)$, $\alpha.fstate \in reachable(A)$, and $\beta.fstate \in reachable(B)$ hold.

From this, we assert the following facts:

1. $P_A(\alpha.fstate)$ holds since P_A is an invariant of A and $\alpha.fstate$ is a reachable state of A. $P_B(r(\alpha.fstate))$ follows from an analogous reason.
2. From $\alpha.fstate \in dom(r^*)$ and the definition of $dom(r^*)$, we have $\alpha.fstate \in dom(r)$ and $P^*(\alpha.fstate)$.
3. From Facts 1 and 2, the assumptions of Condition 2 of the new refinement r is satisfied with respect to α. Thus, $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β_1 with $\beta_1.fstate = r(\alpha.fstate)$, $trace(\beta_1) = trace(\alpha)$, and $\beta_1.lstate = r(\alpha.lstate)$.
4. $P^*(\alpha.lstate)$ holds since P^* is a step invariant using $\lambda s.P_B(r(s))$, $\alpha.fstate$ is a reachable state of A, $P^*(\alpha.fstate)$ holds from the Fact 2, and $P_B(r(\alpha.fstate))$ holds from Fact 1.

Using the above facts, we now prove that $\alpha.lstate \in dom(r^*)$ and B has a closed execution fragment β with $\beta.fstate = r^*(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r^*(\alpha.lstate)$.

First, we prove $\alpha.lstate \in dom(r^*)$. It is sufficient to prove $\alpha.lstate \in dom(r)$ and $P^*(\alpha.lstate)$. The first condition follows from Fact 3, and the second condition follows from Fact 4.

Now we prove that B has a closed execution fragment β that satisfies the required conditions. For β_1 in Fact 3, from the definition of r^*, $\beta_1.fstate = r(\alpha.lstate) = r^*(\alpha.lstate)$ and $\beta_1.lstate = r(\alpha.lstate) = r^*(\alpha.lstate)$ (note that $r^*(\alpha.fstate)$ and $r^*(\alpha.lstate)$ are well defined since $\alpha.fstate \in dom(r^*)$ from our assumption, and $\alpha.lstate \in dom(r^*)$ as proved). Thus, this β_1 satisfies the required conditions.

We can prove Condition 3 of the definition of a weak refinement similarly to the case of Condition 2.

□

4.2.3 Close correspondence between executions of two automata implied by a forward simulation

The existence of a simulation relation from A to B actually implies more than just trace inclusion – it implies a close correspondence, involving both traces and states, between each execution fragment of A and some execution fragment of B.

We first formally define this close correspondence between two execution fragments. To do this, we need a notion of a *sample* of an execution fragment.

Definition 4.19. Let A be a timed I/O automaton, and let α be an execution fragment of A. A *sample* of α is a (possibly infinite) sequence of closed execution fragments $\alpha_0\alpha_1\cdots$ of A such that $\alpha_i.lstate = \alpha_{i+1}.fstate$ for any $i \geq 0$ and $\alpha = \alpha_0 \frown \alpha_1 \frown \cdots$.

Informally, we can consider each $\alpha_i.fstate$ as a sampling point of α for the given sample $\alpha_0\alpha_1\cdots$.

Now we define a close correspondence between two execution fragments.

Definition 4.20. Let A and B be comparable timed I/O automata. Let α and β be execution fragments of A and B, respectively. Let R be a relation over Q_A and Q_B. Let $\Sigma = \alpha_0\alpha_1\cdots$ be a sample of α. We say that α and β *correspond* with respect to R and Σ, provided that there exists a sample $\beta_0\beta_1\cdots$ of β such that for any $i \geq 0$, $\alpha_i.fstate\ R\ \beta_i.fstate$, $\alpha_i.lstate\ R\ \beta_i.lstate$, and $trace(\beta_i) = trace(\alpha_i)$.

Since $trace(\beta_i) = trace(\alpha_i)$ for any $i \geq 0$, $ltime(\alpha_0 \frown \alpha_1 \frown ... \frown \alpha_k) = ltime(\beta_0 \frown \beta_1 \frown ... \frown \beta_k)$, for any $k \geq 0$.

Theorem 4.21. *Let A and B be comparable timed I/O automata, and let R be a forward simulation from A to B. Let x_A and x_B be states of A and B, respectively, such that $x_A R x_B$. For any execution fragment α of A starting with x_A and any sample Σ of α, there is an execution β of B starting with x_B such that α and β correspond with respect to R and Σ.*

Proof. By using the same construction scheme as in the proof of Lemma 4.9 as a "subroutine", for each closed execution fragment α_i in sample Σ, we can inductively construct a corresponding execution fragment β_i of B such that $\beta_i.lstate = \beta_{i+1}.fstate$, $\alpha_i.lstate\ R\ \beta_i.lstate$, and $trace(\alpha_i) = trace(\beta_i)$ for any $i \geq 0$. Note that α_i's here and α_i's in the proof of Lemma 4.9 is defined differently: in this proof, each α_i corresponds to α in Lemma 4.9. We split each α_i into a concatenation of smaller execution fragments $\alpha_i^0 \frown \alpha_i^1 \frown \cdots \alpha_i^{k_i}$ to construct a corresponding β_i as in the proof of Lemma 4.9 (since each α_i is closed, it is split into a finite concatenation). Since each β_i^j that corresponds to α_i^j is closed (by a construction using simulation relation R as in the proof of Lemma 4.9), and β_i is constructed by a finite concatenation $\beta_i^0 \frown \beta_i^1 \frown \cdots \beta_i^{k_i}$, each β_i is closed. Now let $\beta = \beta_0 \frown \beta_1 \cdots$. From Theorem 4.7 of [6], β is an execution fragment of B. It is easy to see that α and β correspond with respect to R and Σ, by sample $\beta_0\beta_1 \cdots$ of β. □

4.2.4 Invariants deduced from a forward simulation

As we discussed in Section 4.2.3, a forward simulation from A to B actually implies more than just trace inclusion ($\mathit{traces}_A \subseteq \mathit{traces}_B$) proved by the soundness theorem (Corollary 4.10). Using Theorem 4.21 proved in Section 4.2.3, we can guarantee that for any invariant of B, A has a corresponding similar invariant. We formally state this claim as follows.

Theorem 4.22. *Let A and B be timed I/O automata. Let R be a simulation relation from A to B. Let P_B be an invariant of B. For any $s \in \mathit{reachable}(A)$, there exists $t \in Q_B$ such that sRt and $P_B(t)$ (equivalently, the predicate $\lambda s.(\exists t \in Q_B : sRt \wedge P_B(t))$ is an invariant of A.)*

Proof. From Theorem 4.21, there is an execution β of B such that α and β correspond with respect to R and sample α (a sequence with a single element α). Thus, for this β, $\alpha.\mathit{lstate}\ R\ \beta.\mathit{lstate}$. Since β is a valid execution of B, $\beta.\mathit{lstate}$ is a reachable state of B. Thus $P_B(\beta.\mathit{lstate})$ holds. □

From Theorems 4.12, 4.14, and 4.18, we have the following corollary of Theorem 4.22.

Corollary 4.23. *Let A and B be timed I/O automata. Let r be a refinement, a weak refinement, or a weak refinement using step invariants, from A to B. Let P_B be an invariant of B. The predicate $\lambda s.P_B(r(s))$ is an invariant of A.*

Proof. First we consider the case when r is an ordinary refinement. From theorem 4.12, r is a simulation relation from A to B. Thus from Theorem 4.22, for any $s \in \mathit{reachable}(A)$, there exists $t \in Q_B$ such that sRt and $P_B(t)$. Since r is a partial function, there exists exactly one t such that sRt, namely $r(t)$. Thus for any $s \in \mathit{reachable}(A)$, $P_B(r(s))$ holds, as required.

Now we consider the case when r is a weak refinement, or a weak refinement using step invariants from A to B. From Theorems 4.14 and 4.18, there exists some ordinary refinement r^* from A to B such that $\mathit{dom}(r^*) \subseteq \mathit{dom}(r)$ and $\forall s \in \mathit{dom}(r^*), r^*(s) = r(s)$. For this r^*, by using th same argument as above, we assert that for any $s \in \mathit{reachable}(A)$, $P_B(r^*(s))$ holds. Since $r^*(s) = r(s)$ for any $s \in \mathit{dom}(r^*)$, and $\mathit{reachable}(A) \subseteq \mathit{dom}(r^*)$, for any $s \in \mathit{reachable}(A)$, $P_B(r(s))$ holds. □

Chapter 5

Continuous model of SATS and its safe separation property

5.1 Introduction

In this chapter, we introduce a new model of the SATS landing protocol that more realistically reflects the aircraft dynamics and the airport geometry. We use the timed I/O automata framework presented in Chapter 4 to construct the new model.

This chapter is organized as follows. In Section 5.2, we study an extension of the discrete model of [2] presented in [12], what the authors call a *hybrid model.* In the hybrid model of [12], the movement of the aircraft in the approach area and the missed approach zones is modeled as a continuous behavior. In Section 5.3, we present our new continuous model $ContSATS$, and compare it with both the discrete model presented in Chapter 2 and the hybrid model of [12]. Section 5.5 is devoted to carrying over the previous results to the new model. Using the refinement technique, we prove that the safe separation properties analogous to those for the discrete model hold for the new model. In Section 5.6, we verify several spacing properties of aircraft in $ContSATS$.

5.2 Hybrid model of [12]

In Chapter 2, we have constructed a discrete model of the SATS landing protocol based on [2]. We have used the I/O automata framework to construct the model. Using this discrete model, we have stated and proved the safe separation property of the protocol in terms of the bounds on the number of aircraft in specific discretized zones.

However, to state and prove safety properties of the protocol that involve more realistic dynamics of aircraft, such as a lower bound on the spacing between aircraft, we need a more detailed modeling of the aircraft dynamics and the geometry of the airport. To treat such properties, an extension of the discrete model of [2], what the authors call a *hybrid model*, is presented in [12]. In the hybrid model of [12], the movement of the aircraft in the approach area and the missed approach zones is modeled as continuous behavior: This particular sub-area (the approach area and the missed approach zones) of the airport is modeled as a collection of lines representing pre-determined paths of aircraft on which aircraft continuously move according to their velocity vector (the amount of speed, plus the direction on the line on which the aircraft is moving). See Fig. 5.1 for a picture of this sub-area. Now the discrete transitions for aircraft in the approach area and the maz zones (Merging, FinalSegment, Landing, MissedApproach, and HoldingPatternDescend) are performed when an aircraft reaches the intersection points of two consecutive lines, in order to reassign the line on which that aircraft moves. On the other hand, the behavior of aircraft in the area outside of this area and these zones is still discretized: aircraft in this area (holding3, holding2, and lez) move in logically divided zones by discrete transitions in the exact same way as in the discrete model of [2]. Even though a formal specification of this hybrid model does not appear in [12],[1] we can formalize this model as follows: In the hybrid model of [12], aircraft have a new attribute pos. This pos attribute of an aircraft represents the position in the line at which the aircraft is located. Using pos, the preconditions of the transitions performed for aircraft in the approach area and the maz zones in the discrete model are modified as follows. Such a transition is enabled when the original precondition for that transition is satisfied, and in addition, the aircraft a that moves by the transition is at the end point of the line in which a moves ($a.\mathsf{pos} = \mathsf{L_z}$, where $\mathsf{L_z}$ is the length of the line z in which a is).

In [12], the authors assume that the velocity v of each aircraft in the approach area and the missed approach zones is bounded by $\mathsf{V_{min}} \leq v \leq \mathsf{V_{max}}$ (where $0 < \mathsf{V_{min}} \leq \mathsf{V_{max}}$) and the initial spacing of $\mathsf{S_0}$ is guaranteed when an aircraft initiates the approach. Using these assumptions, lower bounds on the spacings between two aircraft in the approach area and the missed approach zones, respectively, of the hybrid model are claimed and exhaustively checked in [12] using a symbolic model-checking technique. To formally state the bounds the authors obtained in [12], we use the following constants. Let $\mathsf{L_B}$, $\mathsf{L_I}$, $\mathsf{L_F}$, and $\mathsf{L_M}$ represent the lengths of lines representing base, intermediate, final, and maz, respectively. In [12], they consider the case that the lengths of the base zones are different on the right and left sides. However, since this slight generalization does not drastically change our results, we assume in this thesis that the airport geometry is symmetric.[2] We indicate by $\mathsf{L_T}$ the total length that aircraft fly in the approach area, that is,

[1] The authors model-checked some properties of the model. Thus there is a formal specification of the model. However, such a specification does not appear in [12]: instead, the authors informally described how the hybrid model differs from the previous discrete model of [2].

[2] Indeed, the results would change in a very subtle manner in an asymmetric case for the results of [12]: If we

$\mathsf{L_B} + \mathsf{L_I} + \mathsf{L_F}$. The function D is used to represent the distance that a specific aircraft has flown in the approach area, and then in the missed approach zone:

$$\mathsf{D}(a) = \begin{cases} a.\mathsf{pos} & \text{if } a \text{ is in } \mathsf{base} \\ \mathsf{L_B} + a.\mathsf{pos} & \text{if } a \text{ is in } \mathsf{intermediate} \\ \mathsf{L_B} + \mathsf{L_I} + a.\mathsf{pos} & \text{if } a \text{ is in } \mathsf{final} \\ \mathsf{L_B} + \mathsf{L_I} + \mathsf{L_F} + a.\mathsf{pos} & \text{if } a \text{ is in } \mathsf{maz} \\ 0 & \text{otherwise} \end{cases}$$

In [12], the spacing between two aircraft a and b in the approach area and the maz zones is determined by the difference between the values of this D function for a and b, that is, $|\mathsf{D}(a) - \mathsf{D}(b)|$. Note that this spacing does not represent the Euclidean distance between a and b. Indeed, if a is in the right-most portion of base(right) and b in the left-most portion of base(left), then the spacing between a and b using D is *zero*, even if they are on the opposite side of each other. In addition, for instance, if a is in intermediate and b is in base, then the spacing between a and b is not the Euclidean distance between a and b, but the distance that b must fly in the determined path in the approach area from the current position of b to reach the current position of a ($(L_b - b.\mathsf{pos}) + a.\mathsf{pos}$, see two aircraft in base(right) and in intermediate, respectively, in Fig. 5.1).

We are now ready to present the spacing bounds model-checked in [12]. Let $\Delta = \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{min}}}$. For any reachable states of the hybrid model, the following two conditions hold:

1. For any two aircraft a and b that are in the approach area, $\mathsf{D}(a) - \mathsf{D}(b) \leq \mathsf{S_T}$, where $\mathsf{S_T} = \mathsf{S_0} - (\mathsf{L_T} - \mathsf{S_0})\Delta$.

2. For any side σ and any two aircraft a and b that are in $\mathsf{maz}(\sigma)$, $\mathsf{D}(a) - \mathsf{D}(b) \leq \mathsf{S_M}$, where $\mathsf{S_M} = \min(\mathsf{L_T} - \mathsf{L_M}\Delta,\ 2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta)$.

A limitation of this hybrid model is that it captures only the dynamic behavior of aircraft in the approach area and the maz zones – the area outside of these area and zones is still discretized.

In this chapter, we present a new *continuous model* that more realistically captures the aircraft dynamics and the airport geometry than the hybrid model of [12] studied in this section. In contrast to the hybrid model of [12], our continuous model captures the continuous movement of aircraft in the entire Self Controlled Area. As we will see in Section 5.3, a double transition that the discrete model of [2] (and thus also the model in Chapter 2) exhibits by

use an asymmetric geometry, the only change in the approach area is the length $\mathsf{L_B}$ of the base zones: one of the base zone is longer than the other. Let $\mathsf{L_{max}}$ be the length of the longer base zone, and $\mathsf{L_{min}}$ be the length of the shorter one. We can obtain the lower bounds for the asymmetric case checked in [12] by replacing every $\mathsf{L_B}$ that appears in the lower bounds by either $\mathsf{L_{min}}$ or $\mathsf{L_{max}}$ in a way that the replacement increases the spacing.

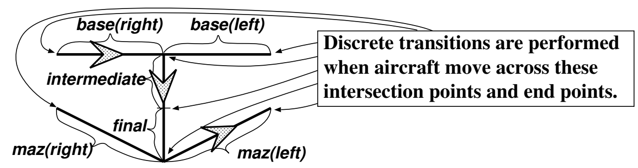

base(right)
base(left)
Discrete transitions are performed when aircraft move across these intersection points and end points.
intermediate
final
maz(right)
maz(left)

Figure 5.1: Sub-area that exhibits a continuous behavior in the hybrid model of [12]

LowestAvailableAltitude is no longer performed in this model: only one aircraft move from one zone to another by each transition.

Using this model, we first formally verify that the safe separation properties proved for the discrete model in Chapter 3 also hold in our new model. We use a *weak refinement using a step invariant*, the new refinement technique presented in Chapter 4, to carry over the results for the discrete model to the new model. Next we verify several lower bounds on the spacing of aircraft including those model-checked in [12]. For the spacing of aircraft in the maz zones, we actually obtain a stronger bound than the one model-checked in [12] using an additional reasonable assumption.

5.3 Our New Continuous model

In the hybrid model of [12], the dynamics of the aircraft in the approach area and the missed approach zones are described as continuous behavior: these area and zones are presented as a collection of lines in which aircraft move continuously according to their velocity. Using this model, the authors of [12] model-checked some spacing properties of aircraft for the model. The model of [12] is sufficient to reason about bounds on the spacing of aircraft in the approach area and the missed approach zone. However, if we want to examine similar properties for the rest of the area, rather than just the approach area and the missed approach zones, we need a model that describes a more complete dynamics of aircraft in the entire Self Controlled Area.

In this section, we present a new *continuous model* of the landing protocol, $ContSATS$, that more realistically reflects the dynamics of the aircraft movement in a real system than the model of [12]. To describe a complete continuous dynamics in the entire Self Controlled Area, we use the same strategy as used for the model of [12]: in $ContSATS$, we model all the possible paths of aircraft as a collection of lines, with aircraft moving on them according to their velocity; the discrete transitions are performed on the intersection points of consecutive lines (see Fig. 5.2, and compare it with Fig. 2.3 and 2.4). In the new model, analogously to the hybrid model of [12], we use the discrete transitions to re-assign the line on which an aircraft moves when that aircraft reaches the intersection point of two lines. Note that the discrete transitions are performed *instantaneously*: no time elapses during the transitions. This assumption is reasonable considering that they just re-assign the logical zone (the line) to which the aircraft currently belongs.

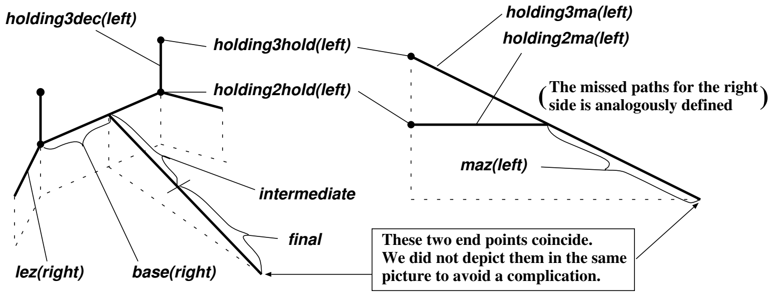

holding3dec(left)
holding3hold(left)
holding2hold(left)
holding3ma(left)
holding2ma(left)
(The missed paths for the right side is analogously defined)
maz(left)
intermediate
final
These two end points coincide. We did not depict them in the same picture to avoid a complication.
lez(right)
base(right)

Figure 5.2: The continuous model $ContSATS$

There are two major problems that make this extension nontrivial. The first problem is that the system has holding fixes (holding3 zones and holding2 zones), where aircraft hover until the condition for the next procedure (modeled as a discrete transition) is satisfied. In the real system, when an aircraft reaches some specific holding point, it starts hovering by circling around the point. We abstractly model hovering aircraft as follows: when an aircraft reaches a specific holding point (holding3hold or holding2hold of either side in Fig. 5.2), it temporarily stops moving along on the lines, and stays at the point until the condition for the next procedure is satisfied. To obtain some reasonable upper bound on the duration of this hovering after the next procedure (transition) gets enabled, we set a "time bound" for this next procedure to be performed. We will discuss more details of this time bound in Section 5.3.1.

The second problem arises from the design of a specific transition of the discrete model. The procedure for aircraft to go back to the holding fixes when missing the approach is represented by LowestAvailableAltitude. As the name implies, the aircraft moves to the lowest available altitude by the above stated transition: if holding2 and holding3 of that side are both empty, then it moves to holding2; and if holding3 is empty, but holding2 is not, then it moves to holding3. A problem occurs when holding3 is not empty at the moment the transition is performed: the missed aircraft then moves to holding3, and *at the same time*, an aircraft in holding3 descends to holding2. Therefore, LowestAvailableAltitude exhibits a *double transition* of aircraft in this situation. Considering that we cannot fully synchronize two aircraft in a real system, this movement is not a faithful modeling of the reality. The problem was not resolved in the hybrid model of [12] either, because the holding fixes are still discretized, and the transition does still exhibit a double transition. In [2], the authors partially justified using a double transition by claiming that a double transition never occurs in a real protocol since the SATS concept precludes an aircraft from hovering at one altitude when a lower altitude is available. This indicates that when a missed aircraft moves to holding3, the aircraft previously in holding3 has already descended to holding2. However, they could not verify such a property for the discrete model of [12] or the hybrid model of [2], since the model does not have any time-dependent constraints, such as a time bound for `HoldingPatternDescend` to be performed when a lower altitude becomes available (that is, `holding2` becomes empty).

In our new model, we modify the effects of LowestAvailableAltitude(σ) so that the transition will never exhibit a double transition: if holding2(σ) and holding3(σ) are both empty, then the aircraft moves (its line is re-assigned) to holding2(σ); otherwise it moves to holding3(σ) *without checking* if holding3(σ) is empty or not. Therefore only one aircraft moves at once in the logical zones by the transition. In addition, we will guarantee (by Lemma 5.8 and Theorem 5.13) that holding3(σ) is actually empty whenever LowestAvailableAltitude(σ) occurs in $ContSATS$, provided that the distances that an aircraft flies in the approach area and in the missed approach zone are sufficiently long. Informally, this fact states that an explicit check of the emptiness of holding3(σ) can be replaced by reasonable time-dependent constraints. We will formally state these constraints in Section 5.3.6. The emptiness of holding3(σ) in the latter situation of the transition is crucial in the proof of the refinement (Theorem 5.13) to match up the effects of the transition in $ContSATS$ and in the discrete model.

5.3.1 Formal Specification for $ContSATS$

Now we present formal code for $ContSATS$. It is written in the timed I/O automata specification language (TIOA, [3]). Analogous to the formal code for the discrete model presented in Chapter 2, the automaton definition imports vocabulary ContSatsVocab presented in Figure 5.3.

As we have discussed, each logical zone has (possibly multiple) corresponding lines representing paths of aircraft on which aircraft move, presented by enumeration Lines in ContSatsVocab (again, see Fig. 5.2). All zones but holding2 and holding3 have exactly one line corresponding to their zone-queue counterpart. Each holding2 zone has one point, holding2hold, and one line, holding2ma, which respectively represent a hovering point of aircraft and the path from the end point of the missed approach path to the hovering point. Each holding3 zone has one point and two lines, where the point and one line are analogous to those of holding2's – a hovering point, holding3hold, and the path, holding3ma; and the other line, holding3dec, is used to represent the descending path from holding3hold to holding2hold.[3]

[3] We do not split this path to two: one belongs to holding2 and the other to holding3. The reason is because, to our best knowledge, a typical minimum vertical separation of two aircraft for the safety purpose is set to 1000 feet. Presumably, the 1000-feet difference in the altitudes of holding fixes comes from this issue, though we could not find the statement that specifically mention this minimum vertical separation in [16]. If we split the line,

```
vocabulary ContSatsVocab
  types
    %% ADDED %%
    Lines enumeration
                 [LINE_holding3holdL, LINE_holding3holdR, LINE_holding3maL, LINE_holding3maR,
                  LINE_holding3decL, LINE_holding3decR, LINE_holding2holdL, LINE_holding2holdR,
                  LINE_holding2maL, LINE_holding2maR, LINE_lezL, LINE_lezR, LINE_mazL, LINE_mazR,
                  LINE_baseL, LINE_baseR, LINE_intermediate, LINE_final, LINE_runway,
                  ]

    %% NEW ATTRIBUTES ADDED %%
    Aircraft tuple [
      mahf: Side, % Missed approach holding fix assignment.
      id  : ID    % ID of the aircraft
      line: Lines           % Which line the aircraft is on.
      pos : AugmentedReal   % The position of the aircraft in the zone.
      t   : AugmentedReal   % The time when the discrete transition for the aircraft gets enabled.
                            % it is set to -1 if the transition is not enabled yet, or
                            % the transition does not have a time bound.
      ]

    %% Everything else is exactly the same as SatsVocab used for the discrete model.
```

Figure 5.3: Vocabulary for *ContSATS*

The line on which a specific aircraft currently moves is specified by a new attribute of aircraft, line. To distinguish the logical-zone queues and the line names as an aircraft attribute, we use the prefix "LINE_" for the line names; for example, the final zone as a line is represented as LINE_final. The position of a specific aircraft in the line is specified by another new attribute of aircraft, pos. Using both the line value and pos value, we can uniquely determine on which line, and at what position in that line the aircraft is now.

Another new attribute of Aircraft is t, which is used to express a time bound on the duration from the time some specific transitions become enabled to the time those transitions are performed. Transitions StartDescending, VerticalApproachInitiation, and Taxiing, have these time bounds. StartDescending is a new transition that makes an aircraft holding at holding3hold start descending. The VerticalApproachInitiation transition is "inherited" from the discrete model of [2], and this transition also makes a hovering aircraft (at holding2hold, this time) start moving. Taxiing is also inherited from the discrete model. This transition has a time bound in *ContSATS* due to the change in the precondition of Landing as we will discuss in Section 5.3.3. Informally, we can consider these time bounds as follows. When one of the above the transitions gets enabled, the aircraft a corresponding to the transition (the one that will move by the transition)

we can easily see that the entry of an aircraft to holding3 when another aircraft just across the border of these split line will violate this minimum vertical separation of 1000 feet. Thus we decided to have the descending path belong just to holding3.

has its t value reset to the value of now. As we will see in Section 5.3.5, by the **stop when** clause in the trajectory definition, we will insist *ContSATS* to fire the transition either before or at the time the value of $\mathsf{now} - a.\mathsf{t}$ gets the following pre-determined time bound for that transition. T_3, T_2, and $\mathsf{T}_{\mathsf{Tax}}$ represents the time bounds for StartDescending, VerticalApproachInitiation, and Taxiing respectively. We use function T that maps the name of a zone to the above specified time bounds for aircraft in that zone:

$$\mathsf{T}(z) = \begin{cases} \mathsf{T}_3 & \text{if } z = \mathsf{holding3}(\sigma) \text{ for some side } \sigma \\ \mathsf{T}_2 & \text{if } z = \mathsf{holding2}(\sigma) \text{ for some side } \sigma \\ \mathsf{T}_{\mathsf{Tax}} & \text{if } z = \mathsf{runway} \\ 0 & \text{otherwise} \end{cases}$$

We set t of aircraft outside of the holding zones or of the runway to -1, indicating that the timers are not set for those aircraft. In formal code, we represent these conditions of the deadline for the transitions by the **stop when** statement in the trajectory definition. Note that after the transition that has a time bound is performed, the t attribute for the respective aircraft is reset to -1.

We now introduce some constants and functions for the lengths of lines that are used in formal code of *ContSATS*. $\mathsf{L}_{\mathsf{3dec}}$, $\mathsf{L}_{\mathsf{3ma}}$, L_{B}, L_{I}, L_{F}, and L_{M} respectively represents the lengths of holding3dec, holding3ma, base, intermediate, final, and maz. We use the function L to obtain the length from the name of the line. For example, $\mathsf{L}(\mathsf{final}) = \mathsf{L}_{\mathsf{F}}$. We indicate by L_{T} the total length that aircraft fly in the approach area, that is, $\mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + \mathsf{L}_{\mathsf{F}}$. We use the function D to represent the distance a specific aircraft has flown in the approach area, and then in the missed approach zone:

$$\mathsf{D}(a) = \begin{cases} a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_base}(\sigma) \text{ for some side } \sigma \\ \mathsf{L}_{\mathsf{B}} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_intermediate} \\ \mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_final} \\ \mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + \mathsf{L}_{\mathsf{F}} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_maz}(\sigma) \text{ for some side } \sigma \\ 0 & \text{otherwise} \end{cases}$$

We present formal code for *ContSATS* in Figures 5.4 - 5.6. We use three effects set_pos, set_line, and set_t to re-assign the pos, line, and t attributes of aircraft, respectively. We retain the queue structure of the logical zones in *ContSATS* since having the same structure as the discrete model is useful when proving a refinement from *ContSATS* to the discrete model. We maintain the consistency between the zone queues and the lines representing paths of aircraft in *ContSATS* as follows: when an aircraft achieves an intersection point of two consecutive lines representing two specific zones, the discrete transition is triggered (since the trajectory is stopped by the stopping condition in the **stop when** clause), and re-assigns the line on which the aircraft moves, and also moves the same aircraft to the corresponding queue in the logical zones.

```
automaton ContSATS
  imports ContSatsVocab

%% All original discrete transitions are considered as the output transitions.
%% We added four new internal transitions, as well as the trajectory definition.
signature
  output
    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),
  internal
    StartHolding2(ac:Aircraft,side:Side),
    StartHolding3(ac:Aircraft,side:Side),
    StartDescending(ac:Aircraft,side:Side),
    SetTime
states
  zones : zone_map, % mapping from a zone name to a zone
  nextmahf : Side, % Next missed approach holding fix
  landing_seq : Zone % landing sequence is defined as a queue
  now : AugumentedReal % the time elapsed from the initial state
  initially
    zones = initialZones ∧ nextmahf = right ∧ landing_seq = empty ∧ now = 0

    %% The auxiliary functions are the same as SATS
```

Figure 5.4: Formal code for *ContSATS*, Part 1 of 3

We will formally prove the consistency between the zone queues and the lines in *ContSATS* as Lemma 5.2.

5.3.2 State of *ContSATS*

The *ContSATS* model has one new analog state variable now (see "states" section of the code in Figure 5.4). It keeps track of the time that has elapsed since the system starts its execution. The evolution of the value of now is described as $d(\mathsf{now}) = 1$ in **evolve** statement in the trajectory statement in Figure 5.6, line 17.

5.3.3 Transitions inherited from the discrete model to *ContSATS*

All of the twelve output transitions are "inherited" from the discrete model. The preconditions and the effects of these transitions were modified in a way that they correctly express the movement of the aircraft on lines representing paths. More specifically, for the transitions that represent the movement of aircraft from one zone to another, the following three modifications

transitions

```
output VerticalEntry($a, id, side$)
 pre let $time$ = IF empty_qn(holding2(side)) THEN now
        ELSE -1 FI in
     let $line$ = IF $side$ = right THEN AC_holding3holdR
                 ELSE AC_holding3holdL in
     virtual($side$) $< 2 \wedge$
     $\neg$on_approach_qn($side$) $\wedge$
     empty_qn(maz($side$)) $\wedge$
     empty_qn(lez($side$)) $\wedge$
     empty_qn(holding3($side$)) $\wedge$
     $a$ = aircraft($side$,$id$,$line$,$time$) $\wedge$
     $\forall$ac: Aircraft
      (in_queue_qn(ac, landing_seq) $\vee \exists z$, on_zone_qn(z, ac)
        $\Rightarrow$ ac.id $\neq id$)
 eff zones :=
       assign(zones, holding3($side$), add(holding3($side$), $a$));
     landing_seq := add(landing_seq, $a$);
     nextmahf := opposite($a$.mahf);

output LateralEntry($a, id, side$)
 pre let $line$ = IF $side$ = right THEN AC_lezR
        ELSE AC_lezL in
     virtual($side$) $= 0 \wedge$
     $a$ = aircraft($side$,$id$,$line$,-1) $\wedge$
     $\forall$ac: Aircraft
      (in_queue_qn(ac, landing_seq $\vee \exists z$, on_zone_qn(z, ac))
        $\Rightarrow$ ac.id $\neq id$)
 eff zones :=
        assign(zones, lez($side$), add(lez($side$), $a$));
     landing_seq := add(landing_seq,$a$);
     nextmahf := opposite($a$.mahf);

internal StartDescending($a, side$)
 pre $\neg$empty_qn(holding3($side$)) $\wedge$
     $a$ = first(holding3($side$)) $\wedge$
     $a$.line = AC_holding3hold($side$)
 eff set_line($a$, AC_holding3dec($side$));
     set_pos($a$, 0);
     set_t($a$, -1);

output HoldingPatternDescend($a, side$)
 pre $\neg$(empty_qn(holding3($side$))) $\wedge$
     $a$ = first(holding3($side$)) $\wedge$
     $a$.line = AC_holding3dec($side$) $\wedge$
     $a$.x = $L_3$
 eff set_line($a$, AC_holding2hold($side$));
     set_pos($a$, 0);
     IF length(base(opposite($side$))) $\leq 1 \wedge$
        (first_in_seq_qn($a$) $\vee$
             (on_approach_qn(leader($a$,landing_seq)) $\wedge$
        D(leader($a$,landing_seq)) $\geq S_0$))
     THEN set_t($a$, now) FI
     zones:=move(holding3($side$),holding2($side$),zones);
```

```
output VerticalApproachInitiation($a, side$)
 pre $\neg$(empty_qn(holding2($side$))) $\wedge$
     $a$ = first(holding2($side$)) $\wedge$
     length(base(opposite($side$))) $\leq 1 \wedge$
     (first_in_seq_qn($a$) $\vee$
        (on_approach_qn(leader($a$,landing_seq)) $\wedge$
        D(leader($a$,landing_seq)) $\geq S_0$))
 eff set_line($a$, AC_base($side$));
     set_pos($a$, 0);
     set_t($a$, -1);
     zones := move(holding2($side$),base($side$),zones)

output LateralApproachInitiation($a, side$)
 pre $\neg$(empty_qn(lez($side$))) $\wedge$
     $a$ = first(lez($side$)) $\wedge$
     $a$.x = $L_L$
 eff set_pos($a$, 0);
     IF length(base(opposite($side$))) $\leq 1 \wedge$
        (first_in_seq_qn($a$) $\vee$
        (on_approach_qn(leader($a$,landing_seq)) $\wedge$
            D(leader($a$,landing_seq)) $\geq S_0$))
     THEN set_line($a$, AC_base($side$))
            zones := move(lez($side$),base($side$),zones);
     ELSE set_line($a$, AC_holding2hold($side$))
            zones := move(lez($side$),holding2($side$),zones);
     FI;

internal SetTime
 pre $\exists$ a:Aircraft
     (a.line = AC_holding2holdL $\vee$
        a.line = AC_holding2holdR) $\wedge$
     a.t = -1 $\wedge \neg$first_in_seq_qn($a$) $\wedge$
     on_approach_qn(leader($a$,landing_seq)) $\wedge$
     D(leader($a$,landing_seq)) = $S_0$
 eff zones:= setTime(zones, landing_seq)

output Merging($a, side$)
 pre $\neg$(empty_qn(base($side$))) $\wedge$
     $a$ = first(base($side$)) $\wedge$
     (first_in_seq_qn($a$) $\vee$
     on_zone_qn(intermediate,leader($a$,landing_seq))$\vee$
     on_zone_qn(final,leader($a$,landing_seq))) $\wedge$
     $a$.x = $L_B$
 eff set_line($a$, AC_intermediate);
     set_pos($a$,0)
     zones := move(base($side$),intermediate,zones);

output Exit($a$)
 pre $\neg$(empty_qn(intermediate)) $\wedge$
     $\neg$(empty_qn(landing_seq)) $\wedge$
     $a$ = first(intermediate) $\wedge$
     first_in_seq_qn($a$)
 eff zones:= assign(zones,intermediate,rest(intermediate));
     landing_seq := rest(landing_seq)
```

Figure 5.5: Formal code for *ContSATS*, Part 2 of 3

```
output FinalSegment(a)
 pre ¬(empty_qn(intermediate)) ∧
     a = first(intermediate)∧
     a.x = $L_I$
 eff set_line(a, AC_final);
     set_pos(a, 0)
     zones := move(intermediate, final, zones);

output Landing(a)
 pre ¬(empty_qn(final)) ∧
     ¬(empty_qn(landing_seq)) ∧
     a = first(final) ∧
     a.x = $L_F$
 eff set_line(a, AC_runway);
     set_pos(a, 0);
     set_t(a, now);
     zones := move(final,runway,zones);
     landing_seq := rest(landing_seq);

output Taxiing(a)
 pre ¬(empty_qn(runway)) ∧
     a = first(runway)
 eff set_t(a, −1);
     zones:= assign(zones, runway, rest(runway));

output MissedApproach(a)
 pre ¬(empty_qn(final)) ∧
     ¬(empty_qn(landing_seq)) ∧
     a = first(final) ∧
     a.x = $L_F$
 eff set_line(a, AC_maz(a.mahf));
     set_pos(a, 0)
     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);

output LowestAvailableAltitude(a, side)
 pre ¬(empty_qn(maz(side))) ∧
     a = first(maz(side)) ∧
     a.x = $L_M$;
 eff IF empty_qn(holding3(side)) ∧ empty_qn(holding2(side))
     THEN set_line(a, AC_holding2ma(side));
          set_pos(a, 0);
          zones := move(maz(side),holding2(side),zones);
     ELSE set_line(a, AC_holding3ma(side));
          set_pos(a, 0)
          zones := move(maz(side),holding3(side),zones);
     FI

internal StartHolding3(a, side)
 pre ¬(empty_qn(holding3(side))) ∧
     a = first(holding3(side)) ∧
     a.line = AC_holding3ma(side) ∧
     a.pos = $L_{3ma}$;
 eff set_line(a, AC_holding3hold(side));
     set_pos(a, 0);
     IF empty_qn(holding2(side)) THEN set_t(a, now); FI

internal StartHolding2(a, side)
 pre ¬(empty_qn(holding2(side))) ∧
     a = first(holding2(side))
     a.line = AC_holding2ma(side)
     a.pos = $L_{2ma}$
 eff set_line(a, AC_holding2hold(side));
     set_pos(a, 0);
     IF length(base(opposite(side))) ≤ 1 ∧
            (first_in_seq_qn(a) ∨
            (on_approach_qn(leader(a,landing_seq)) ∧
            D(leader(a,landing_seq)) ≥ $S_0$))
     THEN set_t(a,now) FI

trajectories
   stop when
      (∃ a:Aircraft,
          (∃ z:Zone, on_zone_qn(z, a)) ∧
          a.x ≥ L(a.line))
   ∨  (∃ a:Aircraft,
          (∃ z:Zone, on_zone_qn(z, a)) ∧
          a.t ≠ −1 ∧ now − a.t ≥ T(a.line))
   ∨  (∃ a:Aircraft,
          (∃ z:Zone, on_zone_qn(z, a)) ∧
          ((a.line = holding2L ∨ a.line = holding2R)∧
            a.t = −1 ∧
             ¬first_in_seq_qn(a) ∧
             on_approach_qn(leader(a,landing_seq)) ∧
             D(leader(a,landing_seq)) = $S_0$))
   evolve
      d(now) = 1
      ∀ a: Aircraft
          IF (a.line=holding3decL ∨ a.line=holding3decR)
          THEN ($V_{d\_min} \le d(a.x) \le V_{d\_max}$)
          ELSE ($V_{min} \le d(a.x) \le V_{max}$) FI
```

Figure 5.6: Formal code for *ContSATS*, Part 3 of 3

are performed. First, the condition $a.\mathsf{pos} = \mathsf{L}_*$ is added into the precondition where a is the aircraft moving by the transition and L_* is the length of the line on which a is currently located. This expresses that when the abstract movement of an aircraft between zones occur, that aircraft has to be on the intersection point of two lines (in other words, the end point of the line on which the aircraft is).

Second, an appropriate set_line effect is added to the effects of the transition so that the transition not only moves the aircraft in logical zones, but also re-assigns the lines on which the aircraft moves. An appropriate set_pos effect is also added to reset the pos value of the aircraft to 0 when the line is re-assigned.

Lastly, some specific transitions have a set_t effect added in their effects. As discussed in Section 5.3.1, when a specific transition gets enabled, the t value of the aircraft corresponding to that transition is re-assigned to the current value of now, and when the transition is performed, the t value is reset to -1. For example, HoldingPatternDescend sets the t value of the aircraft it moves if VerticalApproachInitiation (one of the transitions that have a time bound) gets immediately enabled after the transition. And VerticalApproachInitiation has a set_t effect that resets the t value to -1.

Analogously to the hybrid model of [12], by the precondition of the approach initiation transitions (VerticalApproachInitiation and LateralApproachInitiation), the model guarantees the initial separation distance of S_0 in the approach area between an aircraft initiating the approach and its immediate leader. The condition is expressed in terms of the D value of the leader aircraft as follows: $\mathsf{D}(\text{leader}(a,\mathsf{landing\_seq})) \geq \mathsf{S}_0$.

Three transitions inherited from the discrete model to $ContSATS$ have more differences than the above stated general modifications. This is because we modified these three transitions in order to more realistically represent a real system. The first transition modified is LowestAvailableAltitude. As discussed in the beginning of this section, the effects of LowestAvailableAltitude are modified so that it never exhibits a double transition. More specifically, the transition is modified as follows: if holding2 and holding3 are both empty, then the aircraft moves (its line is re-assigned) to holding2; otherwise it moves to holding3 *without checking* if holding3 is empty or not. Therefore only one aircraft moves at once in the logical zones by the transition. Recall that a double transition of aircraft in the discrete model occurs only when holding3 is not empty at the time LowestAvailableAltitude is performed. In addition, if holding3 is empty at the time LowestAvailableAltitude is performed, the effects of the transition are actually the same between the discrete model and $ContSATS$ (when we ignore the effects that just apply to $ContSATS$, such as set_line). The emptiness of holding3(σ) in the latter situation of the transition is crucial in the proof of the refinement (Theorem 5.13) to match up the effects of the transition in

$ContSATS$ and in the discrete model. Indeed, we guarantee this fact in the proof of refinement (Theorem 5.13) using Lemma 5.8.

The second transition modified is Landing. In the discrete model, the precondition of Landing checks if runway is empty when the transition is performed. In a continuous model, this is not reasonable considering that we cannot stop aircraft that have already started the final approach. Therefore we remove this condition in $ContSATS$. Instead, we have a time bound for `Taxiing` so that it removes aircraft frequently. We will guarantee (in the proof of refinement, Theorem 5.13, using Lemma 5.11) that this time bound indeed works for firing `Taxiing` frequently enough to avoid the landing when some aircraft is still on the runway.

The last transition modified is HoldingPatternDescend. The precondition of HoldingPatternDescend checks if holding2 is empty. Since a descending aircraft cannot stop and wait until holding2 gets empty in a real system, it is more reasonable to check this condition when they start descending by StartDescending, and not when HoldingPatternDescend is performed. Therefore, the emptiness condition of holding2 within the precondition of HoldingPatternDescend is removed, but the precondition of StartDescending does check this emptiness. When proving a refinement from $ContSATS$ to the discrete model of Chapter 2, we have to guarantee that, in $ContSATS$, holding2 is actually empty when HoldingPatternDescend is performed. We use Lemma 5.9 to prove this fact.

5.3.4 New Internal Transitions in $ContSATS$

In $ContSATS$, we add four new internal transitions (recall that the discrete model does not have any internal transitions). Since these are internal, they do not appear in traces of $ContSATS$. StartHolding3(σ) makes an aircraft that reaches the hovering point of holding3(σ) start hovering. Since this hovering occurs within a holding3 zone, the transition just re-assigns the line attribute of an aircraft from LINE_holding3ma to LINE_holding3hold, and does not change the logical-zone queues.

StartHolding2(σ) does a job analogous to StartHolding3(σ) for aircraft in holding2(σ).

StartDescending(σ) makes a hovering aircraft in holding3(σ) start moving again, as discussed before. Analogously to StartHolding3, the transition re-assigns the line attribute of an aircraft from LINE_holding3hold to LINE_holding3dec, and does not change the logical-zone queues.

SetTime sets the t value of an aircraft in holding2 to the current value of now when VerticalApproachInitiation for that aircraft becomes enabled. Note that VerticalApproachInitiation can be enabled by the leader aircraft of an aircraft in holding2 reaching the initial spacing position S_0. In such a case, as we will explain in Section 5.3.5, the trajectory will be stopped by the conditions specified in the **stop when** clause in the trajectory definition. Then, this internal transition, SetTime, takes care of setting a deadline for VerticalApproachInitiation. To perform the re-assignment of the t attribute of aircraft in holding2, we use the auxiliary function setTime. This function can be formally defined as follows. The auxiliary function setTimeAircraft sets the t attribute of aircraft to now if its leader just reaches the point S_0. Using this function, setTimeZone sets the t attribute of all aircraft in one zone. The function setTime sets the t attribute of aircraft in both holding2 zones using setTimeZone.

```
setTime(zones:zone_map, landing_sequence:queue, now:AugumentedReal) :=
    let zones' = assign(zones(holding2(right)),setTimeZone(zones(holding2(right)),landing_sequence, now)) in
    let zones''= assign(zones'(holding2(left)),setTimeZone(zones(holding2(left)), landing_sequence, now)) in
    zones''

setTimeZone(zone:queue, landing_sequence:queue, now:AugumentedReal):=
    IF NOT empty?(queue) THEN
    first(setTimeAircraft(rest(queue),landing_sequence, now), setTimeZone(cdr(queue),landing_sequence,now))
    ELSE queue

setTimeAircraft(a:Aricraft, landing_sequence:queue, now:AugumentedReal):=
    IF  a.t = -1 AND
        NOT first_in_seq_qn(a) AND
        on_approach_qn(leader(a,landing_seq)) AND
        D(leader(a,landing_seq)) = S0
    THEN  set_t(a, now)
    ELSE  a
```

5.3.5 Trajectories of *ContSATS*

The trajectory section of the TIOA code in Figure 5.6 defines the trajectories of *ContSATS*. This section consists of the **stop when** clause and the **evolve** clause.

stop when clause

This clause determines the condition when the trajectory is stopped, and thus some discrete transition has to be performed. In *ContSATS*, this clause is used to stop the trajectory in the following three ways: First, if an aircraft a reached the end point of the line on which it moves, the trajectory stops in order for some discrete action to be performed to re-assign the line attribute of a. This condition is stated in lines 3 - 5 of Figure 5.6.

Second, as explained in Section 5.3.3, the trajectory is stopped when the deadline for the specified time bound for the specific transitions is reached. More specifically, the trajectory is stopped when, for some aircraft a with its t value not equal to -1, $\mathsf{now} - a.\mathsf{r} \geq \mathsf{T}(a.\mathsf{line})$ is satisfied. The above stated inequality represents that at least $\mathsf{T}(a.\mathsf{line}$ time has elapsed since the transition for a became enabled. This condition is stated in lines 6 - 8 of Figure 5.6.

Third, the trajectory is stopped when VerticalApproachInitiation for aircraft a becomes enabled by the leader of aircraft a reaching the initial separation S_0. The trajectory need to be stopped in this situation because, as discussed in Section 5.3.4, the SetTime transition must be performed to set the `t` value of a to the current value of now in order to initiate a timer for VerticalApproachInitiation. This condition is stated in lines 9 - 15 of Figure 5.6. The condition in line 11 checks that a is in a `holding2` zone (for which VerticalApproachInitiation is performed). The condition in line 12 checks if the t value of a has not been set to the value of now. After the t value is set, this condition does not hold, and thus the trajectory can evolve without being stopped after the SetTime transition. The conditions in lines 13 - 15 check if the precondition for VerticalApproachInitiation is satisfied.

evolve clause

As stated in Section 5.3.2, the `now` state variable evolves at rate one.

As in [12], we assume that the velocity of the aircraft is bounded. We have two kinds of bounds depending on whether the aircraft is moving straight to some destination or is descending by circling down. Namely, $\mathsf{V_{dmin}} \leq d(a.\mathsf{pos}) \leq \mathsf{V_{dmax}}$ (where $0 < \mathsf{V_{dmin}} \leq \mathsf{V_{dmax}}$) holds for aircraft a when its line value is LINE_holding3dec(σ) for some σ, and $\mathsf{V_{min}} \leq d(a.\mathsf{pos}) \leq \mathsf{V_{max}}$ (where $0 < \mathsf{V_{min}} \leq \mathsf{V_{max}}$) holds otherwise. Two kinds of bounds are used since it is reasonable to consider that an aircraft flying toward some destination moves faster compared to an aircraft descending by circling down in the holding zones. These constraints are specified in Figure 5.6, lines 18-21 of the trajectory statement.[4]

5.3.6 Assumptions for *ContSATS*

We will assume the following conditions for *ContSATS* in this thesis (all assumptions and the constant definitions introduced in this subsection are summarized in Table 5.1).

First, considering that the SATS concept is created for increasing the volume of the landings by simultaneously operating the multiple landings, it is reasonable to assume that the initial separation of aircraft in the approach area is less than the total distance aircraft fly in that area, $\mathsf{S_0} < \mathsf{L_T}$, since otherwise, obviously, there can be at most one aircraft in the approach area.

For the next assumption, we use the following new constants. Let

$$\mathsf{S_T} = \mathsf{S_0} - (\mathsf{L_T} - \mathsf{S_0})\Delta,$$

where

$$\Delta = \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{min}}}$$

[4] The former velocity bound is used even for aircraft hovering at holding2hold or holding3hold. However, since the pos value of these aircraft is not used in the precondition or the effects of the transition, it does not make any difference if we used any other bound for the pos value of these hovering aircraft in the evolve statement.

We use this Δ in the rest of the thesis. As presented in Section 5.1, this length $\mathsf{S_T}$ is a lower bound on the spacing between two aircraft in the approach area model-checked for the hybrid model in [12]. We will state an analogous claim in Lemma 5.10. We assume that $\mathsf{S_T} > 0$. This is a necessary requirement for the system so that aircraft would never crash, since the bound is tight. Indeed, the spacing between two aircraft a and b becomes exactly $\mathsf{S_T}$ in the following scenario B:

Scenario B:

1. Aircraft a initiates the approach, and a moves in the approach area at its minimum possible speed $\mathsf{V_{min}}$.

2. Aircraft b initiates the approach exactly when a has flown the initial separation distance $\mathsf{S_0}$, and b moves in the approach area at its maximum possible speed $\mathsf{V_{max}}$.

3. When a reaches the end point of $\mathsf{LINE\_final}$, $\frac{(\mathsf{L_T}-\mathsf{S_0})}{\mathsf{V_{min}}}$ time has elapsed since b initiated the approach. Thus the distance between a and b has been reduced by $\frac{(\mathsf{L_T}-\mathsf{S_0})}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$ compared to the initial distance. Thus the distance between a and b at this point is $\mathsf{S_0} - \frac{(\mathsf{L_T}-\mathsf{S_0})}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}}) = \mathsf{S_T}$.

For an analogous reason, we assume $2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta > 0$. As we will prove in Section 5.6, the term in the left-hand side of the above inequality $(2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta)$ is a lower bound of the spacing between two aircraft in a missed approach zone. We can also easily check that this minimum spacing is also tight by examining the following scenario C:

Scenario C:

1. Aircraft a initiates the approach, and a moves in the approach area at its minimum possible speed $\mathsf{V_{min}}$.

2. Aircraft b initiates the approach exactly when a has flown the initial separation distance $\mathsf{S_0}$, and b moves in the approach area at any speed withing the specified velocity bound.

3. Aircraft c initiates the approach exactly when b has flown the initial separation distance $\mathsf{S_0}$, and c moves in the approach area at its maximum possible speed $\mathsf{V_{max}}$.

4. Aircraft a misses the approach, and goes to $\mathsf{maz}(\sigma)$. It continues moving at the speed of $\mathsf{V_{min}}$

5. Aircraft b misses the approach, and goes to $\mathsf{maz}(\mathit{opposite}(\sigma))$. (Recall that aircraft's mahf are assigned alternately. Thus b's mahf is assigned the opposite side from a's mahf)

6. Aircraft c misses the approach, and goes to $\mathsf{maz}(\sigma)$. It continues moving at the speed of $\mathsf{V_{max}}$

7. We can examine in a way analogous to Scenario B that, when a reaches the end point of $\mathsf{maz}(\sigma)$, the spacing between a and b is exactly $2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta$.

Thus the above inequality is a necessary condition of the system so that aircraft never crash.

In order to obtain a refinement from *ContSATS* to the discrete model, we have to assume the following two conditions. First, we assume the following inequality.

$$\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_3dec}}{V_{dmin}} < \frac{\mathsf{L_T} + \mathsf{L_M}}{\mathsf{V_{max}}}.$$

Informally, this says that the largest possible duration that an aircraft takes to go through a $\mathsf{holding3}$ zone is shorter than the smallest possible duration that an aircraft takes to go through the approach area and the missed approach zone.[5] This inequality is used to guarantee that $\mathsf{holding3}(\sigma)$ is always empty whenever $\mathsf{LowestAvailableAltitude}(\sigma)$ is performed.

We also assume $\mathsf{T_{Tax}} < \frac{\mathsf{S_T}}{\mathsf{V_{max}}}$. Informally, this inequality states that the $\mathsf{Taxiing}$ transition is performed frequently relative to the minimum spacing of aircraft in the approach area. It is used to guarantee that runway is empty whenever $\mathsf{Landing}$ is performed.

Constant 1.	$\Delta = \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{min}}}$
Constant 2.	$\mathsf{S_T} = \mathsf{S_0} - (\mathsf{L_T} - \mathsf{S_0})\Delta$
Assumption 1.	$\mathsf{S_T} > 0$
Assumption 2.	$2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta > 0$
Assumption 3.	$\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_3dec}}{V_{dmin}} < \frac{\mathsf{L_T} + \mathsf{L_M}}{\mathsf{V_{max}}}$
Assumption 4.	$\mathsf{T_{Tax}} < \frac{\mathsf{S_T}}{\mathsf{V_{max}}}$

Table 5.1: Constants and Assumptions of *ContSATS*

5.4 Basic invariants of *ContSATS*

Here we state and prove some basic invariants that we can prove straightforwardly by induction.

As in the case of the discrete model, the uniqueness of IDs in the logical zones is needed to prove other auxiliary invariants. The proof can be done in the exact same way as in the discrete model.

[5] Of course, from the definition of *ContSATS*, an aircraft in $\mathtt{holding3}(\sigma)$ can potentially hover at $\mathsf{LINE\_holding3hold}(\sigma)$ for an unbounded time as long as $\mathtt{holding2}(\sigma)$ is not empty. However, as we will discuss in the proof of Lemma 5.7, $\mathtt{holding2}(\sigma)$ is actually empty when an aircraft is in $\mathsf{holding3}(\sigma)$ and another aircraft is in $\mathtt{maz}(\sigma)$. From this fact, we can determine the bound on the duration that one aircraft takes to go through $\mathsf{holding3}(\sigma)$ when another aircraft is in $\mathtt{maz}(\sigma)$.

Lemma 5.1. (Uniqueness of ID of aircraft on the entire logical zones) *For any reachable state of $ContSATS$, the following two conditions hold:*

(i). (The uniqueness of ID on one zone) *For any logical zone z and natural numbers i and j where $i < j$ and both numbers are less than the length of z, the ID of the i-th positioned aircraft in the zone z and the ID of the j-th positioned aircraft in the zone z are different.*

(ii). (The uniqueness of ID between different zones) *For any two distinct logical zones z_1 and z_2, and any two aircraft $a \in z_1$ and $b \in z_2$, the ID of a and the ID of b are different.*

The following lemma states the consistency of the position of aircraft between the logical zone queues and the lines representing paths (represented as the line attribute of aircraft).

Lemma 5.2. *Let $\sim$ be the relation between z_name and Lines that is represented by the following pairs:*
holding2L $\sim$ LINE_holding2Lhold, holding2R $\sim$ LINE_holding2Rhold,
holding2L $\sim$ LINE_holding2Lma, holding2R $\sim$ LINE_holding2Rma,
holding3L $\sim$ LINE_holding3Lhold, holding3R $\sim$ LINE_holding3Rhold,
holding3L $\sim$ LINE_holding3Lma, holding3R $\sim$ LINE_holding3Rma,
holding3L $\sim$ LINE_holding3Ldec, holding3R $\sim$ LINE_holding3Rdec,
lezL $\sim$ LINE_lezL, lezR $\sim$ LINE_lezR,
mazL $\sim$ LINE_mazL, mazR $\sim$ LINE_mazR,
baseL $\sim$ LINE_baseL, baseR $\sim$ LINE_baseR,
intermediate $\sim$ LINE_intermediate,
final $\sim$ LINE_final, runway $\sim$ LINE_runway,
and any other two are not related.

For any reachable state of $ContSATS$, the following holds.

$$\forall a : \text{Aircraft } \forall z : \text{zone\_name},\ \text{on\_zone\_qn}(z, a) \Leftrightarrow z \sim a.\mathsf{line} \land (\exists z, \text{on\_zone\_qn}(z, a))$$

Proof. By induction. The initial case is trivial since there is no aircraft in any zone. Now we consider the inductive case. Since the trajectory does not affect either the line attribute of the aircraft or the logical zones expressed as queues, we prove the cases for the discrete transitions in the following.

Note that on_zone_qn(z, a) directly implies (on_zones_qn(a) $\lor$ on_zone_qn(runway$, a$)). Thus we prove $z \sim a.\mathsf{line}$ for the sufficient condition in the following.

For the entry transitions, the new aircraft gets assigned the appropriate line value with respect to its entering zone. The condition for the rest of the aircraft follows from the induction hypothesis.

If the transition re-assigns the `line` attribute of the aircraft, it is easy to see from the definition of the transitions that the re-assignment of the `line` attribute of the aircraft is done in a consistent way with the movement of it in the logical zone queues.

In the case of the transitions that remove one aircraft from the logical zones (Exit and Taxiing):
Lemma 5.1 implies that after the transition, the removed aircraft is not in the logical zones. Thus both the right-hand side and the left-hand side of the equivalent condition is evaluated to be false. Thus the condition holds for this aircraft. The condition for the rest of the aircraft holds from the induction hypothesis.

The SetTime transition does not affect either the line value of any aircraft or the logical zone queues. Thus the condition holds from the induction hypothesis. □

The following two invariants can be easily proved using the constraints by the **stop when** statement of the trajectory.

The first invariant states that the pos value of an aircraft will never increase beyond the actual length of the line the aircraft is on.

Lemma 5.3. *For any reachable state of $ContSATS$, the following holds.*

$$\forall a : \text{Aircraft}, (\exists z : \text{z\_name}, \text{on\_zone\_qn}(z, a)) \Rightarrow a.\mathsf{pos} \leq \mathsf{L}(a.\mathsf{line})$$

Proof. By induction. The condition holds in the initial state since there is no aircraft in the operation area. Now we consider the inductive case. We first consider the discrete transitions. When an aircraft enters the operation area, its pos is set to 0, which is smaller than $\mathsf{L}(a.\mathsf{line})$ values. Now consider the case that aircraft a is already in the operation area before the transition. All the transitions that affect pos set the value of pos to 0. Since $\mathsf{L}(a.\mathsf{line})$ is bigger than 0 for all zones, the condition holds after the transition. In the case when the transition does not affect pos of a, it is easy to see from the definition of the transition that it does not affect line attribute of a, either. Thus the condition immediately follows from the induction hypothesis.

Next let us consider the case of the trajectory. Suppose β is a closed trajectory of $ContSATS$ such that $\beta.fstate$ satisfies the condition of the lemma. From the **stop when** condition in the trajectory definition, the final state of β satisfies the condition of the lemma, as required. □

The following invariant states the bound on now when the timer is set.

Lemma 5.4. *For any reachable state of $ContSATS$, the following holds.*

$$\forall a : \text{Aircraft}, (\exists z : \text{z\_name}, \text{on\_zone\_qn}(z, a)) \wedge a.\mathsf{t} \neq -1 \Rightarrow 0 \leq \mathsf{now} - a.\mathsf{t} \leq \mathsf{T}(a.\mathsf{line})$$

Proof. By induction. The condition holds in the initial state since there is no aircraft in either the operation area or the runway.

Now we consider the inductive case. We first consider the discrete transitions. The SetTime transition can set some aircraft's t value to *now*. In this case, we have to check if $0 \leq \mathsf{now} - \mathsf{now} \leq \mathsf{T}(a.\mathsf{line})$. Since $\mathsf{T}(a.\mathsf{line})$ is bigger than 0 for any zone where an aircraft gets the t value other than -1, the condition holds after the transition. Each of the rest of the transitions sets t of one aircraft to either -1 or *now*, or does not affect t. The condition trivially holds for the transitions that set t to -1. In the case when the transition sets t of aircraft a to *now*, we can prove that the condition holds analogously to the case of SetTime. In the case when the transition does not affect the t value, it is easily proved by induction that $a.\mathsf{t}$ has the value -1 both before and after the transition. Thus the condition holds.

Next let us consider the case of the trajectory. Suppose β is a closed trajectory of $ContSATS$ such that $\beta.fstate$ satisfies the condition of the lemma. From the **stop when** condition in the trajectory definition, the final state of β satisfies the condition of the lemma, as required. □

5.5 Carrying over results from the discrete model using a refinement

In Chapter 3, we formally verified the safe separation of aircraft in terms of the number of aircraft in the logical zones. If we can carry over these results to $ContSATS$, we can guarantee the same safe separation as the discrete model. Some readers may wonder why we care about the separation verified in the discrete model, since we have obtained a finer continuous model. However, the consistency of the positions of aircraft between the lines and the logical-zone queues (proved as Lemma 5.2) tells us that these properties actually imply important spacing properties in $ContSATS$: For example, from the property that there is at most one aircraft in one holding3, we can guarantee that two aircraft would never get close in the same holding3 zone.

On the other hand, we cannot guarantee spacing properties of two aircraft in two adjacent zones from the properties of the discrete model. Some of these properties will actually be proved as auxiliary lemmas for the refinement (as step invariants). We also prove other such properties in Section 5.6.

We use the refinement technique presented in Chapter 4 to carry over the results from the discrete model to the continuous model. To make the discrete model of Chapter 2 (an ordinary I/O automaton) comparable to $ContSATS$ (a timed I/O automaton), we first construct $ExtSATS$, a natural extension of the discrete model to a timed I/O automaton. This extension can be done in the following generic way.

Generic extension scheme from an ordinary I/O automaton to a timed I/O automaton: Suppose A is an ordinary I/O automaton. We construct A' that is an timed extension (timed I/O automaton version) of A. First, in A', we add a new now state component to A which evolves at rate 1 (d(now) = 1). There is no **stop when** statement for A', and all discrete transitions are exactly the same as before the extension. From this straightforward extension, it is easy to see that all invariants of A are also invariants of A'.

By this extension, we obtained the following *ExtSATS*.

```
automaton ExtSATS
    imports SatsVocab    % ExtSATS uses the same vocablary as the discrete model.

signature: The same as the discrete model.

states
    zones : zone_map, % mapping from a zone name to a zone
    nextmahf : Side, % Next missed approach holding fix
    landing_seq : Zone % landing sequence is defined as a queue
    now :AugumentedReal % ** time variable is added **
    initially
        zones = initialZones ∧
        nextmahf = right ∧
        landing_seq = empty∧
        now = 0

    %% Auxiliary function definitions are exactly the same as the discrete model.

transitions : The same as the discrete model.

trajectories
    evolve
        d(now) = 1
```

5.5.1 Refinement mapping

To assert the fact that there is a refinement from *ContSATS* to *ExtSATS*, we define a refinement mapping from $Q_{ContSATS}$ to $Q_{ExtSATS}$, and prove that for every step of the concrete model, there is a sequence of corresponding steps of the specification.

A refinement is primarily used to show a trace inclusion, that is, every possible external behavior of the implementation automata (*ContSATS*, in our case) can be exhibited by the specification automata (*ExtSATS*). However, it indeed implies a stronger correspondence between the concrete model and the specification, as stated as Theorem 4.22 in Chapter 4. We use this correspondence property to show that every invariant of $ExtSATS$ is also an invariant of $ContSATS$.

Before defining a mapping, we first need to define the "equality" between a zone of $ContSATS$ and a zone of $ExtSATS$. Since the Aircraft types used are different for these two models, we have to define what "a zone in $ContSATS$ is equal to a zone in $ExtSATS$" means. The equality between zones components of $ContSATS$ and $ExtSATS$ is defined by the following zone-wise equality that ignores the new attributes of aircraft in $ContSATS$:

Definition 5.5. Two zone queues z and z' of $ContSATS$ and $ExtSATS$, respectively, are *equal*, denoted by $zone\_equal(z, z')$, if the following two conditions hold.

1. $\text{length}(z) = \text{length}(z')$.

2. For any position $i : 1 \leq i \leq \text{length}(z)$, $z[i].\mathsf{ID} = z'[i].\mathsf{ID} \wedge z[i].\mathsf{mahf} = z'[i].\mathsf{mahf}$.

Since type Zone is a queue of aircraft, we will use this equality between the landing sequences of $ContSATS$ and $ExtSATS$.

By using the above definition of equality between zones of $ContSATS$ and $ExtSATS$, we define the equality between the logical zones of $ContSATS$ and that of $ExtSATS$.

Definition 5.6. The logical zones $zones$ of $ContSATS$ is equal to the logical zones $zones'$ of $ExtSATS$, denoted by $zones\_equal(zones, zones')$ if for any zone name z, $zone\_equal(zones(z), zones(z'))$ holds.

One straightforward refinement mapping to consider (and actually the one we use for the refinement proof in Section 5.5.3 is the following mapping r from the states of $ContSATS$ to the states of $ExtSATS$: for all $s \in Q_{ContSATS}$, $r(s) = t$ such that

$$zones\_equal(s.\mathsf{zones}, t.\mathsf{zones}) \;\wedge\; s.\mathsf{nextmahf} = t.\mathsf{nextmahf} \;\wedge$$
$$zone\_equal(s.\mathsf{landing\_seq}, t.\mathsf{landing\_seq}) \;\wedge\; t.\mathsf{now} = s.\mathsf{now}.$$

This mapping maps a state of $ContSATS$ to a state of $ExtSATS$ so that every component of the state of $ExtSATS$ matches the corresponding component of the state of $ContSATS$. Note that such a state $r(s)$ in $ExtSATS$ is uniquely determined for every s in $ContSATS$ since the above conditions specify all components of $ExtSATS$.

In order to prove a refinement, we usually prove some auxiliary invariants of the models first, and then prove the main refinement using these invariants. In our case, as discussed in Section 5.3.3, we need invariants to guarantee, for instance, that $\mathsf{holding3}(\sigma)$ is empty when the $\mathsf{LowestAvailableAltitude}(\sigma)$ transition is performed. The *weak refinement* technique, presented formally in Section 4.2, has been developed to make use of the invariants of the *specification automaton* (the automaton to whose states a refinement maps), as well as those of the implementation automaton (the automaton from whose states a refinement maps), in the proof of the refinement. By using this technique, we can assume that invariants of the specification hold in the state considered in the inductive case of the refinement.

It turns out that we have to use a *weak refinement using a step invariant* presented in Section 4.2.2 so that we can use invariants of $ExtSATS$ in the proof of *invariants of* $ContSATS$. This is because in order to prove some invariants of $ContSATS$ – for example, the emptiness of $\mathsf{holding3}$ as stated above, we actually need some invariants of $ExtSATS$ that have been verified. Since we can assert the fact that the invariants of $ExtSATS$ also hold for $ContSATS$ *only after* proving the refinement, we need a refinement using a step invariant to avoid circular reasoning.

In Section 5.5.2, we prove invariants of $ContSATS$ in a form that, when we combine all of the invariants as one conjunction, they form a step invariant defined in Section 4.2.2.

5.5.2 Auxiliary invariants needed for refinement proof

Now we start proving the auxiliary invariants of $ContSATS$ needed for the refinement proof.

The following Condition Φ states the conditions needed to prove the auxiliary invariants of $ContSATS$ in this subsection. As we will see in Section 5.5.3, this Φ corresponds to $\lambda s.P_B(r(s))$ in the definition of a weak refinement using a step invariant in Section 4.2.2. In the rest of the thesis, we refer to the first condition of Φ by Φ.1, the second condition by Φ.2, and so on.

It is easy to see that every condition in Φ is actually an invariant of the discrete model $SATS$, and therefore an invariant of $ExtSATS$. Some of them are exactly the same as the main properties of the $SATS$ automaton that we proved in Chapter 3: The conditions Φ.1 and Φ.2 are Property 3 and Property 6. The conditions Φ.6 and Φ.7 follow from Property 2, which states the number of aircraft in an initiation area of one side is at most two.

The rest of the properties follow from the auxiliary lemmas we have proved in Chapter 3. This fact indicates that, by proving these auxiliary invariants, the assertional style proof gives us more insight into how the system works than the exhaustive state exploration. The condition Φ.3 states the condition proved as Corollary 3.8. The condition Φ.4 easily follows from Case 1 and Case 3 of Lemma 3.26. The condition Φ.5 follows from Case 4 of Lemma 3.26.

Condition Φ:

1. $\forall \sigma : \text{side},\ \text{length}(\mathsf{holding3}(\sigma)) \leq 1 \wedge \text{length}(\mathsf{holding2}(\sigma)) \leq 1$

2. $\forall \sigma : \text{side},\ \neg\text{empty\_qn}(\mathsf{lez}(\sigma)) \Rightarrow$
   $\text{empty\_qn}(\mathsf{holding2}(\sigma)) \wedge \text{empty\_qn}(\mathsf{holding3}(\sigma)) \wedge \text{empty\_qn}(\mathsf{maz}(\sigma))$

3. $\text{first}(\mathsf{final}) = \text{first}(\mathsf{landing\_seq})$

4. $\forall\sigma : \text{side},\ (\text{on\_approach\_qn}(\sigma) \wedge \neg\text{empty\_qn}(\mathsf{maz}(\sigma))) \Rightarrow \text{empty\_qn}(\mathsf{holding3}(\sigma))$

5. $\forall\sigma : \text{side}, \text{on\_approach\_qn}(\sigma) \Rightarrow\ \text{length}(\mathsf{holding2}(\sigma)) + \text{length}(\mathsf{holding3}(\sigma)) \leq 1$

6. $\forall\sigma : \text{side}, \mathit{length}(\mathsf{maz}(\sigma)) \geq 2 \Rightarrow \text{empty\_qn}(\mathsf{holding2}(\sigma)) \wedge \text{empty\_qn}(\mathsf{holding3}(\sigma))$

7. $\forall\sigma : \text{side}, \neg\text{empty\_qn}(\mathsf{maz}(\sigma))) \Rightarrow\ \text{length}(\mathsf{holding2}(\sigma)) + \text{length}(\mathsf{holding3}(\sigma)) \leq 1$

The invariants in this subsection are stated in a way that, when we combine all the invariants as one conjunction, they form a step invariant using Φ.

We start by proving two invariants, Lemma 5.7 and Lemma 5.8, which are used to prove whenever $\mathsf{LowestAvailableAltitude}(\sigma)$ is performed, $\mathsf{holding3}(\sigma)$ is empty. With these two invariants, we bound the distance that a missed aircraft has flown in $\mathsf{maz}(\sigma)$ when $\mathsf{holding3}(\sigma)$ is not empty. Informally, we will show that no aircraft in $\mathsf{maz}(\sigma)$ can catch up with an aircraft in $\mathsf{holding3}(\sigma)$. The important observation needed to prove this fact is that from $\Phi.7$: if there are some aircraft in both $\mathsf{maz}(\sigma)$ and $\mathsf{holding3}(\sigma)$, then $\mathsf{holding2}(\sigma)$ is empty. This implies that an aircraft hovering at $\mathsf{holding3hold}(\sigma)$ can start descending within the time bound T_3 after another aircraft enters $\mathsf{maz}(\sigma)$. This fact gives us the following upper bound on the pos value of aircraft a on $\mathsf{maz}(\sigma)$ when $\mathsf{holding3}(\sigma)$ is not empty: $a.\mathsf{pos} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T}_3 + \frac{\mathsf{L_{3dec}}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}}$. The term in the right-hand side of the inequality describes the maximum time that an aircraft takes to go through the entire $\mathsf{holding3}(\sigma)$ zone (the time to go through $\mathsf{holding3ma}(\sigma)$; plus the time bound on the hovering; plus the time to go through $\mathsf{holding3dec}(\sigma)$) times the maximum possible velocity of a. The condition $\Phi.5$ gives us further information to improve this bound: if there are some aircraft assigned σ on the approach, and there are some aircraft in $\mathsf{holding3}(\sigma)$, then $\mathsf{holding2}(\sigma)$ is empty. This indicates that $\mathsf{holding2}(\sigma)$ has been empty since an aircraft in $\mathsf{maz}(\sigma)$ first initiated its final approach. Using this fact, we will improve the above stated bound to $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T}_3 + \frac{\mathsf{L_{3dec}}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}} - \mathsf{L_t}$.

To formally capture the above discussion about $\Phi.5$, we have the first invariant, Lemma 5.7, and we state the final bound in Lemma 5.8 and prove it using Lemma 5.7. In both Lemmas 5.7 and 5.8, we have three different cases depending on which line or point in holding3 the aircraft is on.

Lemma 5.7 states the upper bound of D(a) for aircraft a on the approach when an aircraft b is in holding2. There are three conclusions (1) (2) and (3) in Lemma 5.7. The three conclusions state the bound on D(a) in different situations: Conclusion (1) is for the case when b is in LINE_holding3ma; Conclusion (2) is for the case when b is in LINE_holding3hold; and Conclusion (3) is for the case when b is in LINE_holding3dec.

Lemma 5.8 has the exact same three conclusions as Lemma 5.7, but has a different assumption: In Lemma 5.8, we assume aircraft a is in maz and aircraft b is in $\mathsf{holding3}$. Thus, Lemma 5.7 states the "pre situation" of the situation that we assume in Lemma 5.8 before a moves to maz by the `MissedApproach` transition.

Lemma 5.7. *Consider a reachable state s of $ContSATS$ that satisfies Conditions Φ and the following Condition A_1.*

$$
\begin{aligned}
(\mathbf{A_1})\ :&\forall a, b : \text{Aircraft}, \forall \sigma : \text{side},\\
&\quad \text{on\_approach\_qn}(a) \wedge a.\mathsf{mahf} = \sigma \wedge \text{on\_zone\_qn}(\mathsf{holding3}(\sigma), b)\\
&\quad \Rightarrow (1) \wedge (2) \wedge (3)
\end{aligned}
$$

$$
\begin{aligned}
&(1)\ b.\mathsf{line} = \mathsf{LINE\_holding3ma}(\sigma) \Rightarrow \mathsf{D}(a) \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}.\\
&(2)\ b.\mathsf{line} = \mathsf{LINE\_holding3hold}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}}.\\
&(3)\ b.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}}.
\end{aligned}
$$

Let α be a step of $ContSATS$ starting with s. Then $\alpha.lstate$ satisfies Condition A_1.

Proof. **Case 1:** We first consider the case when α consists of one discrete transition. In this proof, we focus on proving the condition for the aircraft affected by the transition since the condition for the rest of aircraft immediately follows from A_1 in s.

- $\mathsf{LowestAvailableAltitude}(\sigma)$: If $\mathsf{holding2}(\sigma)$ is empty before the transition, the aircraft moves to that zone by the transition. Thus the transition does not affect $\mathsf{holding3}(\sigma)$ or the approach area, and thus the condition immediately follows from A_1 that holds before the transition. If $\mathsf{holding2}(\sigma)$ is not empty before the transition, from Φ.4, $\mathsf{maz}(\sigma)$ is empty. This contradicts with the precondition of the transition.

- $\mathsf{VerticalApproachInitiation}(\sigma)$: The aircraft a that initiates the approach by this transition is set its pos value to 0. Thus $\mathsf{D}(a) = 0$ after the transition. Since the pos value of every aircraft is more than or equal to 0, the condition holds.

- $\mathsf{LateralApproachInitiation}(\sigma)$: From the precondition of the transition, $\mathsf{lez}(\sigma)$ is not empty before the transition. It follows from Φ.2 that $\mathsf{holding3}(\sigma)$ is empty before the transition and also after the transition since it does not affect $\mathsf{holding3}(\sigma)$.

The rest of the transitions do not move an aircraft to the approach zone or set the line value of an aircraft to $\mathsf{holding3ma}(\sigma)$. Thus the condition immediately follows from the same condition A_1 that holds in s.

Now we consider the case of the trajectory. Let δ be the duration of α. Let D_a and D'_a be the value of $\mathsf{D}(a)$ in states s and $\alpha.lstate$, respectively. Let $b.pos$ and $b.pos'$ be the value of $b.\mathsf{pos}$ in states s and $\alpha.lstate$, respectively. From the **evolve** statement of the trajectory, $b.pos' \geq b.pos + \delta \cdot \mathsf{V_{min}}$ and $D'_a \leq D_a + \delta \cdot \mathsf{V_{max}}$ hold. We can easily show $D'_a \leq \frac{b.pos'}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$ follows from $D'_a \leq \frac{b.pos}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$ using the above two inequalities.

Case 2: We first consider the case when β consists of one discrete transition.

- $\mathsf{VerticalEntry}(\sigma)$: When there are some aircraft assigned σ on the approach, the transition is disabled.

- $\mathsf{StartHolding3}(\sigma)$: This transition changes the line value of one aircraft from $\mathsf{LINE\_holding3ma}(\sigma)$ to
$\mathsf{LINE\_holding3hold}(\sigma)$. The precondition for the transition ensures that the pos value of the aircraft that starts holding is $\mathsf{L_{3ma}}$. Since this aircraft had the line value of $\mathsf{LINE\_holding3ma}(\sigma)$, it follows from the first condition of A_1 that, before the transition, for any aircraft a assigned σ on the approach, $\mathsf{D}(a) \leq \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$. From $\Phi.5$ and the fact that there are at least one aircraft assigned σ on the approach and one aircraft at $\mathsf{holding3}(\sigma)$, there is no aircraft in $\mathsf{holding2}(\sigma)$ before and also after the transition, since it does not affect $\mathsf{holding2}(\sigma)$. It implies that the t value is set to now after the transition, and thus $\mathsf{now} - b.\mathsf{t} = 0$. Therefore, it is sufficient to show that $\mathsf{D}(a) \leq \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$ holds for any aircraft a assigned σ on the approach. This inequality immediately follows from $\mathsf{D}(a) \leq \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$.

- $\mathsf{VerticalApproachInitiation}(\sigma)$: For the aircraft a that initiates the approach, $\mathsf{D}(a) = 0$ holds after the transition. In addition, from Lemma 5.4, $\mathsf{now} - b.\mathsf{t} \geq 0$ holds for any aircraft b in $\mathsf{holding3}(\sigma)$. Since $\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} \geq 0$, the required inequality holds.

- $\mathsf{LateralApproachInitiation}(\sigma)$: We can use the same discussion as in Case 1 to show that $\mathsf{holding3}(\sigma)$ is empty.

The rest of the transitions do not move an aircraft to the approach zone or set the line value of an aircraft to $\mathsf{LINE\_holding3hold}(\sigma)$. Thus the condition immediately follows from the same condition A_1 that holds in s.

The trajectory case can be proved analogously to Case 1. In this case, we have the exact value for now in the last state of the trajectory: if the trajectory is of length δ, then the value of now increases exactly by δ. Using this fact and a similar bound on $a.\mathsf{pos}$ as in Case 1, we can easily obtain the required inequality.

Case 3: We first consider the case when β consists of one discrete transition.

- $\mathsf{StartDescending}(\sigma)$: Suppose aircraft b starts descending by this transition. From Lemma 5.4, $\mathsf{now} - b.\mathsf{t} \leq \mathsf{T}_3$ holds in state s. From the second condition of Condition A_1, $\mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}}$ holds in s. It follows from these two inequality that $\mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T}_3) \cdot \mathsf{V_{max}}$ Considering that the pos value of b is set to 0 after the transition, this inequality is exactly what we require.

- $\mathsf{VerticalApproachInitiation}(\sigma)$: We can use the same discussion to prove the condition as in Case 2 using the fact that $\mathsf{D}(a) = 0$ for the aircraft a that initiate the approach after the transition, and $(-\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} - \mathsf{T}_3)\ \mathsf{V_{d\_min}} \leq 0$.

- $\mathsf{LateralApproachInitiation}(\sigma)$: We can use the same discussion as in Case 1 to show that $\mathsf{holding3}(\sigma)$ is empty.

The rest of the transitions do not move an aircraft to the approach zone or set the line value of an aircraft to $\mathsf{holding3dec}(\sigma)$. Thus the condition immediately follows from the same condition A_1 that holds in s.

The trajectory case can be proved analogously to Case 1. □

As we discussed in the explanation of Lemma 5.7, we assume in Lemma 5.8 a different situation than Lemma 5.7, but it has the exact same three conclusions. The situation that we assume in Lemma 5.8 states the "post situation" that aircraft a that was on the approach in Lemma 5.7 has moved to maz by $\mathsf{MissedApproach}$. This is why we use Condition A_1 defined in Lemma 5.7 to prove the inductive step of the conclusions of Lemma 5.8 for $\mathsf{MissedApproach}$.

Lemma 5.8. *Consider a reachable state s of $ContSATS$ that satisfies Conditions Φ, Condition A_1 in Lemma 5.7, and the following Condition A_2.*

$(\mathbf{A_2})$:$\forall a, b$: Aircraft, $\forall \sigma$: side, on_zone_qn($\mathsf{maz}(\sigma), a$) $\wedge$ on_zone_qn($\mathsf{holding3}(\sigma), b$) $\Rightarrow$

(1) $\wedge$ (2) $\wedge$ (3)

$$(1)\ b.\mathsf{line} = \mathsf{LINE\_holding3ma}(\sigma) \Rightarrow \mathsf{D}(a) \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}.$$

$$(2)\ b.\mathsf{line} = \mathsf{LINE\_holding3hold}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}}.$$

$$(3)\ b.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T}_3 + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}}.$$

Let α be a step of ContSATS starting with s. Then α.lstate satisfies Condition A_2.

Proof. As in the proof of Lemma 5.7, we focus on proving the condition for the aircraft affected by the transition since the condition for the rest of aircraft immediately follows from A_2 in s.

Since $\mathsf{D}(a) = \mathsf{L_T} + a.\mathsf{pos}$ for aircraft a in maz, it is sufficient to prove the following three conditions:

$$(1)\ b.\mathsf{line} = \mathsf{LINE\_holding3ma}(\sigma) \Rightarrow a.\mathsf{pos} \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}} - \mathsf{L_T}.$$

$$(2)\ b.\mathsf{line} = \mathsf{LINE\_holding3hold}(\sigma) \Rightarrow a.\mathsf{pos} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}} - \mathsf{L_T}.$$

$$(3)\ b.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma) \Rightarrow a.\mathsf{pos} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}} - \mathsf{L_T}.$$

Case 1: We first consider the case when α consists of one discrete transition.

- MissedApproach: Let a be the aircraft that misses the approach, and b be the aircraft that has a line attribute of LINE_holding3ma(σ). From the precondition of the transition, $\mathsf{D}(a) = \mathsf{L_T}$ holds in s. Since a was on the approach before the transition, it follows from the first condition of A_1 that $b.\mathsf{pos} \geq \frac{\mathsf{L_T}}{\mathsf{V_{max}}} \cdot \mathsf{V_{min}}$ in s. From this inequality, we obtain $\frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}} - \mathsf{L_T} \geq 0$. Since $a.\mathsf{pos}$ is set to 0 after the transition, and $b.\mathsf{pos}$ does not change by the transition, the above inequality is exactly what we require.

- LowestAvailableAltitude: If holding2(σ) is empty, the aircraft moves to that zone. Thus holding3(σ) or the approach area does not change by the transition. Thus the condition immediately follows

  Now suppose holding2(σ) is not empty. If there are at least one aircraft in maz(σ) after the transition, it implies that there are at least two aircraft in that zone before it. This contradicts to Φ.6.

The rest of the transitions do not affect maz(σ) or set the *zone* attribute of aircraft to LINE_holding3ma(σ). Thus the condition immediately follows from the same condition A_2 that holds before the transition.

The proof for the trajectory case can be easily done by a discussion analogous to Lemma 5.7 using the bound on the change of $a.\mathsf{pos}$ and $b.\mathsf{pos}$ between the first and last state of the trajectory.

Case 2: We first consider the case when β consists of one discrete transition.

- MissedApproach: The proof can be done in a way similar to Case 1, using Case 2 of A_1 instead of Case 1 of A_1. Let a be the aircraft that misses the approach, and b be the aircraft that has a line attribute of $\mathsf{LINE\_holding3}(\sigma)$. From the precondition of the transition, $\mathsf{D}(a) = \mathsf{L_T}$ holds in s. Since a was on the approach before the transition, it follows from Case 2 of A_1 that $\mathsf{now} - b.\mathsf{t} \geq \frac{\mathsf{L_T}}{\mathsf{V_{max}}} - \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}}$. From this inequality, we obtain $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}} - \mathsf{L_T} \geq 0$ Since $a.\mathsf{pos}$ is set to 0 after the transition, the above inequality is exactly what we need.

- $\mathsf{VerticalEntry}(\sigma)$: If $\mathsf{maz}(\sigma)$ is not empty, this transition is disabled.

- $\mathsf{StartHolding}(\sigma)$: The proof is similar to Case 2 of Lemma 5.7. If $\mathsf{holding3}(\sigma)$ and $\mathsf{maz}(\sigma)$ are both not empty, it follows from $\Phi.7$ that $\mathsf{holding2}(\sigma)$ is empty. Therefore the t attribute of the aircraft b that starts holding is set to now after the transition. In addition, from the precondition of the action, $b.\mathsf{pos} = \mathsf{L_{3ma}}$ holds before the transition. Now let a be an arbitrary aircraft in $\mathsf{maz}(\sigma)$. From Case 1 of Condition A_2 and the fact that $b.\mathsf{pos} = \mathsf{L_{3ma}}$, $a.\mathsf{pos} \leq \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}} - \mathsf{L_T}$ holds before the transition. Considering that $\mathsf{now} - b.\mathsf{t} = 0$ after the transition, and that $a.\mathsf{pos}$ does not change by the transition, the above inequality is exactly what we need.

The rest of the transitions do not affect $\mathsf{maz}(\sigma)$ or set the *zone* attribute of an aircraft to $\mathsf{LINE\_holding3}(\sigma)$. Thus the condition immediately follows from the same condition A_2 that holds before the transition.

The proof for the trajectory case can be easily done analogously to Case 1.

Case 3: We first consider the case when β consists of one discrete transition.

- $\mathsf{MissedApproach}$: The proof is similar to Case 1 and Case 2. In this case we use Case 3 of Lemma 5.7. Let a be the aircraft that misses the approach, and b be the aircraft that has a line attribute of $\mathsf{LINE\_holding3}(\sigma)$. From the precondition of the transition, $\mathsf{D}(a) = \mathsf{L_T}$ holds in s. Since a was on the approach before the transition, it follows from Case 3 of A_1 that $b.\mathsf{pos} \geq (\frac{\mathsf{L_T}}{\mathsf{V_{max}}} - \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} - \mathsf{T})\ \mathsf{V_{d\_min}}$ before the transition. This inequality is equivalent to $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T} + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}})\mathsf{V_{max}} - \mathsf{L_T} \geq 0$. Considering that $a.\mathsf{pos}$ is set to 0, the above inequality is exactly what we need.

- $\mathsf{StartDescending}(\sigma)$: Let a be any aircraft in $\mathsf{maz}(\sigma)$, and b be the aircraft that starts descending. From Lemma 5.4, $\mathsf{now} - b.\mathsf{t} \leq \mathsf{T}$ holds before the transition. And from Case 2 of Condition A_2, $a.\mathsf{pos} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}} - \mathsf{L_T}$ holds in s. Using these two inequalities, we obtain $a.\mathsf{pos} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T}) \cdot \mathsf{V_{max}} - \mathsf{L_T}$. Considering that $b.\mathsf{pos}$ is set to 0 after the transition and $a.\mathsf{pos}$ does not change by the transition, the above inequality is exactly what we need.

The rest of the transitions do not affect $\mathsf{maz}(\sigma)$ or set the *zone* attribute of an aircraft to $\mathsf{LINE\_holding3dec}(\sigma)$. Thus the condition immediately follows from the same condition A_2 that holds before the transition.

The proof for the trajectory case can be easily done analogously to Case 1. □

The following lemma is used in the case of HoldingPatternDescend in the refinement proof. As discussed in Section 5.3.3, the $\mathsf{HoldingPatternDescend}(\sigma)$ transition has been modified from the discrete model so that it no longer checks the emptiness of $\mathsf{holding2}(\sigma)$: instead, an aircraft does so when it starts descending ($\mathsf{StartDescending}(\sigma)$). Thus, we have to guarantee that holding2 is empty when the transition is performed in $ContSATS$, in order to obtain the correspondence step of this transition in $ExtSATS$ in the refinement. We assert by the following invariant that whenever some aircraft is descending from $\mathsf{holding3}(\sigma)$ to $\mathsf{holding2}(\sigma)$, $\mathsf{holding2}(\sigma)$ is empty.

Lemma 5.9. *Consider a reachable state s of $ContSATS$ that satisfies Conditions Φ and the following condition B.*

$$(\mathbf{B}) : \forall a : \text{Aircraft}, \quad \forall \sigma : \text{side},$$
$$(\text{on\_zone\_qn}(\mathsf{holding3}(\sigma)) \wedge a.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma)) \Rightarrow \text{empty\_qn}(\mathsf{holding2}(\sigma)).$$

Let α be a step of $ContSATS$ starting with s. Then $\alpha.lstate$ satisfies the condition B.

Proof. Since the zone queues and the line attribute of the aircraft does not change by a trajectory, it is sufficient to prove the condition for execution fragments with one discrete transition.

- $\mathsf{StartDescending}(\sigma)$: From the precondition of the transition, $\mathsf{holding2}(\sigma)$ is empty.
- $\mathsf{HoldingPatternDescend}(\sigma)$: From the condition $\Phi.1$, there is exactly one aircraft in $\mathsf{holding3}(\sigma)$ before the transition. Thus there is no aircraft in the zone after the transition. Therefore the condition holds.
- $\mathsf{LateralApproachInitiation}(\sigma)$: From the condition $\Phi.2$ and the precondition of the transition, there is no aircraft in $\mathsf{holding3}(\sigma)$. Thus the condition holds after the transition.

The rest of the transitions do not add an aircraft to $\mathsf{holding2}(\sigma)$, or change the line attribute of an aircraft to $\mathsf{LINE\_holding3dec}(\sigma)$. Thus the condition holds from the same condition B that holds before the transition. □

The next invariant states the lower bound on the spacing between an aircraft and its immediate leader in the approach area. The separation is defined in terms of the difference between the

D values of two aircraft. It actually states a generalized bound on the spacing in the approach area than the bound model-checked in [12]. Indeed, if we substitute the largest possible value $\mathsf{L_T}$ of $\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq}))$, then we have the same bound of $\mathsf{S_T} = \mathsf{S_0} - \frac{\mathsf{L_T} - \mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$. We use this property to prove Lemma 5.11.

Lemma 5.10. *Consider a reachable state s of ContSATS that satisfies Conditions Φ and the following Condition C_1.*

$(\mathbf{C_1})$:$\forall a$: Aircraft,

$$(\text{on\_approach\_qn}(a) \wedge \neg\text{first\_in\_seq\_qn}(a)) \Rightarrow$$

$$\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) - \mathsf{D}(a) \geq \mathsf{S_0} - \frac{\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) - \mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}}).$$

Let α be a step of ContSATS starting with s. Then α.lstate satisfies Condition C_1.

Proof. We first prove the condition for the discrete transitions.

- $\mathsf{VerticalApproachInitiation}(\sigma)$: the precondition of the transition ensures that $\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) \geq \mathsf{S_0}$. Since $\mathsf{D}(a) = 0$ after the transition,

$$\begin{aligned} & \mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) - \mathsf{D}(a) \\ = \; & \mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) \\ \geq \; & \mathsf{S_0} \\ \geq \; & \mathsf{S_0} - \frac{\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq})) - \mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}}). \end{aligned}$$

  Thus the condition holds.

- $\mathsf{LateralApproachInitiation}(\sigma)$: If the aircraft moves to the approach area, we can use the same discussion to prove the condition as in the case of $\mathsf{VerticalApproachInitiation}$. If the aircraft moves to $\mathsf{holding2}$, the condition immediately follows from Condition C_1 that holds before the transition. since the transition does not affect the approach area.

- Exit: From the uniqueness of the aircraft, the exiting aircraft is no longer in the approach area after the transition. Thus the condition for the aircraft holds. The precondition of the transition guarantees that the exiting aircraft is the first aircraft in the landing sequence. It follows that the aircraft that immediately follows the exiting aircraft becomes the first aircraft of the landing sequence after the transition. Thus the condition for this new first aircraft trivially holds. The conditions for the rest of the aircraft follows from Condition C_1 that holds before the transition.

- $\mathsf{Landing}$ and MissedApproach: From the condition $\Phi.3$, the first aircraft of final is the first aircraft of the landing sequence. Thus we can use the same discussion as in the case of Exit.

- $\mathsf{Merging}$ and $\mathsf{FinalSegment}$ These transitions change the line and pos values of one aircraft on the approach, say a, but it is easy to see that from the definition of the function D, these changes does not affect the value of $\mathsf{D}(a)$. The conditions for the rest of the aircraft follows from Condition C_1 that holds before the transition.

The rest of the transitions do not affect the approach area, and thus do not change the value of D of the aircraft in the area. Thus Condition C_1 for the rest of the aircraft after the transition follows from the same Condition C_1, which holds before the transition.

Now we prove Condition C_1 for trajectories. Let b be the leader of a in the landing sequence, and δ be the duration of the closed trajectory in β. From the "evolve" statement defined in the trajectory, the following conditions hold, where $a.x$ is the value of $a.\mathsf{pos}$ in the first state of the trajectory, and $a.x'$ is the value of $a.\mathsf{pos}$ in the last state of the trajectory. $b.x$ and $b.x'$ are defined analogously.

$$\begin{aligned} a.x' &\leq a.x + \mathsf{V}_{\mathsf{max}} \cdot \delta, \quad \text{and} \\ b.x' &\geq b.x + \mathsf{V}_{\mathsf{min}} \cdot \delta. \end{aligned}$$

Thus, from the definition of the function D,

$$\begin{aligned} D'_a &\leq D_a + \mathsf{V}_{\mathsf{max}} \cdot \delta, \quad \text{and} \\ D'_b &\geq D_b + \mathsf{V}_{\mathsf{min}} \cdot \delta, \end{aligned}$$

where D_a and D'_a represents the value of $\mathsf{D}(a)$ in the first and the last states of the trajectory, respectively. D_b and D'_b are defined analogously.

Thus,

$$\begin{aligned} D'_b - D'_a &\geq D_b + \mathsf{V}_{\mathsf{min}} \cdot \delta - D_a - \mathsf{V}_{\mathsf{max}} \cdot \delta \\ &= (D_b - D_a) + \mathsf{V}_{\mathsf{min}} \cdot \delta - \mathsf{V}_{\mathsf{max}} \cdot \delta \\ &\geq \mathsf{S}_0 - \frac{D_b - \mathsf{S}_0}{\mathsf{V}_{\mathsf{min}}}(\mathsf{V}_{\mathsf{max}} - \mathsf{V}_{\mathsf{min}}) + \mathsf{V}_{\mathsf{min}} \cdot \delta - \mathsf{V}_{\mathsf{max}} \cdot \delta \quad \text{(from the induction hypothesis)} \\ &\geq \mathsf{S}_0 - \frac{D'_b - \mathsf{V}_{\mathsf{min}} \cdot \delta - \mathsf{S}_0}{\mathsf{V}_{\mathsf{min}}}(\mathsf{V}_{\mathsf{max}} - \mathsf{V}_{\mathsf{min}}) + \mathsf{V}_{\mathsf{min}} \cdot \delta - \mathsf{V}_{\mathsf{max}} \cdot \delta \\ &= \mathsf{S}_0 - \frac{D'_b - \mathsf{S}_0}{\mathsf{V}_{\mathsf{min}}}(\mathsf{V}_{\mathsf{max}} - \mathsf{V}_{\mathsf{min}}). \end{aligned}$$

Therefore Condition C_1 holds for the trajectory. □

The following invariant is used in the case of Landing in the refinement proof. As discussed in Section 5.3.3, Landing in *ContSATS* does not check if runway is empty or not. Instead, we set a time bound for Taxiing so that the system removes aircraft from runway frequently enough. This invariant helps us to show, in the refinement proof, that the zone is empty whenever an aircraft lands.

Lemma 5.11. *Consider a reachable state s of ContSATS that satisfies Conditions* Φ*, Condition* C_1 *in Lemma 5.10, and the following Condition* C_2*.*

$$(\mathbf{C_2}) : \forall a, b : \text{Aircraft}, (\text{on\_zone\_qn}(\mathsf{runway}, \text{a}) \ \wedge \ \text{on\_approach\_qn}(\text{b})) \Rightarrow$$

$$\mathsf{now} - a.\mathsf{t} \geq \frac{\mathsf{D}(b) - (\mathsf{L_T} - \mathsf{S_T})}{\mathsf{V_{max}}}$$

Let α *be a step of ContSATS starting with s. Then* α*.lstate satisfies Condition* C_2*.*

Proof. There are really four cases that we have to check. Three discrete transitions (Landing, VerticalApproachInitiation, and LateralApproachInitiation), and the trajectory case. The Taxiing transition also affects the runway zone, but it does so in a preferable way with respect to the lemma: it removes an aircraft from runway. The rest of the transitions does not add or remove the aircraft from either the approach area or runway, and thus the condition immediately follows from the same condition C_2 that holds before the transition.

- Landing: The first aircraft of the final zone, say a, lands by this transition. From the condition Φ.3, a is the first aircraft of the landing sequence. Thus it precedes any aircraft that is in the sequence. In addition, from Condition C_1 that holds before the transition, any aircraft and its leader has a separation of at least S_T if both are on the approach. It implies that there is a separation of at least S_T between a and any other aircraft b in the approach area. Therefore $\mathsf{D}(b) - (\mathsf{L_T} - \mathsf{S_T}) \geq (\mathsf{D}(a) - \mathsf{S_T}) - (\mathsf{L_T} - \mathsf{S_T}) = \mathsf{D}(a) - \mathsf{L_t}$. Since $\mathsf{D}(a) = \mathsf{L_T}$ holds from the precondition, $\mathsf{D}(b) - (\mathsf{L_T} - \mathsf{S_T}) \geq 0$ before the transition. Since $\mathsf{D}(b)$ does not change by the transition, the same inequality holds after the transition. Considering that $a.\mathsf{t}$ is set to now by the transition, which gives us $\mathsf{now} - a.\mathsf{t} = 0$, and the above inequality, we can obtain the required inequality. The condition for the rest of the aircraft follows from the same condition C_2 that holds before the transition.

- VerticalApproachInitiation: Let b be the aircraft that initiates the approach. After the transition, $\mathsf{D}(b)$ becomes 0. Since $\mathsf{L_T} \geq \mathsf{S_T}$, $-\frac{\mathsf{L_T} - \mathsf{S_T}}{\mathsf{V_{max}}}$ is non-positive. Considering that, from Lemma 5.4, $\mathsf{now} - a.\mathsf{t} \geq 0$ if $a.\mathsf{t} \neq -1$, the required inequality holds. The condition for the rest of the aircraft follows from the same condition C_2 that holds before the transition.

- LateralApproachInitiation: If the moving aircraft really initiates the approach (that is, it moves to the approach area), we can use the same discussion to the case of VerticalApproachInitiation. If the aircraft moves to holding2, the transition does not affect the approach area. Thus the condition follows from the same condition C_2 that holds before the transition.

- The trajectory: We can prove the condition easily in a way analogous to the proof of the trajectory case of Lemma 5.10. In this case, we have the exact value for now in the last state of the trajectory: if the trajectory is of length δ, then the value of now increases exactly by δ. In addition, a.t does not change throughout the trajectory. Using these facts and a bound on b.pos by the **evolve** statement, we can obtain the required inequality.

□

From Lemma 4.16 and auxiliary lemmas proved in this subsection (Lemmas 5.7, 5.8, 5.9, 5.10, and 5.11), we have the following corollary.

Corollary 5.12. *The conjunction $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ forms a step invariant of $ContSATS$ using Φ.*

We use this step invariant to prove a refinement from $ContSATS$ to $ExtSATS$ in the next subsection.

5.5.3 Refinement proof

Now we prove the refinement from $ContSATS$ to $ExtSATS$. We use the mapping r defined in Section 5.5.1. Recall that r is the mapping from the states of $ContSATS$ to the states of $ExtSATS$ such that for all $s \in Q_{ContSATS}$, $r(s) = t$ such that

$$zones\_equal(s.\mathsf{zones}, t.\mathsf{zones}) \ \wedge \ s.\mathsf{nextmahf} = t.\mathsf{nextmahf} \ \wedge$$
$$zone\_equal(s.\mathsf{landing\_seq}, t.\mathsf{landing\_seq}) \ \wedge \ t.\mathsf{now} = s.\mathsf{now}.$$

This maps a state of $ContSATS$ to the state of $ExtSATS$ so that every component of the state of $ExtSATS$ matches the corresponding component of the state of $ContSATS$. Also recall that such a state $r(s)$ in $ExtSATS$ for every s in $ContSATS$ is uniquely determined since the above conditions specify all components of the automata.

To prove r is a weak refinement using a step invariant, we need three predicates: invariants P_A and P_B of $ExtSATS$ and $ContSATS$, respectively, and a step invariant P^* of A using $\lambda s.P_B(r(s))$. For P_A, we use the conjunction of all invariants that have been proved for

$ContSATS$ in this chapter. Let us call this conjunction Inv_A. For P_B, we use the conjunction of the conditions of $ExtSATS$ corresponding to Φ. Φ is first defined to express the property of $ExtSATS$ in $ContSATS$. However, since we can use the exact same expression to represent Conditions Φ in both $ExtSATS$ and $ContSATS$, we define Inv_B to be the conjunction of the conditions stated in Φ. Lastly, we will use $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ defined in the last subsection as P^*. Since $\Phi = \lambda s.Inv_B(r(s))$, this conjunction satisfies the definition of a step invariant of $ContSATS$ using $\lambda s.Inv_B(r(s))$.

Theorem 5.13. *The function r is a weak refinement from $ContSATS$ to $ExtSATS$ using Inv_A, Inv_B, and $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$.*

Proof. Condition 1: No aircraft is in any zone or the landing sequence in the initial states s_0 and t_0 of $ExtSATS$ and $ContSATS$, respectively. Thus the conditions $A_1, A_2, B, C_1,$ and C_2 hold. And the values of nextmahf and now are the same in both automata. It implies that $r(s_0) = t_0$. Condition 2 and 3: Suppose α is a step of A. We refer to $\alpha.fstate$ as s and $\alpha.lstate$ as s' in the following. It is easy to see that $s' \in dom(r)$ since r is a total function. We also assume invariants of $ContSATS$, Conditions Φ, and $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ hold in s. We first prove the case that α consists of one discrete transition (Condition 2). We have the following cases of the possible transitions.

- LowestAvailableAltitude(σ): From the precondition of the transition, there is at least one aircraft in maz(σ) in s, and thus also in $r(s)$. It follows that LowestAvailableAltitude(σ) is enabled in $r(s)$, and thus an execution fragment β of $ExtSATS$ starting with $r(s)$ that consists of one LowestAvailableAltitude(σ) surrounded by two point trajectories is a valid execution fragment of $ExtSATS$. It is easy to see $trace(\alpha) = trace(\beta)$.

  Now we prove $\beta.lstate = r(s')$. As discussed in Section 5.3.3, if holding3(σ) is empty in s, LowestAvailableAltitude(σ) in s has the exact same effects in $ContSATS$ and $ExtSATS$ with respect to r. Thus $trace(\alpha) = trace(\beta)$ and $\beta.lstate = r(s')$ hold in such a case. Thus it is sufficient to prove that holding3(σ) is empty in s. From the precondition, there is an aircraft a such that $\mathsf{a.x} = \mathsf{L_M}$, and a.line = LINE_maz($\sigma$). From Lemma 5.3 and the condition A_2 defined in Lemma 5.8, if holding3(σ) is not empty $\mathsf{a.x} = \mathsf{L_M} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_{3dec}}}{\mathsf{V_{d\_min}}})\mathsf{V_{max}} - \mathsf{L_T}$. This contradicts the assumption that $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_{3dec}}}{\mathsf{V_{d\_min}}})\mathsf{V_{max}} < \mathsf{L_T} + \mathsf{L_M}$. Thus there is no aircraft in holding3(σ).

- HoldingPatternDescend(σ): As discussed in 5.3.3, the transition in $ContSATS$ does not check if holding2(σ) is empty. To guarantee that HoldingPatternDescend(σ) is enabled in

$ExtSATS$ when it is so in $ContSATS$, holding2(σ) must be empty in $ExtSATS$ when the transition is performed.

From the precondition, there is an aircraft with the line value of LINE_holding3dec(σ) in s. It follows from the condition B defined in Lemma 5.7 that holding2(σ) is empty in that state. Thus holding2(σ) is also empty in $r(s)$, and therefore HoldingPatternDescend(σ) is enabled in that state. Let β be the execution fragment of $ExtSATS$ starting with $r(s)$ that consists of one HoldingPatternDescend(σ). It is easy to see that $trace(\alpha) = trace(\beta)$ and $\beta.lstate = r(s')$

- Landing: From the precondition of the transition, there is at least one aircraft in final in s, and thus also in $r(s)$. Consider an execution fragment β of $ExtSATS$ starting with $r(s)$ that consists of one Landing transition. It is easy to see $trace(\alpha) = trace(\beta)$. It follows from $\Phi.3$ that, in both automata, this transition removes the first aircraft from both final and landing_sequence. Thus $\beta.lstate = r(s')$ holds. Now we prove Landing is indeed enabled in $r(s)$. It is sufficient to prove that runway is empty in s. Suppose, for a contradiction, aircraft a is in runway in s. From the precondition of the transition, there is an aircraft b in final such that $\mathsf{b.x} = \mathsf{L_T}$. From the condition C_2 defined in Lemma 5.11, $\mathsf{now} - \mathsf{a.t} \geq \frac{\mathsf{L_T}-(\mathsf{L_T}-\mathsf{S_T})}{\mathsf{V_{max}}} = \frac{\mathsf{S_T}}{\mathsf{V_{max}}}$. In addition, from Lemma 5.4, $\mathsf{now} - \mathsf{a.t} \leq \mathsf{T_{Tax}}$. These two inequalities give us $\mathsf{T_{Tax}} \geq \frac{\mathsf{S_T}}{\mathsf{V_{max}}}$. This contradicts the assumption $\mathsf{T_{Tax}} < \frac{\mathsf{S_T}}{\mathsf{V_{max}}}$.

- StartHolding2, StartDescending3, StartDescending, or SetTime: Let β be the point trajectory of $r(s)$. These transitions do not affect either nextmahf, zones, landing_sequence, or now. Thus $\beta.lstate = r(s')$ holds. The condition $trace(\alpha) = trace(\beta)$ also holds since these transitions are internal.

For the rest of the transitions, the precondition of each of these transitions immediately implies the precondition of the corresponding transition of $ExtSATS$ in $r(s)$. Consider the execution fragment β of $ExtSATS$ that starts with $r(s)$ and consisting of that corresponding transition of $ExtSATS$. The condition $trace(\alpha) = trace(\beta)$ obviously follows. It is also easy to see that $\beta.lstate = r(s')$ holds from the definition of these transitions.

Now we consider the case in which α consists of one closed trajectory (Condition 3). Let β be an execution fragment of $ExtSATS$ consisting of a single closed trajectory such that the duration of β is the same as the duration of α. It is easy to see that $trace(\alpha) = trace(\beta)$. Since the values of the variable now in both $ExtSATS$ and $ContSATS$ increase by the same amount (the duration of the trajectory) by these trajectories, and the other components referred in r do not change by them, $\beta.lstate = r(s')$ holds. □

We have proved that there is a weak refinement using step invariants from $ContSATS$ to $ExtSATS$. From Corollary 4.10 and Theorems 4.12, 4.14, and 4.18, this fact implies that $traces(ContSATS) \subseteq traces(ExtSATS)$.

Furthermore, we can prove that invariants of $ExtSATS$ are also invariants of $ContSATS$ using Corollary 4.23.

Corollary 5.14. *Let P be an invariant of $ExtSATS$. Then, $\lambda s.P(r(s))$ is an invariant of $ContSATS$.*

Proof. From Theorem 5.13, there is a weak refinement from $ExtSATS$ to $ContSATS$. The required condition immediately follows from Corollary 4.23. □

Thus all conditions stated in Φ are invariants of $ContSATS$, and thus so are all of A_1, A_2, B, C_1, and C_2 since these form a step invariant using Φ, and all these conditions hold in the start state of $ContSATS$. We state this fact as a corollary as follows.

Corollary 5.15. *Conditions Φ, A_1, A_2, B, C_1, and C_2 are invariants of $ContSATS$*

In addition, from Corollary 5.14, all safe separation properties proved in Chapter 3 are invariants of $ContSATS$. From this fact, we can guarantee the following safe separation properties of $ContSATS$.

Corollary 5.16. *The following facts hold for $ContSATS$.*

1. *From Property 3 proved in Chapter 3 (the number of aircraft in each vertical fix is at most one), any two aircraft would never get close in* holding2 *and* holding3 *zones.*

2. *From Property 5 proved in Chapter 3 (the number of aircraft in a lateral entry zone is at most one), any two aircraft would never get close in* lez *zones.*

3. *From Property 6 proved in Chapter 3 (if a* lez *zone of one specific side is not empty, then* holding2 *and* holding3 *and* maz *zones of the same side are all empty), when an aircraft is in a* lez *zone, no aircraft is in the missed approach path or is hovering in the vertical fixes. (See Figure 5.2 again and observe that an aircraft in a lateral entry path represented as the line* lez *and an aircraft in a missed approach path represented as the line* maz *would get close if these aircraft were on these two path, respectively, at the same time. The guarantee stated above prevents such a scenario.)*

5.6 Spacing properties of aircraft in *ContSATS*

In the previous section, by using a refinement technique, we proved that all invariants of the discrete model of SATS that have been proved in Chapter 3 are also invariants of *ContSATS*. From this fact, we can guarantee some safe separation properties of *ContSATS*, as stated in Corollary 5.16.

The spacing properties stated in Corollary 5.16 express the safe separation of aircraft in one specific zone. However, one might be interested in the safe separation of aircraft in the areas other than those stated above, or aircraft in two consecutive zones. In addition, maz zones may contain two aircraft at the same time (recall that Property 4 proved in Chapter 3 states that at most two aircraft are in each maz zone, and there is a reachable state of the discrete model in which two aircraft in one maz zone). Thus we also want to prove a lower bound on the spacing of aircraft in maz zones. In this section, we conclude the safe separation property analysis for *ContSATS* in this thesis, by proving such spacing properties for all pairs of consecutive zones and for the maz zones in *ContSATS*.

The *spacing* between two aircraft is defined as the distance of the two aircraft with respect to the pre-determined paths of *ContSATS*:

Definition 5.17. For any two consecutive lines L_1 and L_2 on which aircraft move from L_1 to L_2 (that is, at the end point of L_1, the line value of aircraft is re-assigned to L_2), The *spacing* between aircraft a on L_1 and aircraft b on L_2, denoted by $\mathsf{S}_{(a,b)}$ is defined as follows:

$$\mathsf{S}_{(a,b)} = (L(a.\mathsf{line}) - a.\mathsf{pos}) + b.\mathsf{pos}$$

An overview of the spacing properties of aircraft in two consecutive zones that we prove in this section is depicted in Figure 5.7. Each bi-directed arrow in the picture represents a lower bound on the spacing of aircraft that we prove in this section.

The spacing of two aircraft does not always represents the Euclidean distance between the two aircraft. For example, the arrow labeled S_T in Figure 5.7 forms an “L” shape. The distance between the two aircraft that are at the end points of this arrow is determined by the length of this “L” shape.

5.6.1 S_T: the spacing of aircraft in the approach area

We first prove a lower bound on the spacing of two aircraft in the approach area. Though the approach area consists of four different zones, we prove a lower bound on the spacing between two aircraft in any (not necessarily consecutive) zones in the approach area using the D function

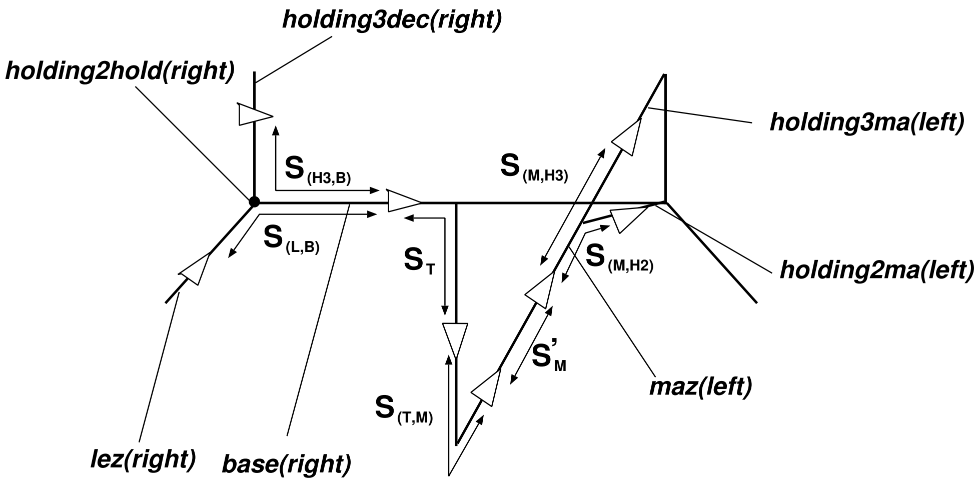


Figure 5.7: Lower bounds on the spacing of aircraft in two consecutive zones in $ContSATS$

of aircraft. Recall that the value of D function for aircraft a (D(a)) represents the distance a has flown in the approach area, and then in maz:

$$\mathsf{D}(a) = \begin{cases} a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_base}(\sigma) \text{ for some side } \sigma \\ \mathsf{L_B} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_intermediate} \\ \mathsf{L_B} + \mathsf{L_I} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_final} \\ \mathsf{L_B} + \mathsf{L_I} + \mathsf{L_F} + a.\mathsf{pos} & \text{if } a.\mathsf{line} = \mathsf{LINE\_maz}(\sigma) \text{ for some side } \sigma \\ 0 & \text{otherwise} \end{cases}$$

We prove an upper bound on the difference between two aircraft's D values: $|\mathsf{D}(a) - \mathsf{D}(b)|$. Here we explain what this difference $|\mathsf{D}(a) - \mathsf{D}(b)|$ represents in a different situation.

1. If two aircraft are in the same zone (line), then $|\mathsf{D}(a) - \mathsf{D}(b)|$ represents the difference of the pos values of two.

2. If the pair of the lines two aircraft are respectively on is either (base(right), intermediate), (base(left), intermediate), or (intermediate, final), then $|\mathsf{D}(a) - \mathsf{D}(b)|$ represents the spacing of two aircraft $\mathsf{S}_{(a,b)}$ as defined is Definition 5.17. If the pair is (base(σ), final) for some side σ, then $|\mathsf{D}(a) - \mathsf{D}(b)| = (L(a.\mathsf{line}) - a.\mathsf{pos}) + b.\mathsf{pos} + \mathsf{L_I}$. This is the spacing of aircraft a on base(σ) and b on final when we consider two zones final and intermediate as one larger zone.

3. The only subtle point is what $|\mathsf{D}(a) - \mathsf{D}(b)|$ represents when one aircraft is in base(right), and the other is in base(left) (Figure Figure 5.8). In this case, we have two aircraft merging to the center of the approach area, instead of one catching up with the other (like the situations we have seen in Cases 1 and 2). In this case, the Euclidean distance between two aircraft is actually larger than or equal to $|\mathsf{D}(a) - \mathsf{D}(b)|$ from the following

reason: Suppose, without loss of generality, $\mathsf{D}(a) \geq \mathsf{D}(b)$. The Euclidean distance between a and b is $2\mathsf{L_B} - (\mathsf{D}(a) + \mathsf{D}(b))$. Now the difference between the Euclidean distance and $|\mathsf{D}(a) - \mathsf{D}(b)|$ is $(2\mathsf{L_B} - (\mathsf{D}(a) + \mathsf{D}(b))) - (\mathsf{D}(a) - \mathsf{D}(b)) = 2\mathsf{L_B} - 2\mathsf{D}(a) = 2\mathsf{L_B} - 2(a.\mathsf{pos})$. From Lemma 5.3, $\mathsf{L_B} \geq a.\mathsf{pos}$. Thus the Euclidean distance between a and b is larger than or equal to $|\mathsf{D}(a) - \mathsf{D}(b)|$. Thus, a bound on $|\mathsf{D}(a) - \mathsf{D}(b)|$ implies some meaningful spacing bound of aircraft for this situation as well.

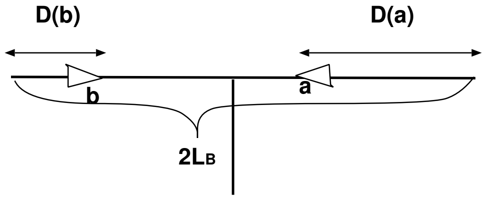


Figure 5.8: Two aircraft are respectively in the different sides of the base zones.

From the above discussion, a lower bound for $|\mathsf{D}(a) - \mathsf{D}(b)|$ of two aircraft a and b in the approach area express a meaningful safe separation of aircraft. Thus, we prove a lower bound for this value as the following theorem.

Theorem 5.18. *For any reachable state s of $ContSATS$ and aircraft a and b in the approach area, the following condition holds.*

$$|\mathsf{D}(a) - \mathsf{D}(b)| \geq \mathsf{S_T}$$

Proof. To prove this bound, we use Condition C_1 that are used to the refinement proof in Section 5.5. Recall that C_1 is an invariant of $ContSATS$ as stated in Corollary 5.15. As discussed in Section 5.5.2, Condition C_1 states a generalized lower bound on the spacing in the approach area than the one model-checked in [12]. Indeed, if we substitute the largest possible value $\mathsf{L_T}$ of $\mathsf{D}(\text{leader}(a, \mathsf{landing\_seq}))$ obtained from Lemma 5.3, then we have the same bound of $\mathsf{S_T} = \mathsf{S_0} - \frac{\mathsf{L_T} - \mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$ as model-checked in [12]. Note, however, that, rigorously speaking, the condition C_1 states the separation between a specific aircraft and its immediate leader in term of the D values when both are in the approach area. Nevertheless, we can guarantee that this lower bound holds for any two aircraft in the approach area, since aircraft follow the order in the landing sequence when initiating the approach (the leader of an aircraft initiates the approach before that aircraft does so), and each pair of an aircraft and its leader maintain this spacing property of $\mathsf{S_T}$ in terms of the D values. □

5.6.2 $\mathsf{S}_{(\mathsf{H3,B})}$ and $\mathsf{S}_{(\mathsf{L,B})}$: the spacing between aircraft in the initiation zones (holding3 and lez) and aircraft in the base zones

First we prove a lower bound on the spacing between aircraft in holding3(σ) and aircraft in holding2(σ), as follows:

Theorem 5.19. *For any reachable state of ContSATS, aircraft a in* holding3*(σ), and aircraft b in* holding2*(σ),* $\mathsf{S}_{(a,b)} \geq \mathsf{L}_3$.

Proof. From Condition B (defined in Lemma 5.7) that is proved to be an invariant of $ContSATS$ by Corollary 5.15, no aircraft is descending from `holding3`(σ) to `holding2`(σ) (no aircraft is on LINE_holding3dec(σ)) when aircraft b is in `holding2(`σ`)`. This implies that the spacing between two aircraft in `holding3` and in `holding2`, respectively, is at least L_3, the length of the descending path from the hovering point `LINE_holding3hold`(σ) to the hovering point `LINE_holding2hold`(σ). □

We have also obtained a safe separation property for aircraft in lez(σ) and aircraft in holding2(σ), as Fact 4 in Corollary 5.16: no two aircraft are in lez(σ) and holding2(σ), respectively, at the same time.

However, LINE_holding2hold(σ) – the part of holding2(σ) that connects `holding3`(σ), `lez`(σ), and base(σ) – is not really a line, but a holding point (a line with length 0). Thus, it is reasonable to consider a lower bound on the spacing for aircraft in these three zones `holding3`(σ), `lez`(σ), and base(σ). From Property 6 proved in Chapter 3, we can guarantee that aircraft are not in lez(σ) and in holding3(σ) at the same time. We also want to have a lower bound on the spacing between aircraft a in either holding3(σ) or lez(σ) and aircraft b in base(σ) (see the arrows labeled by $\mathsf{S}_{(\mathsf{H3,B})}$ and $\mathsf{S}_{(\mathsf{L,B})}$ in Figure 5.7).

We first prove the case in which aircraft a is in holding3(σ).

Theorem 5.20. *For any reachable state of ContSATS, and aircraft a in* holding3*(σ) and aircraft b in* base*(σ),* $\mathsf{S}_{(a,b)} \geq S_{(H3,B)}$*, where*

$$\mathsf{S}_{(\mathsf{H3,B})} = \mathsf{L}_{\mathsf{3dec}} - \frac{\mathsf{L}_{\mathsf{3dec}}}{\mathsf{V}_{\mathsf{maz}}}(\mathsf{V}_{\mathsf{max}} - \mathsf{V}_{\mathsf{min}}).$$

Proof. We obtain lower bounds for the spacing between aircraft a in holding3(σ) and aircraft b in base(σ) by the following operational argument: Informally, we are looking at the situation where aircraft a is "catching up with" aircraft b. From Theorem 5.19, aircraft a cannot start moving on the line LINE_holding3dec(σ) when b is in LINE_holding2hold(σ). Thus it is only after b starts moving on LINE_base(σ) that a starts "catching up with" aircraft b. The worst case occurs when a moves at its maximum speed $\mathsf{V}_{\mathsf{max}}$, and b moves at its minimum speed

$\mathsf{V_{min}}$. In this case, the spacing of a and b is reduced by $\frac{\mathsf{L_{3dec}}}{\mathsf{V_{maz}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$ when a reaches the end point of LINE_holding3dec(σ). Thus, a lower bound on the spacing between aircraft in LINE_holding3dec(σ) and LINE_base(σ) is $\mathsf{S_{(H3,B)}} = \mathsf{L_{3dec}} - \frac{\mathsf{L_{3dec}}}{\mathsf{V_{maz}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$ □

The above arguments are informal operational arguments. However, we can easily formalize these arguments as invariant proofs in a way similar to step invariants proved in 5.5.

Now we prove a lower bound on the spacing between aircraft a in lez(σ) and aircraft b in base(σ).

Theorem 5.21. *For any reachable state of ContSATS, and aircraft a in* lez*(σ) and aircraft b in* base*(σ), $\mathsf{S}_{(a,b)} \geq S_{(L,B)}$, where*

$$\mathsf{S_{(L,B)}} = \mathsf{L_l} - \frac{\mathsf{L_l}}{\mathsf{V_{maz}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$$

Proof. We can prove this lower bound by an analogous argument for Theorem 5.20, using the fact (which follows from Fact 4 in Corollary 5.16) that if b is in holding2(σ), no aircraft is in lez(σ). □

5.6.3 $\mathsf{S_{(M,H3)}}$: the spacing of aircraft in a missed approach path, part 1

In this subsection, we prove a lower bound on the spacing between aircraft a on LINE_maz(σ) and aircraft b on LINE_holding3ma(σ).

Theorem 5.22. *For any reachable state of ContSATS, aircraft a on* LINE_maz(σ) *and aircraft b on* LINE_holding3ma(σ)*, $\mathsf{S}_{(a,b)} \geq \mathsf{S_{(M,H3)}}$, where*

$$\mathsf{S_{(M,H3)}} = \mathsf{L_M} + \mathsf{L_T} - \mathsf{L_{3ma}}\Delta.$$

Proof. We can easily derive this lower bound using Condition A_2 defined in Lemma 5.8, and proved to be an invariant of *ContSATS* in the end of Section 5.5. From Conclusion (1) of A_2, (i) $\mathsf{D}(a) \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$ holds for aircraft a and b we are considering. Since $\mathsf{D}(a) = a.\mathsf{pos} + \mathsf{L_T}$ for a on LINE_maz(σ), the above inequality (i) is equivalent to (ii) $a.\mathsf{pos} \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}} - \mathsf{L_T}$. Thus,

$$\begin{aligned}
\mathsf{S}_{(a,b)} &= (\mathsf{L_M} - a.\mathsf{pos}) + b.\mathsf{pos} \\
&\geq (\mathsf{L_M} - (\frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}} - \mathsf{L_T})) + b.\mathsf{pos} \quad \text{(from inequality (ii))} \\
&= \mathsf{L_M} + \mathsf{L_T} - \frac{b.\mathsf{pos}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}}) \\
&\geq \mathsf{L_M} + \mathsf{L_T} - \frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}}) \quad \text{(from Lemma 5.3)}
\end{aligned}$$

Thus we obtained the required lower bound $\mathsf{S_{(M,H3)}}$.

□

5.6.4 S'_{M} and $\mathsf{S}_{(\mathsf{T},\mathsf{M})}$: the spacing of aircraft in a missed approach path, part 2

In [12], a lower bound on the spacing of aircraft in maz zones is stated and model-checked. The authors obtained the following lower bound S_{M} on the spacing as the minimum of two spacing bounds:

$$\mathsf{S}_{\mathsf{M}} = \min(\mathsf{L}_{\mathsf{T}} - \mathsf{L}_{\mathsf{M}}\Delta,\ 2\mathsf{S}_0 - (\mathsf{L}_{\mathsf{T}} + \mathsf{L}_{\mathsf{M}} - \mathsf{S}_0)\Delta).$$

We prove a stronger bound S'_{M} in this subsection using an invariant-proof technique. The bound is stronger in that the spacing we will obtain is actually the second term ($\mathsf{S}'_{\mathsf{M}} = 2\mathsf{S}_0 - (\mathsf{L}_{\mathsf{T}} + \mathsf{L}_{\mathsf{M}} - \mathsf{S}_0)$ in the minimum in S_{M} stated above. The proof is done in a way analogous to the auxiliary lemmas that formed a step invariant in Section 5.5. In this case, however, since we have obtained a refinement from $ContSATS$ to $ExtSATS$, we can use invariants of $ExtSATS$ as invariants of $ContSATS$, and thus can use an ordinary-style invariant proof, as opposed to a step invariant.

To prove the spacing property for maz zones, we need a lower bound on the spacing between two aircraft that are respectively in the approach area and in the missed approach zone. In Condition C_1 of Lemma 5.10, we specified an lower bound on the spacing of an aircraft and its leader in the approach area. We can state the new lower bound analogously as follows.

Lemma 5.23. *Let s be a reachable state of $ContSATS$, a the first aircraft in the landing sequence that is on the approach, b the last aircraft of* maz*(σ) for some σ, that is, $b = \mathsf{maz}(\sigma)[|\mathsf{maz}(\sigma)| - 1]$. Then the following holds.*

$$\mathsf{D}(b) - \mathsf{D}(a) \geq \mathsf{S}_0 - \frac{\mathsf{D}(b) - \mathsf{S}_0}{\mathsf{V}_{\mathsf{min}}}(\mathsf{V}_{\mathsf{max}} - \mathsf{V}_{\mathsf{min}}).$$

Note that the inequality is almost identical to the one stated in the condition C_1 of Lemma 5.10. The only difference is that the leader aircraft in C_1 is replaced by the last aircraft in the missed approach zone.

Proof. The proof can be done in a way analogous to Lemma 5.10. In the initial state, there is no aircraft in the logical zones. Thus the condition holds. Now we consider the induction step. Suppose the condition holds in a reachable state s of $ContSATS$. Let α be a step of $ContSATS$ starting with s. We denote $\alpha.lstate$ by s'.

First we consider the case α consists of a single discrete transition surrounded by two point trajectories. We prove the condition for aircraft that are affected by the transition, since the condition for the rest of the aircraft immediately follows from the induction hypothesis.

- VerticalApproachInitiation(σ): From the assumption of the condition, we assume that the aircraft that initiates the approach by the transition is the first aircraft of the landing sequence. Since b is in the missed approach area, we have $\mathsf{D}(b) \geq \mathsf{L_T}$. We also have an assumption on $\mathsf{S_0}$: $\mathsf{S_0} < \mathsf{L_T}$. Hence we have $\mathsf{D}(b) > \mathsf{S_0} > \mathsf{S_0} - \frac{\mathsf{D}(b)-\mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$. Considering that the value of $\mathsf{D}(a)$ is 0 in s', the above inequality is exactly what we need.

- LateralApproachInitiation(σ): If the aircraft moves to the approach area, we can use the same discussion to prove the condition as in the case of VerticalApproachInitiation. If the aircraft moves to holding2, the condition immediately follows from the induction hypothesis, since the transition does not affect the approach area or the maz zones.

- Exit and Landing: The second aircraft of the landing sequence becomes the first aircraft by the transition. We have the required spacing before the transition between the first aircraft and the last aircraft in a maz zone. In addition, before the transition, from the condition C_1 (recall C_1 is now an invariant of $ContSATS$), the first aircraft and the second aircraft maintain a minimum spacing stated in the condition. Thus the spacing between the first aircraft of the landing sequence in s' (which was the second aircraft in s) and the last aircraft in a maz zone clearly satisfies the spacing stated in the lemma.

- MissedApproach: the first aircraft of the landing sequence whose mahf assignment is σ becomes the last aircraft of maz(σ), and the second aircraft of the landing sequence becomes the first aircraft. The spacing between these two aircraft immediately follows from the lower bound on the spacing stated in the condition C_1 in Lemma 5.10. The spacing between the second aircraft of the landing sequence and the last aircraft of the maz zone of the opposite side of σ follows from the same discussion as in the case of Exit and Landing.

- LowestAvailableAltitude(σ): From Property 4 proved in Chapter 3, the number of aircraft in one maz zone is at most two. If there are two aircraft in maz(σ), then the condition immediately holds since the last aircraft does not change (thought it also becomes the first aircraft in that zone). If there is one aircraft in the zone, then there is no aircraft after the transition. Thus the condition holds.

- Merging and FinalSegment These transitions change the line and pos values of one aircraft on the approach, say a, but it is easy to see that from the definition of the function D, these changes does not affect the value of $\mathsf{D}(a)$. The conditions for the rest of the aircraft follows from Condition C_1 that holds before the transition.

The rest of the transitions do not affect the approach area or the missed approach zones, and thus do not change the value of D of the aircraft in these areas. Thus the conditions for the rest of the aircraft follows from the induction hypothesis.

The trajectory case can be proved in the exact same way as in Lemma 5.10.

□

By using an algebraic manipulation to the inequality proved in Lemma 5.23, we have $\mathsf{D}(a) - \mathsf{D}(b) \geq \mathsf{S_0} - \frac{\mathsf{D}(a)}{\mathsf{V_{max}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$ for aircraft a on the approach and aircraft b in a maz zone. By substituting $\mathsf{L_T}$, the largest possible value of $\mathsf{D}(a)$ for a on the approach, we have the following lower bound on the spacing between aircraft a on the approach and aircraft b in a maz zone:

$$\mathsf{S_{(T,M)}} = \mathsf{S_0} - \frac{\mathsf{L_T}}{\mathsf{V_{max}}}(\mathsf{V_{max}} - \mathsf{V_{min}}).$$

The key property of *ContSATS* needed to prove an lower bound on the spacing for aircraft in one maz zone is the alternate assignment property of mahf stated as Lemma 3.11. From this property, two consecutive aircraft in the approach would go to the different maz zones if they miss the approach. Thus, a key to prove an lower bound on the spacing in the maz zones is to bound the spacing of an aircraft and its *leader of the leader* in the approach area. This observation gives us the following Lemma 5.24. Using this lemma, we prove Lemma 5.25, which states the spacing between an aircraft a with $a.\mathsf{mahf} = \sigma$ in the approach area (and thus can possible come to $\mathsf{maz}(\sigma)$ in case it misses the approach), and the last aircraft in $\mathsf{maz}(\sigma)$. Finally, using Lemma 5.25, we prove Theorem 5.26, which states a lower bound on the spacing between two aircraft in one maz zone (see Figure 5.9, in which a bi-directed arrow represents the minimum spacing that each lemma and theorem is specifying).

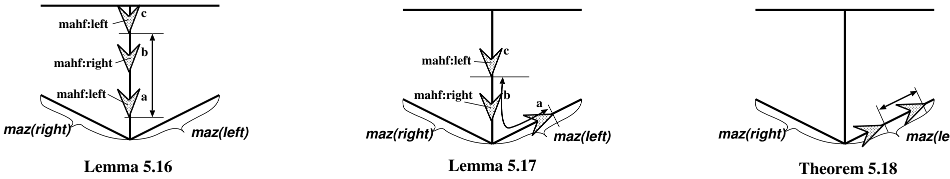


Figure 5.9: Two lemmas and one theorem that state the minimum spacing of aircraft

Lemma 5.24. *Let s be a reachable state of ContSATS, a an aircraft with $a = \mathsf{landing\_seq}[i]$ for $i > 2$, and b the leader of the leader of a in the landing sequence. Then, the following holds:*

$$\mathsf{D}(b) - \mathsf{D}(a) \geq 2\mathsf{S_0} - (\mathsf{D}(b) - \mathsf{S_0})\Delta.$$

Proof. There is no aircraft in the logical zones in the initial state. Thus the condition for that state trivially holds. Now we consider the inductive case.

Suppose the condition holds in a reachable state s of *ContSATS*. Let α be a step of *ContSATS* starting with s. We denote $\alpha.lstate$ by s'.

First we consider the case α consists of a single discrete transition surrounded by two point trajectories. We prove the condition for aircraft that are affected by the transition, since the condition for the rest of the aircraft immediately follows from the induction hypothesis.

- $\mathsf{VerticalApproachInitiation}$: The precondition of the transition guarantees that the spacing between the aircraft a that initiates the approach and its leader b is at least S_0. It implies that $\mathsf{D}(b) - \mathsf{D}(a) \geq \mathsf{S}_0$. We want to bound the spacing between a and b's leader c. From the condition C_1 of Lemma 5.10, it follows that $\mathsf{D}(c) - \mathsf{D}(b) \geq \mathsf{S}_0 - (\mathsf{D}(c) - \mathsf{S}_0)\Delta$ By summing up these two inequality side by side, we have the required bound.

- $\mathsf{LateralApproachInitiation}$: If the aircraft moves to the approach area, we can use the same discussion to prove the condition as in the case of $\mathsf{VerticalApproachInitiation}$. If the aircraft moves to $\mathsf{holding2}$, the condition immediately follows from the induction hypothesis, since the transition does not affect the approach area.

- Exit, $\mathsf{Landing}$, and $\mathsf{MissedApproach}$: These transitions remove the first aircraft in the landing sequence, and thus the first in final from the approach area. The condition trivially holds for the removed aircraft since it is no longer in the approach area.

- $\mathsf{Merging}$ and $\mathsf{FinalSegment}$ These transitions change the line and pos values of one aircraft on the approach, say a, but it is easy to see that from the definition of the function D, these changes does not affect the value of $\mathsf{D}(a)$. The conditions for the rest of the aircraft follows from Condition C_1 that holds before the transition.

The rest of the transitions do not affect the approach area or the missed approach zones, and thus do not change the value of D of the aircraft in these areas. Thus the conditions for the rest of the aircraft follows from the induction hypothesis.

The trajectory case can be proved easily by using bounds of the velocity of the aircraft specified in the **evolve** statement in the same way as Lemma 5.10. □

The following lemma states the minimum spacing between an aircraft assigned σ in the approach area, and an aircraft in $\mathsf{maz}(\sigma)$. Note also that the specified inequality is the exact same inequality as in Lemma 5.24, but for a different assumption for aircraft a and b.

Lemma 5.25. *Let s be reachable state s of $ContSATS$, a an aircraft on the approach with $a.\mathsf{mahf} = \sigma$, and b an aircraft in $\mathsf{maz}(\sigma)$. Then, the following holds:*

$$\mathsf{D}(b) - \mathsf{D}(a) \geq 2\mathsf{S}_0 - (\mathsf{D}(b) - \mathsf{S}_0)\Delta.$$

Proof. The proof can be done in analogously to Lemma 5.24. The initial case is trivial since there is no aircraft in the logical zones.

Now we consider the inductive case. Suppose the condition holds in a reachable state s of $ContSATS$. Let α be a step of $ContSATS$ starting with s. We denote $\alpha.lstate$ by s'.

First we consider the case α consists of a single discrete transition surrounded by two point trajectories. Again, we prove the condition for aircraft that are affected by the transition, since the condition for the rest of the aircraft immediately follows from the induction hypothesis.

- VerticalApproachInitiation: Let a be the aircraft that initiates the approach by the transition, and suppose a's mahf is σ. Let b be the aircraft in $\mathsf{maz}(\sigma)$. From Lemma 3.26 proved in Chapter 3, if there is an aircraft assigned σ in the approach area, then there is at most one aircraft in $\mathsf{maz}(\sigma)$. Thus b is the only aircraft in $\mathsf{maz}(\sigma)$.

  If a is not the first aircraft of the landing sequence at the moment it initiates the approach, it is the second aircraft in the sequence because of the following reason. From Case 2 of Lemma 3.26, if there is exactly one aircraft in $\mathsf{maz}(\sigma)$, there is at most one aircraft assigned σ in the approach area. Considering the alternate assignment property of mahf's of aircraft, if a is positioned at third or larger in the landing sequence, there must be at least two aircraft assigned σ in the approach area, which is a contradiction.

  If a is the second aircraft of the landing sequence, then from the precondition of the transition, the spacing between a and the first aircraft c is at least S_0. Thus $\mathsf{D}(c) - \mathsf{D}(a) \geq \mathsf{S_0}$ holds. In addition, by Lemma 5.23, $\mathsf{D}(b) - \mathsf{D}(c) \geq \mathsf{S_0} - (\mathsf{D}(b) - \mathsf{S_0})\Delta$. By summing up these two inequality side by side, we have the required inequality.

  If a is the first aircraft in the landing sequence, it implies that its leader aircraft c has already landed or missed the approach. Considering the alternate assignment property of mahf's of aircraft, c was assigned the opposite side of σ as its mahf when it was on the approach. It follows that c and b are two different aircraft. Considering that c was a's immediate leader, b had already been in $\mathsf{maz}(\sigma)$ when c reached the landing point (that is, when $\mathsf{D}(c) = \mathsf{L_T}$). In this moment, by Lemma 5.23, $\mathsf{D}(b) - \mathsf{L_T} \geq \mathsf{S_0} - (\mathsf{D}(b) - \mathsf{S_0})\Delta$ holds. From this, we have (i) $\mathsf{D}(b) \geq \mathsf{S_0} + \frac{\mathsf{L_T}}{1+\Delta}$, and using this, we also have (ii) $2\mathsf{S_0} - \frac{\mathsf{L_T}}{1+\Delta}\Delta \geq 2\mathsf{S_0} - (\mathsf{D}(b) - \mathsf{S_0})\Delta$. Considering that the value of $\mathsf{D}(b)$ monotonically increases, the above stated two inequalities (i) and (ii) also hold at the time a initiates the approach. Since $\mathsf{D}(a) = 0$ after the transition, it is sufficient to show $\mathsf{S_0} + \frac{\mathsf{L_T}}{1+\Delta} \geq 2\mathsf{S_0} - \frac{\mathsf{L_T}}{1+\Delta}\Delta$ to obtain the required inequality for the lemma. This inequality is equivalent to $\mathsf{L_T} \geq \mathsf{S_0}$, and the inequality holds from our assumption $\mathsf{L_T} > \mathsf{S_0}$.

- $\mathsf{LateralApproachInitiation}$: If the aircraft moves to the approach area, we can use the same discussion to prove the condition as in the case of $\mathsf{VerticalApproachInitiation}$. If the aircraft moves to $\mathsf{holding2}$, the condition immediately follows from the induction hypothesis, since the transition does not affect the approach area.

- Exit and $\mathsf{Landing}$: These transitions removes the first aircraft of the landing sequence from the approach area. The condition trivially holds since it does not affect D value or the mahf of the remaining aircraft.

- $\mathsf{MissedApproach}$: Before the transition, the only aircraft that has the same mahf assignment as that of the first aircraft of the landing sequence is the third aircraft. This is because of the alternate assignment property of mahf's and the fact that, from Property 1 proved in Chapter 3, there is at most four aircraft in the operation area. Thus the required inequality immediately follows from Lemma 5.24.

- : $\mathsf{Merging}$ and $\mathsf{FinalSegment}$: These transitions change the line and pos values of one aircraft on the approach, say a, but it is easy to see that from the definition of the function D, these changes does not affect the value of $\mathsf{D}(a)$. The conditions for the rest of the aircraft follows from Condition C_1 that holds before the transition.

- $\mathsf{LowestAvailableAltitude}(\sigma)$: The transition removes the first aircraft of $\mathsf{maz}(\sigma)$. As we discussed in $\mathsf{VerticalApproachInitiation}$, if there is an aircraft assigned σ in the approach, there is at most one aircraft in $\mathsf{maz}(\sigma)$. Thus there is no aircraft in $\mathsf{maz}(\sigma)$ after the transition.

The rest of the transitions do not affect the approach area or the missed approach zones, and thus do not change the value of D of the aircraft in these areas. Thus the conditions for the rest of the aircraft follows from the induction hypothesis.

The trajectory case can be proved easily by using bounds of the velocity of the aircraft specified in the **evolve** statement in the same way as Lemma 5.10. □

Now we are ready to prove a lower bound on the spacing for aircraft in one maz zone.

Theorem 5.26. *Let s be a reachable state s of $ContSATS$, a and b be aircraft in $\mathsf{maz}(\sigma)$ with $\mathsf{D}(b) > \mathsf{D}(a)$. Then, the following holds:*

$$\mathsf{D}(b) - \mathsf{D}(a) \geq 2\mathsf{S}_0 - (\mathsf{D}(b) - \mathsf{S}_0)\Delta.$$

Proof. The initial case is trivial since there is no aircraft in the logical zones.

Now we consider the inductive case. Suppose the condition holds in a reachable state s of $ContSATS$. Let α be a step of $ContSATS$ starting with s. We denote $\alpha.lstate$ by s'.

First we consider the case α consists of a single discrete transition surrounded by two point trajectories. Again, we prove the condition for aircraft that are affected by the transition, since the condition for the rest of the aircraft immediately follows from the induction hypothesis.

- $\mathsf{MissedApproach}$: The transition moves the first aircraft in the approach area to the maz zone of the side the aircraft is assigned as its mahf. The required inequality immediately follows from Lemma 5.25.

- $\mathsf{LowestAvailableAltitude}$: From Property 4 proved in Chapter 3, there are at most two aircraft in $\mathsf{maz}(\sigma)$. Thus, after the transition, there is just one aircraft in that zone.

The rest of the transitions do not affect the missed approach zones, and thus do not change the value of D of the aircraft in these areas. Thus the conditions for the rest of the aircraft follows from the induction hypothesis.

The trajectory case can be proved easily by using bounds of the velocity of the aircraft specified in the **evolve** statement in the same way as Lemma 5.10. □

By substituting the largest possible value of $\mathsf{D}(b)$ when b is in a maz zone, which is $\mathsf{L_T}+\mathsf{L_M}$, in the inequality stated in Lemma 5.26, we have the following lower bound on the spacing between two aircraft in one maz zone:

Corollary 5.27. *For any reachable state of $ContSATS$, aircraft a and b in $\mathsf{maz}(\sigma)$, $\mathsf{S}_{(a,b)} \geq \mathsf{S}_{\mathsf{M}'}$, where*

$$\mathsf{S}'_\mathsf{M} = 2\mathsf{S}_0 - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S}_0)\Delta.$$

This bound is stronger than the minimum spacing model-checked in [12], which is $\mathsf{S_M} = \min(\mathsf{L_T} - \mathsf{L_M}\Delta,\ 2\mathsf{S}_0 - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S}_0)\Delta)$. To obtain this stronger bound, we used the assumption $\mathsf{S}_0 < \mathsf{L_T}$. In [12], to our best knowledge, the authors do not mention any assumption on the size of S_0. The lack of this assumption might be the reason they had to take the minimum in their spacing bound $\mathsf{S_M}$. Indeed, a model-checking for the lower bound of $2\mathsf{S}_0 - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S}_0)\Delta$ (the second spacing of the minimum of two spacing values in $\mathsf{S_M}$, which is the bound S'_M we obtained) cannot be succeeded without assumption $\mathsf{S}_0 < \mathsf{L_T}$. This is because, if S_0 can be arbitrary large, the value of $2\mathsf{S}_0 - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S}_0)\Delta$ can also be arbitrary large. They obtained the first term of the minimum in $\mathsf{S_M}$ (which is $\mathsf{L_T} - \mathsf{L_M}\Delta$; note it does not depend on S_0) by considering the scenario that, when an aircraft initiates the approach, its leader has already landed or missed the approach. This is actually the same scenario we considered in the case of the approach initiation transitions in Lemma 5.25, in which we used the assumption $\mathsf{S}_0 < \mathsf{L_T}$.

5.6.5 $\mathsf{S}_{(\mathsf{M},\mathsf{H3})}$: the spacing of aircraft in a missed approach path, part 3

In this subsection, we prove a lower bound on the spacing between aircraft a in $\mathsf{LINE\_maz}(\sigma)$ and aircraft b in $\mathsf{LINE\_holding2ma}(\sigma)$. In Section 5.6.4, we proved a lower bound on the spacing between two aircraft in one maz zone, as Theorem 5.26. The scenario we considered in Theorem 5.26 is that one aircraft a on $\mathtt{LINE\_maz}(\sigma)$ is catching up with another aircraft b on the same line. As we can see from Figure 5.7, this "catching-up" situation still continues even after b moves to $\mathsf{LINE\_holding2ma}(\sigma)$. Theorem 5.29 states an lower bound on the spacing of aircraft we are considering in this subsection. It actually states the same inequality as in Theorem 5.26, except that $\mathsf{D}(b)$ in the original inequality in Theorem 5.26 is replaced by $\mathsf{D}'(b) = \mathsf{L_T} + \mathsf{L_M} + b.\mathsf{pos}$. This is because the D function is meaningfully defined for aircraft on the approach or in a maz zone (for other aircraft, it returns 0). The new D' function can be considered as the D function extended for aircraft on a $\mathsf{LINE\_holding2ma}$ line. Indeed, the value $\mathsf{D'}(b)$ for aircraft b on a $\mathsf{LINE\_holding2ma}$ represents the distance that b has flown in the approach area, and then in a maz zone, and finally in $\mathsf{LINE\_holding2ma}$.

To prove this theorem, we need the following Lemma 5.28 that states the spacing of aircraft a on the approach and aircraft b on $\mathsf{LINE\_holding2ma}(\sigma)$. This lemma states a claim analogous to Lemma 5.25, but consider the situation when b has already flown through $\mathtt{LINE\_maz}(\sigma)$, and thus is on $\mathsf{LINE\_holding2ma}(\sigma)$.

Lemma 5.28. *Let s be a reachable state s of $ContSATS$, a be an aircraft on the approach and b be aircraft on* $\mathsf{LINE\_holding2ma}$*(σ). Then, the following inequality holds:*

$$\mathsf{D}'(b) - \mathsf{D}(a) \geq 2\mathsf{S_0} - (\mathsf{D}'(b) - \mathsf{S_0})\Delta.$$

Proof. The initial case is trivial since there is no aircraft in the logical zones.

Now we consider the inductive case. Suppose the condition holds in a reachable state s of $ContSATS$. Let α be a step of $ContSATS$ starting with s. We denote $\alpha.lstate$ by s'.

The proof for the trajectories and the transitions other than
$\mathtt{LowestAvailableAltitude(}b\mathtt{,}\sigma\mathtt{)}$ can be done analogously to Lemma 5.25.

In the case of $\mathtt{LowestAvailableAltitude(}b\mathtt{,}\sigma\mathtt{)}$: If $\mathtt{holding2}(\sigma)$ is not empty, then b goes to $\mathtt{holding3}(\sigma)$. Thus the condition follows from the induction hypothesis. If $\mathtt{holding2}(\sigma)$ is empty, b moves to $\mathtt{LINE\_holding2ma}(\sigma)$. In this case, we can easily prove the condition from Lemma 5.25 and the fact that $\mathsf{D}(b)$ in s is equal to $\mathsf{D'}(b)$ in s'. □

Theorem 5.29. *Let s be a reachable state s of $ContSATS$, a be an aircraft on* $\mathsf{LINE\_maz}$*(σ) and b be aircraft on* $\mathsf{LINE\_holding2ma}$*(σ). Then, the following holds:*

$$\mathsf{D}'(b) - \mathsf{D}(a) \geq 2\mathsf{S_0} - (\mathsf{D}'(b) - \mathsf{S_0})\Delta.$$

Proof. The initial case is trivial since there is no aircraft in the logical zones.

Now we consider the inductive case. Suppose the condition holds in a reachable state s of $ContSATS$. Let α be a step of $ContSATS$ starting with s. We denote $\alpha.lstate$ by s'.

The proof for the trajectories and the transitions other than `LowestAvailableAltitude(`b`,`σ`)` can be done analogously to Theorem 5.26.

In the case of `LowestAvailableAltitude(`b`,`σ`)`: If `holding2`(σ) is not empty, then b goes to `holding3`(σ). Thus the condition follows from the induction hypothesis. If `holding2`(σ) is empty, b moves to `LINE_holding2ma`(σ). In this case, we can easily prove the condition from Theorem 5.26 and the fact that $\mathsf{D}(b)$ in s is equal to $\mathsf{D'}(b)$ in s'. □

By an algebraic manipulation to the inequality in Theorem 5.29, we have the following inequality for aircraft a on $\mathsf{LINE\_maz}(\sigma)$ and aircraft b on $\mathsf{LINE\_holding2ma}(\sigma)$.

$$\mathsf{D}'(b) - \mathsf{D}(a) \geq (1 + \frac{\mathsf{V_{min}}}{\mathsf{V_{max}}})\mathsf{S_0} - \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{max}}}\mathsf{D}(a).$$

By substituting the maximum value $\mathsf{L_T} + \mathsf{L_M}$ of D(a) to the above inequality, we have the following lower bound $\mathsf{S_{(M,H2)}}$ on the spacing between aircraft a on $\mathsf{LINE\_maz}(\sigma)$ and aircraft b on $\mathsf{LINE\_holding2ma}(\sigma)$.

$$\mathsf{S_{(M,H2)}} = (1 + \frac{\mathsf{V_{min}}}{\mathsf{V_{max}}})\mathsf{S_0} - \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{max}}}(\mathsf{L_T} + \mathsf{L_M}).$$

Chapter 6

Conclusions and Future Work

Summary: In this thesis, 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. This model is a discrete model in that the airspace of the airport and all movements of the aircraft are discretized. Using the reconstructed model, we verified some safety separation properties of aircraft in the Self Controlled Area using invariant-proof techniques. As is often the case, we had to strengthen some properties by using extra conditions, and by proving other properties together with them. All proofs of the properties have been rigorously checked using PVS. We found that using a mechanical prover is helpful in managing a large proof for a moderately complex system such as ours.

To examine properties that involve more realistic dynamics of aircraft, such as the spacing between aircraft, we needed a more detailed modeling of the aircraft kinematics and the geometry of the airport. A continuous model, called $ContSATS$, is presented to verity such properties of the protocol. Safety properties of the model were verified using the refinement technique and the invariant-proof technique. For the refinement technique, we introduced a new technique, *a weak refinement using a step invariant*. Using this new technique, we carried over the verification results for the discrete model to $ContSATS$. On the other hand, we needed a more careful analysis to prove lower bounds on the spacing between aircraft in $ContSATS$, since lower bounds on the spacing between aircraft in the two adjacent zones cannot be directly established by safe separation properties carried over from the discrete model. Using both the safe properties carried over from the discrete model and properties specific to $ContSATS$, we proved several spacing properties of aircraft in $ContSATS$.

Evaluation: In this thesis, we used the following general approach to formally verify safety properties of the given system using the timed I/O automata framework.

1. We first model the system as a *discrete* state-transition system, by abstracting away the

details of the continuous behavior of the system. We sometimes need to express some assumption of a real system by having supplemental conditions in the preconditions of some specific transitions in the discrete model: In this thesis, we first model the SATS landing protocol based on the model presented in [2]. The Landing transition, for instance, has an explicit check for the condition if the zone in which aircraft is moving is empty. Since the discrete model of the protocol does not have a time-dependent assumption, we had to rely on these explicit checks to prove safe separation properties of the model.

2. Using the discrete model constructed in Step 1, we prove some safety properties of the model that can be expressed in the model. Since the state structure of the model is discrete, we state safety properties as bounds on various quantities in of the discrete structure, such as the length of the queue.

3. Then, we construct a new model that has a finer abstraction of a real system: It may have more continuous behavior than the discrete model developed at Step 1, by having, for example, a real time clock structure and time bounds for particular transitions to be performed using that clock. In this new model, some explicit checks in the preconditions of some transitions may be replaced by implicit guarantees that follows from time-dependent behavior of the model, such as time bounds for particular transitions to be performed. We need to prove that such guarantees indeed hold in the new model to obtain a refinement from the continuous model to the discrete model in Step 4.

4. Using the new model, we first prove that the safety properties proved for the discrete model carry over to the new model, by using the simulation relation technique or the refinement technique. To prove the conditions needed for a simulation relation or a refinement, we often need to have an invariant of the new system. Typical invariants needed at this stage are properties that state that time-dependent or continuous behavior of the system indeed replaces particular conditions that are explicitly checked by the precondition of transitions of the discrete model, but are not explicitly checked in the new model. As in our case presented in this thesis, we may need invariants of the discrete model to prove the invariants of the new model needed for a refinement. In such a case, we prove these invariants as step invariants, and use a *weak refinement using a step invariant* introduced in Section 4.2.2.

5. After we establish a simulation relation or a refinement from the new model to the discrete model, we can use invariants of the discrete model as those of the new model, by "adapting" them using a simulation relation or a refinement mapping. This claim is stated as Theorem

4.22 and Corollary 4.23.

6. Using invariants carried over from the discrete model and other auxiliary invariants proved so far, we prove the properties that can be specified by the new model, but not by the discrete model. These properties are mainly concerned with time-dependent or continuous behavior of the new model.

Of course, we can iterate the above steps until we obtain a sufficiently detailed model and its safety properties.

One characteristic of this approach is the fact that we can use invariants of the discrete model to prove even invariants of a new refined model (as step invariants; these step invariants of the new model are guaranteed to be invariants of the model after we establish a refinement or a simulation relation from the new model to the discrete model.) For invariants of a discrete model to be useful for the verification process of a new model, it is crucial to construct a discrete model and find its invariants in a way that these invariants state useful information when carried over to the refined model.

We believe that this approach is general enough to be applied to other case studies, and we discuss some possible applications of this approach in the following future work section.

Future work: One possible new application of the above discussed approach in the real-time safety critical systems domain is verification of *time-triggered bus architectures* for the safety critical systems. Recently, several distributed fault-tolerant bus architecture protocols (for example, SAFEbus [4] that has been used in Boeing 777, TTA [7] that has been used in aircraft by Honeywell, and will be used in new intelligent cars by Audi) for safety critical embedded control systems used for aircraft or automobiles have been developed. These protocols are mainly designed for "*X-by-Wire*" applications (such as steer-by-wire or brake-by-wire), in which an aircraft or an automobile is controlled electrically by computers embedded in it.

These bus protocols offer basic and reliable communication scheme by using fault-tolerant distributed algorithms. Several people have acknowledged the importance of a formal approach for such algorithms used for bus architectures, and there have been several case studies for formal verification of those algorithms (an excellent survey for an overview of formal verification for TTA is given by Rushby [14]).

These protocols use *time-triggered* approach, by which each distributed controller of the system communicates using Time-Devision Media-Access (TDMA) slots. For these systems, it is crucial to synchronize local clocks of distributed controllers. For this reason, a distributed fault-tolerant clock synchronization algorithm is used in the controllers. Even though clock synchronization algorithms used for the bus architectures have been intensively studied in the context of formal verification [13], *start-up* algorithms that establish an initial synchronization of the local clocks has not yet been studied as deeply using a formal approach. Fault-tolerant start-up algorithms for bus architectures use a real-time time-dependent behavior in an intricate manner, and to verify the correctness of the algorithm, we have to consider every possible behavior caused by a faulty process under assumed fault hypotheses. These facts make an automatic verification of the algorithm using model-checking technique infeasible. Thus, instead, for some case study [15], some discretized version of the algorithm that abstracts away the real time using the discrete time is used to model-check the correctness of the algorithm, without proving the soundness of such an abstraction (though the authors have stated an intuition behind why the abstraction is believed to be sound).

Considering their time-dependent behavior, the fact that an automated checking of properties of them is infeasible, and their importance as real industry applications, formal verification of time-triggered bus architectures, especially their start-up algorithms are appropriate for a next application of the approach presented in this thesis. Following that approach, we will try to construct two mathematical models – a discrete model and a continuous model – of the start-up algorithms, and establish a soundness of the abstraction performed for the discrete model by proving a simulation relation or a refinement from the continuous model to the discrete model. We also aim to establish new mathematical techniques and approaches in this case study.

Bibliography

[1] Myla Aicher. Tame: Pvs strategies for special purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, 29, 2001.

[2] G. Dowek, C. Muñoz, and V. Carreño. 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, March 2004.

[3] Stephen Garland. *TIOA User Guide and Reference Manual*, September 2005.

[4] Kenneth Hoyme and Kevin Driscoll. Safebus. *IEEE Aerospace and Electronic Systems Magazine*, 8, no. 3:34–39, March 1993.

[5] http://www.flexray.com. *FlexRay Requirements Specification.*

[6] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata.* Synthesis Lectures on Computer Science. Morgan Claypool Publishers, 2006.

[7] H Kopetz and G Bauer. The time-triggered architecture. *Proceedings of The IEEE*, 91; PART 1:112–126, January 2003.

[8] Hongping Lim, Dilsun Kaynar, Nancy Lynch, and Sayan Mitra. Translating timed I/O automata specifications for theorem proving in PVS. In *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 17–31, Uppsala, Sweden, September 2005.

[9] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – part II: Timing-based systems. *Information and Computation*, 128(1):1 – 25, July 1996.

[10] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.

[11] Robin Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood, Cliffs, 1989.

[12] C. Muñoz and G. Dowek. 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.

[13] Holger Pfeifer, Detlef Schwier, and Friedrich W. von Henke. Formal Verification for Time-Triggered Clock Synchronization. In Charles B. Weinstock and John Rushby (eds.), editors, *Dependable Computing for Critical Applications 7*, volume 12 of *Dependable Computing and Fault-Tolerant Systems*, pages 207–226. IEEE Computer Society, January 1999.

[14] John Rushby. An overview of formal verification for the time-triggered architecture. In W. Damm and E.-R. Olderog, editors, *FTRTFT 2002*, volume 2469 of *Lecture Notes in Computer Science*, pages 83–105, 2002.

[15] Wilfried Steiner, John Rushby, Maria Sorea, and Holger Pfeifer. Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 189–198, Florence, Italy, June 2004. IEEE Computer Society.

[16] T.Abbott, K. Jones, M. Consiglio, D. Williams, and C. Adams. 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.

Appendix A

PVS code

[common_decls.pvs]:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generated by tioa2pvs
%% Date generated: Tue Jul 18 16:02:40 EDT 2006
%% tioa2pvs version: 20060717
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

common_decls : THEORY BEGIN

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% File contents from the file: sats_include1
    ID   : TYPE = posnat
%% End of file contents from the file: sats_include1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Tuples, Enums and Unions

Side: TYPE = {
  right,
  left}

z_name: TYPE = {
  holding3L,
  holding3R,
  holding2L,
  holding2R,
  lezL,
  lezR,
  mazL,
  mazR,
  baseL,
  baseR,
  intermediate,
  final,
  runway}

Aircraft : TYPE = [#
  mahf: Side,
  id: ID #]

% User defined theories
IMPORTING queue[Aircraft]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% File contents from the file: sats_include2
  Zone : TYPE = queue
  zone_map : TYPE = [z_name -> Zone]
  % lambda expression for zones in start state
  initialZones(n: z_name):MACRO Zone = null
%% End of file contents from the file: sats_include2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% File contents from the file: sats_include3
     %% accessing a zone by side
     holding3(side:Side):z_name = IF side = left THEN holding3L ELSE holding3R ENDIF
     holding2(side:Side):z_name = IF side = left THEN holding2L ELSE holding2R ENDIF
     lez(side:Side):z_name      = IF side = left THEN lezL      ELSE lezR      ENDIF
     maz(side:Side):z_name      = IF side = left THEN mazL      ELSE mazR      ENDIF
     base(side:Side):z_name     = IF side = left THEN baseL     ELSE baseR     ENDIF

%%%% holding3 holding2 lez maz base defined in common_decls

     %% side of a zone
     side(z:z_name|
              z=holding3L OR z=holding3R OR z=holding2L OR z=holding2R OR z=lezL OR z=lezR OR
              z=mazL OR z=mazR OR z=baseL OR z=baseR): Side =
          IF z=holding3L OR z=holding2L OR z=lezL OR z=mazL OR z=baseL THEN left ELSE right ENDIF

  %% Opposite side
  opposite(side:Side) : Side =
    IF side = right THEN left
    ELSE right
    ENDIF

  %% Does an aircraft exist in the queue?
  in_queue?(a:Aircraft, q:queue): bool = member(a,q)

  %% leader of an aircraft
  leader(a: Aircraft, q:queue| a /= first(q)): RECURSIVE Aircraft =
     IF in_queue?(a,q) THEN
       IF a = first(rest(q)) THEN first(q)
                             ELSE leader(a,rest(q))
       ENDIF
     ELSE a
     ENDIF
     MEASURE length(q)

  %% Is b the leader aircraft of a ?
  leader?(a,b:Aircraft, q:queue): bool =
    b = leader(a,q)


  %% Is a precedes b in the landing sequence?
  precedes?(a,b:Aircraft, q:queue) : RECURSIVE bool =
    IF empty?(q) OR first(q) = b THEN false
    ELSIF first(q)=a THEN in_queue?(b, rest(q))
    ELSE  precedes?(a,b,rest(q))
    ENDIF
    MEASURE length(q)

  %% Number of aircraft in a zone to assigned to one side
  assigned(z:Zone,side:Side): RECURSIVE nat =
    IF empty?(z) THEN 0
    ElSIF mahf(first(z)) = side THEN 1+assigned(rest(z),side)
    ELSE assigned(rest(z),side)
    ENDIF
    MEASURE length(z)

  %% Is any aircraft in zone z assigned to the mahf side ?
  assigned?(z:Zone,side:Side): bool =
    assigned(z,side) /= 0

  %% Is an aircraft in zone z ?
  on_zone?(z:Zone,a:Aircraft) :bool = in_queue?(a,z)
```

```
%% Is aircraft a on this side?
on?(side:Side, a:Aircraft, z:zone_map):bool =
              on_zone?(z(holding3(side)),a) OR
              on_zone?(z(holding2(side)),a) OR
              on_zone?(z(lez(side)),a) OR
              on_zone?(z(maz(side)),a)


%% Is aircraft a on the approach ?
on_approach?(z:zone_map,a:Aircraft): bool =
  on_zone?(z(baseR),a) or on_zone?(z(baseL),a) or
  on_zone?(z(intermediate),a) or on_zone?(z(final),a)


%% Is aircraft a on any zone (excluding runway)?
on_zones?(zones:zone_map,a:Aircraft): bool =
  EXISTS (z:z_name) : on_zone?(zones(z),a) AND z/=runway


%% # of aircrafts with this mahf on approach
assigned_approach(z:zone_map,side:Side): nat =
  assigned(z(baseR),side) + assigned(z(baseL),side) +
  assigned(z(intermediate),side) + assigned(z(final),side)


%% Is any aircraft on the approach assigned to the mahf side ?
on_approach?(z:zone_map,side:Side): bool =
  assigned?(z(baseR),side) or assigned?(z(baseL),side) or
  assigned?(z(intermediate),side) or assigned?(z(final),side)


%% Acutal number of aircraft at one side (excluding the approach)
actual(z:zone_map,side:Side):nat =
  length(z(holding3(side)))+length(z(holding2(side)))+length(z(lez(side)))+
  length(z(maz(side)))


%% Virtual number of aircraft at one fix
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)



%% Number of aircraft assigned to a fix
assigned2fix(z:zone_map,side:Side):nat =
  assigned(z(holding3R),side) +
  assigned(z(holding3L),side) +
  assigned(z(holding2R),side) +
  assigned(z(holding2L),side) +
  assigned(z(lezR),side) +
  assigned(z(lezL),side) +
  assigned(z(baseR),side) +
  assigned(z(baseL),side) +
  assigned(z(intermediate),side) +
  assigned(z(final),side) +
  assigned(z(mazR),side) +
  assigned(z(mazL),side)


%% Total number of simultaneous arrival operations
```

```
  arrival_op(z:zone_map):nat =
    actual(z,right) + actual(z,left) +
    length(z(baseR)) + length(z(baseL)) +
    length(z(intermediate)) + length(z(final))


  % define movement of aircrafts

  % an aircraft moves from z_from to z_to
  move(z_from, z_to: z_name, zones:zone_map| z_from /= z_to AND NOT empty?(z_from)): zone_map =
    zones WITH [(z_from) := rest(zones(z_from)),
                (z_to)   := add(zones(z_to), first(zones(z_from)))]
%% End of file contents from the file: sats_include3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

END common_decls
```

[sats_decls.pvs]:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generated by tioa2pvs
%% Date generated: Wed Jul 19 18:00:55 EDT 2006
%% tioa2pvs version: 20060717
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sats_decls : THEORY BEGIN

timed_auto_lib: LIBRARY = "../timed_auto_lib"
IMPORTING timed_auto_lib@time_thy
IMPORTING timed_auto_lib@list_rewrites
IMPORTING timed_auto_lib@bool_rewrites
IMPORTING common_decls

% State variables
states: TYPE = [#
  zones: zone_map,
  nextmahf: Side,
  landing_seq: Zone #]

% Automaton function declarations

holding3(side: Side, s: states): Zone = zones(s)(holding3(side))

holding2(side: Side, s: states): Zone = zones(s)(holding2(side))

lez(side: Side, s: states): Zone = zones(s)(lez(side))

maz(side: Side, s: states): Zone = zones(s)(maz(side))

base(side: Side, s: states): Zone = zones(s)(base(side))

intermediate(s: states): Zone = zones(s)(intermediate)

final(s: states): Zone = zones(s)(final)

runway(s: states): Zone = zones(s)(runway)

first_in_seq?(s: states, a: Aircraft): bool = a = first(landing_seq(s))

assigned_on(s: states, side: Side, mahf: Side): nat =
  assigned(zones(s)(holding3(side)), mahf)
    + assigned(zones(s)(holding2(side)), mahf)
    + assigned(zones(s)(lez(side)), mahf)
    + assigned(zones(s)(maz(side)), mahf)

assigned_on?(side: Side, mahf: Side, s: states): bool =
  assigned?(zones(s)(holding3(side)), mahf)
```

```
    OR assigned?(zones(s)(holding2(side)), mahf)
    OR assigned?(zones(s)(lez(side)), mahf)
    OR assigned?(zones(s)(maz(side)), mahf)

on?(side: Side, a: Aircraft, s: states): bool = on?(side, a, zones(s))

on_approach?(s: states, a: Aircraft): bool = on_approach?(zones(s), a)

on_zones?(s: states, a: Aircraft): bool = on_zones?(zones(s), a)

assigned_approach(s: states, side: Side): nat =
  assigned_approach(zones(s), side)

on_approach?(s: states, side: Side): bool = on_approach?(zones(s), side)

actual(s: states, side: Side): nat = actual(zones(s), side)

virtual(s: states, side: Side): nat = virtual(zones(s), side)

assigned2fix(s: states, side: Side): nat = assigned2fix(zones(s), side)

arrival_op(s: states): nat = arrival_op(zones(s))

aircraft(s: states, side: Side, id_: ID): Aircraft =
  (# mahf := IF empty?(landing_seq(s)) THEN side ELSE nextmahf(s) ENDIF,
    id := id_ #)

enter(z_enter: z_name, s: states, side: Side, id: ID, zones_: zone_map):
  zone_map =
  zones_ WITH [(z_enter) := add(zones(s)(z_enter), aircraft(s, side, id))]

reassign(s: states, a: Aircraft): Aircraft =
  a WITH
    [(mahf) :=
      IF empty?(landing_seq(s)) THEN mahf(a) ELSE nextmahf(s) ENDIF]

% Start state
start(s: states): bool = s=s WITH [
  zones := initialZones,
  nextmahf := right,
  landing_seq := empty]

% Actions signatures
actions: DATATYPE BEGIN
  VerticalEntry(ac: Aircraft, id: ID, side: Side): VerticalEntry?
  LateralEntry(ac: Aircraft, id: ID, side: Side): LateralEntry?
  HoldingPatternDescend(ac: Aircraft, side: Side): HoldingPatternDescend?
  VerticalApproachInitiation(ac: Aircraft, side: Side):
    VerticalApproachInitiation?
  LateralApproachInitiation(ac: Aircraft, side: Side):
    LateralApproachInitiation?
  Merging(ac: Aircraft, side: Side): Merging?
  Exit(ac: Aircraft): Exit?
  FinalSegment(ac: Aircraft): FinalSegment?
  Landing(ac: Aircraft): Landing?
  Taxiing(ac: Aircraft): Taxiing?
  MissedApproach(ac: Aircraft): MissedApproach?
  LowestAvailableAltitude(ac: Aircraft, side: Side):
    LowestAvailableAltitude?
END actions

% actions visibility
visible?(a:actions): bool =
  FALSE

% Enabled
enabled(a:actions, s:states):bool =
  CASES a OF
    VerticalEntry(a, id, side):
      virtual(s, side) < 2 AND NOT on_approach?(s, side)
```

```
        AND empty?(maz(side, s))
        AND empty?(lez(side, s))
        AND empty?(holding3(side, s))
        AND a = aircraft(s, side, id)
        AND
          (FORALL (a: Aircraft):
             on_zones?(s, a) OR in_queue?(a, landing_seq(s))
               OR on_zone?(runway(s), a)
               => id(a) /= id),

    LateralEntry(a, id, side):
      virtual(s, side) = 0 AND a = aircraft(s, side, id)
        AND
          (FORALL (a: Aircraft):
             on_zones?(s, a) OR in_queue?(a, landing_seq(s))
               OR on_zone?(runway(s), a)
               => id(a) /= id),

    HoldingPatternDescend(a, side):
      NOT empty?(holding3(side, s)) AND a = first(holding3(side, s))
        AND empty?(holding2(side, s)),

    VerticalApproachInitiation(a, side):
      NOT empty?(holding2(side, s)) AND a = first(holding2(side, s))
        AND length(base(opposite(side), s)) <= 1
        AND
          (first_in_seq?(s, a)
             OR on_approach?(s, leader(a, landing_seq(s)))),

    LateralApproachInitiation(a, side):
      NOT empty?(lez(side, s)) AND a = first(lez(side, s)),

    Merging(a, side):
      NOT empty?(base(side, s)) AND a = first(base(side, s))
        AND
          (first_in_seq?(s, a)
             OR on_zone?(intermediate(s), leader(a, landing_seq(s)))
             OR on_zone?(final(s), leader(a, landing_seq(s)))),

    Exit(a):
      NOT empty?(intermediate(s)) AND NOT empty?(landing_seq(s))
        AND a = first(intermediate(s))
        AND first_in_seq?(s, a),

    FinalSegment(a):
      NOT empty?(intermediate(s)) AND a = first(intermediate(s)),

    Landing(a):
      NOT empty?(final(s)) AND NOT empty?(landing_seq(s))
        AND a = first(final(s))
        AND empty?(runway(s)),

    Taxiing(a):
      NOT empty?(runway(s)) AND a = first(runway(s))
        AND first_in_seq?(s, a),

    MissedApproach(a):
      NOT empty?(final(s)) AND NOT empty?(landing_seq(s))
        AND a = first(final(s)),

    LowestAvailableAltitude(a, side):
      NOT empty?(maz(side, s)) AND a = first(maz(side, s))


  ENDCASES

%enabled (a:actions, s:states):bool = enabled_specific(a,s)

% Transition function
trans(a:actions, s:states):states =
  CASES a OF
    VerticalEntry(a, id, side):
      s WITH
        [landing_seq := add(landing_seq(s), a),

         nextmahf := opposite(mahf(a)),

         zones := enter(holding3(side), s, side, id, zones(s))],

    LateralEntry(a, id, side):
      s WITH
        [landing_seq := add(landing_seq(s), a),

         nextmahf := opposite(mahf(a)),

         zones := enter(lez(side), s, side, id, zones(s))],

    HoldingPatternDescend(a, side):
      s WITH [zones := move(holding3(side), holding2(side), zones(s))],

    VerticalApproachInitiation(a, side):
      s WITH [zones := move(holding2(side), base(side), zones(s))],

    LateralApproachInitiation(a, side):
      s WITH
        [zones :=
           IF length(base(opposite(side), s)) <= 1
                AND
                  (first_in_seq?(s, a)
                     OR on_approach?(s, leader(a, landing_seq(s))))
             THEN move(lez(side), base(side), zones(s))
           ELSE move(lez(side), holding2(side), zones(s))
           ENDIF],

    Merging(a, side):
      s WITH [zones := move(base(side), intermediate, zones(s))],

    Exit(a):
      s WITH
        [landing_seq := rest(landing_seq(s)),

         zones := zones(s) WITH [(intermediate) := rest(intermediate(s))]],

    FinalSegment(a): s WITH [zones := move(intermediate, final, zones(s))],

    Landing(a):
      s WITH
        [landing_seq := rest(landing_seq(s)),

         zones := move(final, runway, zones(s))],

    Taxiing(a):
      s WITH [zones := zones(s) WITH [(runway) := rest(runway(s))]],

    MissedApproach(a):
      s WITH
        [landing_seq := add(rest(landing_seq(s)), reassign(s, a)),

         nextmahf := opposite(mahf(reassign(s, a))),

         zones :=
           zones(s) WITH [(final) := rest(final(s))] WITH
             [(maz(mahf(a))) := add(maz(mahf(a), s), reassign(s, a))]],

    LowestAvailableAltitude(a, side):
      s WITH
        [zones :=
           IF empty?(holding3(side, s)) AND empty?(holding2(side, s))
             THEN move(maz(side), holding2(side), zones(s))
           ELSIF empty?(holding3(side, s))
```

```
             THEN move(maz(side), holding3(side), zones(s))
           ELSE
             move
               (maz(side),
                holding3(side),
                move(holding3(side), holding2(side), zones(s)))
           ENDIF]


  ENDCASES



% Import statements
IMPORTING timed_auto_lib@machine
   [states, actions, enabled, trans, start, visible? ]
visible(a:actions): bool = visible?(a)
reachable(s:states): bool = reachable(s)
equivalent(s,s1:states): bool = equivalent(s, s1)

END sats_decls
```

[dt_lemmas.pvs]:

```
dt_lemmas : THEORY

BEGIN

     IMPORTING sats_decls, list_props[Aircraft]

     parity_axiom: AXIOM (FORALL (i:nat): (even?(i) IFF NOT even?(i+1)))



     %% ID uniqueness for one queue
     ID_queue_uniqueness(q:queue): bool =
        (FORALL (i,j: nat):
           i < length(q) and j < length(q) and i<j IMPLIES
             nth(q,i)`id /= nth(q,j)`id)

     %% uniqueness for one queue
     queue_uniqueness(q:queue): bool =
        (FORALL (i,j: nat):
           i < length(q) and j < length(q) and i<j IMPLIES
             nth(q,i) /= nth(q,j))


     %% An aircraft that entered the landing_sequence (b, here) is preceded by
     %% any aircraft that is already in the sequence (a).
     dt1: LEMMA (Forall (a,b:Aircraft, q:queue):
                  in_queue?(a,q) AND NOT in_queue?(b,q) IMPLIES precedes?(a,b, add(q,b)))

     dt2: LEMMA (FORALL (a:Aircraft, q:queue):
                  in_queue?(a,add(q,a)))

     dt3: LEMMA (FORALL (a:Aircraft, q1,q2: queue):
                  in_queue?(a,q1) IMPLIES in_queue?(a,append(q1,q2)))

     dt4: LEMMA (FORALL (a:Aircraft, q1,q2: queue):
                  in_queue?(a,append(q1,q2)) AND NOT in_queue?(a, q2) IMPLIES
                  in_queue?(a,q1))

     dt4_2: LEMMA (FORALL (a,b:Aircraft, q: queue):
                  in_queue?(a,add(q,b)) AND a/=b IMPLIES
                  in_queue?(a,q))
```

```
dt5: LEMMA (FORALL (a:Aircraft, q:queue):
             length(add(q,a)) = length(q) + 1)

dt6: LEMMA (FORALL (a:Aircraft, i:nat, q:queue):
               NOT empty?(q) AND i<length(q) IMPLIES nth(add(q,a),i) = nth(q,i))

dt7: LEMMA (FORALL (a:Aircraft, q:queue):
              NOT in_queue?(a,q) IMPLIES
               (FORALL (i:nat): i < length(q) IMPLIES nth(q,i) /= a))

dt8: LEMMA (FORALL (a:Aircraft, q:queue):
               nth(add(q,a),length(q)) = a)

 dt10: LEMMA (FORALL (q:queue):
              length(q) > 0 IMPLIES length(q) = 1+length(rest(q)))

 dt11: LEMMA (FORALL (q:queue):
              length(q) >= 2 IMPLIES length(rest(q)) > 0)

 dt12: LEMMA (FORALL (q:queue,a:Aircraft):
              empty?(q) IMPLIES first(add(q,a)) = a)

 dt12_1: LEMMA (FORALL (q:queue,a:Aircraft):
              NOT empty?(q) IMPLIES first(add(q,a)) = first(q)) %% proved

 dt13: LEMMA (FORALL (q:queue,a:Aircraft):
              length(add(q,a)) > 0 AND NOT empty?(add(q,a)))

 dt14: LEMMA (FORALL (q:queue,a:Aircraft):
              in_queue?(a,q) AND a/=first(q) IMPLIES in_queue?(a,rest(q)))

 dt14_2: LEMMA (FORALL (q:queue, a,b:Aircraft):
                precedes?(a,b,q) OR precedes?(b,a,q) IMPLIES in_queue?(a,q))

 dt15_1: LEMMA (FORALL (q:queue, a,b:Aircraft):
              a/=first(q) AND b/=first(q) AND NOT empty?(q) IMPLIES
                                       precedes?(a,b,rest(q)) = precedes?(a,b,q))

 dt16: LEMMA (FORALL (q:queue, a,b:Aircraft):
              a/=b AND precedes?(a,b,q) IMPLIES NOT precedes?(b,a,q))  %% proved using dt15_1

 dt16_1: LEMMA (FORALL (q:queue,a:Aircraft):
                 NOT empty?(q) IMPLIES add(rest(q),a) = rest(add(q,a)))

 dt17: LEMMA (FORALL (q:queue, a,b,c:Aircraft):
              precedes?(a,b,add(q,c)) AND a/=c AND b/=c IMPLIES precedes?(a,b,q))

 dt18: LEMMA (FORALL (q:queue, a,b,c:Aircraft):
              precedes?(a,b,q) AND a/=c AND b/=c IMPLIES precedes?(a,b,add(q,c)))

 %% dt18 opposite direction of dt17 %%

 dt19: LEMMA (FORALL (q:queue, a,b:Aircraft):
               in_queue?(a,q) AND in_queue?(b,q) AND a/=b
                                 IMPLIES (precedes?(a,b,q) OR precedes?(b,a,q)))

 dt19_1: LEMMA (FORALL (q:queue, a:Aircraft):
                in_queue?(a,q) AND a/=first(q) IMPLIES in_queue?(leader(a,q),q))

 dt31: LEMMA (FORALL (q:queue, i:nat):
              NOT empty?(q) AND i<length(q) IMPLIES in_queue?(nth(q,i),q))

 dt20: LEMMA (FORALL (q:queue, a,b:Aircraft):
               precedes?(a,b,q) AND b/=first(q) AND queue_uniqueness(q) IMPLIES
                 precedes?(a, leader(b,q),q) OR a = leader(b,q))

 dt21: LEMMA (FORALL (q:queue, a:Aircraft): NOT precedes?(a,a,q))


 dt22: LEMMA (FORALL (q:queue, a:Aircraft):
```

```
               in_queue?(a,q) IMPLIES (EXISTS (i:nat): i<length(q) AND nth(q,i) = a))

dt23: LEMMA (FORALL (q1,q2:queue, a:Aircraft):
               (in_queue?(a,q1) OR in_queue?(a,q2)) IFF in_queue?(a,append(q1,q2)))

dt23_2: LEMMA (FORALL (q1,q2:queue):
                 empty?(q1) AND empty?(q2) IFF empty?(append(q1,q2)))

dt24: LEMMA (FORALL (q:queue, ac:Aircraft, i:nat):
               NOT empty?(q) AND
               i<length(q) AND
               ac = nth(q,i) AND
               queue_uniqueness(q) AND
               i>0
                IMPLIES
               leader(ac,q) = nth(q,i-1))

dt24_2: LEMMA (FORALL (q:queue, ac:Aircraft, i:nat):
               NOT empty?(q) AND
               i<length(q)-1 AND
               ac /= first(q) AND
               in_queue?(ac,q) AND
               leader(ac,q) = nth(q,i) AND
               queue_uniqueness(q)
                IMPLIES
               ac = nth(q,i+1))

dt25: LEMMA (FORALL (q:queue, ac1,ac2:Aircraft):
               NOT empty?(q) AND
               queue_uniqueness(q) AND
               ac1/=first(q) AND
               ac2/=first(q) AND
               in_queue?(ac1,q) AND
               in_queue?(ac2,q) AND
               leader(ac1,q) = leader(ac2,q)
                 IMPLIES
               ac1 = ac2)

dt26: LEMMA (FORALL (q1,q2:queue):
               NOT empty?(q1) IMPLIES
                first(append(q1,q2)) = first(q1))


dt27: LEMMA (FORALL (q:queue, i:nat):
              NOT empty?(rest(q)) AND NOT empty?(q) AND
              i<length(rest(q))
                 IMPLIES
                    nth(rest(q),i) = nth(q,i+1))

dt28: LEMMA (FORALL (q1,q2:queue, a:Aircraft):
               append(q1, add(q2,a)) = add(append(q1,q2),a))

dt29: LEMMA (FORALL (q1,q2:queue):
               NOT empty?(q1) IMPLIES
                 append(rest(q1),q2) = rest(append(q1,q2)))

dt30: LEMMA (FORALL (q1,q2:queue):
             NOT empty?(q2) IMPLIES
               append(add(q1, first(q2)), rest(q2)) = append(q1,q2))

dt32: LEMMA (FORALL (q:queue):
               NOT empty?(q) AND queue_uniqueness(q) IMPLIES NOT in_queue?(first(q),rest(q)))

dt33: LEMMA (FORALL (q:queue,ac:Aircraft):
               NOT empty?(q) AND ID_queue_uniqueness(q) AND ac`id = first(q)`id IMPLIES
                        NOT in_queue?(ac, rest(q)))

dt15: LEMMA (FORALL (q:queue, a,b:Aircraft):
             NOT empty?(q) AND precedes?(a,b,rest(q)) AND queue_uniqueness(q)
                 IMPLIES  precedes?(a,b,q))
```

```
 %% lemmas immediately follow from the definition
 df0: LEMMA (FORALL (side:Side, z:Zone): length(z)>=assigned(z, side))

 df1: LEMMA (FORALL (side:Side, s:states): virtual(s,side)>=assigned2fix(s,side))

 df2: LEMMA (FORALL (side:Side, s:states): virtual(s,side)>=actual(s,side))

 df3: LEMMA (FORALL (side:Side, s:states): assigned2fix(s,side) >= assigned_approach(s,side))

 df4: LEMMA (FORALL (side:Side, s:states): on_approach?(s,side) => assigned2fix(s,side) >= 1)

 df5: LEMMA (FORALL (side:Side, s:states): on_approach?(s,side) => assigned_approach(s,side) >= 1)

 df6: LEMMA (FORALL (side:Side, q:queue, ac: Aircraft):
                      (ac`mahf /= side) IMPLIES (assigned(add(q,ac),side) = assigned(q,side)))

 df7: LEMMA (FORALL (side:Side, s:states): NOT on_approach?(s,side) => assigned_approach(s,side) = 0)

 df8: LEMMA (FORALL (side:Side, q:queue, ac: Aircraft):
                      (ac`mahf = side) IMPLIES (assigned(add(q,ac),side) = assigned(q,side) + 1))

 df9: LEMMA (FORALL (side:Side, q:queue, ac: Aircraft):
                      (ac`mahf /= side) IMPLIES (assigned?(add(q,ac),side) = assigned?(q,side)))

 df10: LEMMA (FORALL (side:Side, q:queue, ac: Aircraft):
                      (assigned(add(q,ac),side) <= assigned(q,side) + 1))

 df10_1: LEMMA (FORALL (side:Side, q:queue, ac:Aircraft):
                         in_queue?(ac,q) AND ac`mahf = side IMPLIES assigned(q,side)>=1)

 df11: LEMMA (FORALL (side:Side, s:states, ac:Aircraft, mahf:Side):
                       on?(side,ac,s) AND ac`mahf = mahf IMPLIES assigned_on(s,side,mahf)>=1)

 df12: LEMMA (FORALL (side:Side, z1,z2:Zone):
                       NOT empty?(z1) IMPLIES
                        assigned(rest(z1),side) + assigned(add(z2,first(z1)),side) =
                        assigned(z1,side) + assigned(z2,side))

 df13: LEMMA (FORALL (side:Side, z:Zone, a,b :Aircraft):
            on_zone?(z,a) AND on_zone?(z,b) AND a/=b AND a`mahf = side AND b`mahf = side IMPLIES
                         assigned(z,side) >= 2)

 df14: LEMMA (FORALL (side:Side, s:states, a,b:Aircraft, mahf:Side):
            on?(side,a,s) AND on?(side,b,s) AND a`mahf = mahf AND b`mahf = mahf AND a/=b IMPLIES
                         assigned_on(s,side,mahf)>=2)


h3_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (LateralEntry?(a)               OR
          VerticalApproachInitiation?(a) OR
          LateralApproachInitiation?(a)  OR
          Merging?(a)                    OR
          Exit?(a)                       OR
          FinalSegment?(a)               OR
          Landing?(a)                    OR
          Taxiing?(a)                    OR
          MissedApproach?(a)             OR
          (LowestAvailableAltitude?(a)
           AND empty?(holding3(side(a),s))
           AND empty?(holding2(side(a),s))))
            IMPLIES
         holding3(side,trans(a,s)) = holding3(side,s))

h2_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          LateralEntry?(a)               OR
          Merging?(a)                    OR
          Exit?(a)                       OR
          FinalSegment?(a)               OR
          Landing?(a)                    OR
          Taxiing?(a)                    OR
          MissedApproach?(a)             OR
          (LowestAvailableAltitude?(a)
           AND empty?(holding3(side(a),s))
           AND NOT empty?(holding2(side(a),s))))
            IMPLIES
         holding2(side,trans(a,s)) = holding2(side,s))

maz_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          VerticalApproachInitiation?(a) OR
          LateralApproachInitiation?(a)  OR
          Merging?(a)                    OR
          Exit?(a)                       OR
          FinalSegment?(a)               OR
          Landing?(a)                    OR
          Taxiing?(a)                      )
            IMPLIES
         maz(side,trans(a,s)) = maz(side,s))

lez_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          HoldingPatternDescend?(a)      OR
          VerticalApproachInitiation?(a) OR
          Merging?(a)                    OR
          Exit?(a)                       OR
          FinalSegment?(a)               OR
          Landing?(a)                    OR
          Taxiing?(a)                    OR
          MissedApproach?(a)             OR
          LowestAvailableAltitude?(a)      )
            IMPLIES
         lez(side,trans(a,s)) = lez(side,s))

intermediate_unchanged: LEMMA
      (FORALL (s:states, a:actions):
         (VerticalEntry?(a)               OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          VerticalApproachInitiation?(a) OR
          LateralApproachInitiation?(a)  OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      )
             IMPLIES
          intermediate(trans(a,s)) = intermediate(s))

final_unchanged: LEMMA
       (FORALL (s:states, a:actions):
         (VerticalEntry?(a)               OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          VerticalApproachInitiation?(a) OR
          LateralApproachInitiation?(a)  OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      )
             IMPLIES
          final(trans(a,s)) = final(s))

opposite_side_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          VerticalApproachInitiation?(a) OR
          LateralApproachInitiation?(a)  OR
          LowestAvailableAltitude?(a)      ) AND
         side(a) = opposite(side)
            IMPLIES
         holding3(side,trans(a,s)) = holding3(side,s) AND
         holding2(side,trans(a,s)) = holding2(side,s) AND
         lez(side,trans(a,s))      = lez(side,s)      AND
         maz(side,trans(a,s))      = maz(side,s))


assigned_approach_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          Merging?(a)                    OR
          FinalSegment?(a)               OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      ) AND
          enabled(a,s)
            IMPLIES
         assigned_approach(trans(a,s),side) = assigned_approach(s,side))

on_approach?_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (VerticalEntry?(a)              OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          Merging?(a)                    OR
          FinalSegment?(a)               OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      ) AND
          enabled(a,s)
            IMPLIES
         on_approach?(trans(a,s),side) = on_approach?(s,side))

assigned_approach_inequality: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (Exit?(a)                       OR
          Landing?(a)                    OR
          MissedApproach?(a)               ) AND
          enabled(a,s)
            IMPLIES
         assigned_approach(trans(a,s),side) <= assigned_approach(s,side))

on_approach?_implication: LEMMA
      (FORALL (s:states, a:actions, side:Side):
         (Exit?(a)                       OR
          Landing?(a)                    OR
          MissedApproach?(a)               ) AND
          enabled(a,s)
            IMPLIES
         on_approach?(trans(a,s),side) => on_approach?(s,side))

on?_unchanged: LEMMA
      (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
         (HoldingPatternDescend?(a)      OR
          Merging?(a)                    OR
          Exit?(a)                       OR
          FinalSegment?(a)               OR
          Landing?(a)                    OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      ) AND
          enabled(a,s)
            IMPLIES
         on?(side, ac, trans(a,s)) = on?(side, ac, s))

on?_implication: LEMMA
```

```
      (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
         (VerticalApproachInitiation?(a)                          OR
          LateralApproachInitiation?(a)                           OR
          (ac/=ac(a) AND (VerticalEntry?(a) OR LateralEntry?(a))) OR
          (ac/=reassign(s,ac(a)) AND  MissedApproach?(a))          ) AND
          enabled(a,s) AND
          reachable(s)
            IMPLIES
         on?(side, ac, trans(a,s)) => on?(side, ac, s))

on?_implication2: LEMMA
      (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
         (VerticalEntry?(a)                             OR
          LateralEntry?(a)                              OR
          MissedApproach?(a)                            OR
          (VerticalApproachInitiation?(a) AND ac/=ac(a))   ) AND
          enabled(a,s) AND
          reachable(s)
            IMPLIES
         on?(side, ac, s) => on?(side, ac, trans(a,s)))

on?_implies_on_zones?: LEMMA
       (FORALL (s:states, side:Side, ac:Aircraft):
          on?(side, ac, s) => on_zones?(s,ac))

on_approach?_ac_unchanged: LEMMA
       (FORALL (s:states, a:actions, ac:Aircraft):
        (VerticalEntry?(a)               OR
          LateralEntry?(a)               OR
          HoldingPatternDescend?(a)      OR
          Merging?(a)                    OR
          FinalSegment?(a)               OR
          Taxiing?(a)                    OR
          LowestAvailableAltitude?(a)      ) AND
         enabled(a,s)
           IMPLIES
        on_approach?(trans(a,s),ac) = on_approach?(s,ac))


on_approach?_ac_implication: LEMMA
       (FORALL (s:states, a:actions, ac:Aircraft):
         (VerticalApproachInitiation?(a)     OR
          LateralApproachInitiation?(a)      OR
          (ac/=ac(a) AND (Exit?(a) OR Landing?(a) OR MissedApproach?(a)))) AND
         enabled(a,s) AND
         reachable(s)
            IMPLIES
        on_approach?(s,ac) => on_approach?(trans(a,s),ac))

on_approach?_ac_implication2: LEMMA
       (FORALL (s:states, a:actions, ac:Aircraft):
         ((ac/=ac(a) AND
            (VerticalApproachInitiation?(a) OR
             LateralApproachInitiation?(a)    )) OR
         Exit?(a)                                OR
         Landing?(a)                             OR
         MissedApproach?(a)                        ) AND
         enabled(a,s) AND
         reachable(s)
            IMPLIES
         on_approach?(trans(a,s),ac) => on_approach?(s,ac))

%% predicate on_zones? is preserved by movement of ac.
%% This is used in the proof of on_zones?_implication
on_zones?_preserved_by_move: LEMMA
       (FORALL (zones:zone_map, z1,z2:z_name, ac:Aircraft):
          z1/=runway AND z2/=runway AND z1/=z2 AND NOT empty?(zones(z1)) IMPLIES
            on_zones?(move(z1,z2,zones),ac) => on_zones?(zones,ac))
```

```
     on_zones?_implication: LEMMA
            (FORALL (s:states, a:actions, ac:Aircraft):
              ((ac/=ac(a) AND
                 (VerticalEntry?(a) OR
                  LateralEntry?(a)    ))       OR
               HoldingPatternDescend?(a)      OR
               VerticalApproachInitiation?(a) OR
               LateralApproachInitiation?(a)  OR
               Merging?(a)                    OR
               Exit?(a)                       OR
               FinalSegment?(a)               OR
               Landing?(a)                    OR
               Taxiing?(a)                    OR
               LowestAvailableAltitude?(a)      ) AND
              enabled(a,s) AND
              reachable(s)
                 IMPLIES
              on_zones?(trans(a,s),ac) => on_zones?(s,ac))

    %% auxiliary lemma for on?
     on?_support: LEMMA
        (FORALL (s:states, side:Side):
           (NOT empty?(holding3(side,s)) IMPLIES on?(side,first(holding3(side,s)),s)) AND
           (NOT empty?(holding2(side,s)) IMPLIES on?(side,first(holding2(side,s)),s)) AND
           (NOT empty?(maz(side,s))      IMPLIES on?(side,first(maz(side,s)),s))      AND
           (NOT empty?(maz(side,s)) AND NOT empty?(rest(maz(side,s)))
                                        IMPLIES on?(side,first(rest(maz(side,s))),s)))

    %% auxiliary lemma for on_zones?
     on_zones?_support: LEMMA
        (FORALL (s:states, side:Side):
           (NOT empty?(holding3(side,s)) IMPLIES on_zones?(s,first(holding3(side,s)))) AND
           (NOT empty?(holding2(side,s)) IMPLIES on_zones?(s,first(holding2(side,s)))) AND
           (NOT empty?(maz(side,s))      IMPLIES on_zones?(s,first(maz(side,s))))      AND
           (NOT empty?(maz(side,s)) AND NOT empty?(rest(maz(side,s)))
                                        IMPLIES on_zones?(s,first(rest(maz(side,s))))))


    %% from the definition of reachability
     reachability_lemma: LEMMA
            (FORALL (s:states, a: actions):
               reachable(s) AND enabled(a,s) => reachable(trans(a,s)))

     % What is implied by virtual=0
     virtual_0: LEMMA
            (FORALL (s:states,side:Side):
              virtual(s,side) = 0 IMPLIES
                empty?(holding3(side,s)) AND
                empty?(holding2(side,s)) AND
                empty?(lez(side,s))      AND
                empty?(maz(side,s))      AND
                NOT on_approach?(s,side) AND
                (Forall (a:Aircraft):
                  on?(opposite(side),a,s) IMPLIES mahf(a) = opposite(side)))


%% The number of aircraft on zones coincides with the number of ac in the landing sequence.
     n_of_ac_coincides_lemma: LEMMA
             (FORALL (s:states):
              reachable(s) =>
                (arrival_op(s) = length(landing_seq(s))))


%% the number of aircraft on the combined area of final and intermediate zones is
%% less than the number of aircraft in the landing sequence.
%% This lemma is used in "landing_seq_ID_uniqueness_first_final_queue_correspondence."

     final_app_area_lt_seq: LEMMA
            (FORALL (s:states):
               reachable(s) =>
```

```
                  (LET final_approach_area = append(final(s),intermediate(s)) IN
                    length(final_approach_area) <= length(landing_seq(s))))

%% uniqueness
     %% ID uniqueness for zones
     ID_uniqueness(zones:zone_map): bool =
        %% ID uniqueness on the same zone
        (FORALL (i,j: nat, z:z_name):
           i < length(zones(z)) and j < length(zones(z)) and i<j IMPLIES
             nth(zones(z),i)`id /= nth(zones(z),j)`id)
        AND
        %% ID uniqueness between different zones
        (FORALL (a,b: Aircraft, z1,z2:z_name):
           z1/=z2 AND
           (on_zone?(zones(z1),a) AND on_zone?(zones(z2),b)) IMPLIES
           a`id /= b`id)

     ID_uniqueness_preserved_by_move: LEMMA
       (FORALL (z1,z2 :z_name, zones:zone_map):
           (z1/=z2) AND NOT empty?(zones(z1)) IMPLIES
              (ID_uniqueness(zones) => ID_uniqueness(move(z1,z2,zones))))

     ID_uniqueness_lemma: LEMMA
       (FORALL (s:states):
         reachable(s) => ID_uniqueness(zones(s)))

     %% uniqueness for zones
     uniqueness(zones:zone_map): bool =
        %% uniqueness on the same zone
        (FORALL (i,j: nat, z:z_name):
           i < length(zones(z)) and j < length(zones(z)) and i<j IMPLIES
             nth(zones(z),i) /= nth(zones(z),j))
        AND
        %% uniqueness between different zones
        (FORALL (a: Aircraft, z1,z2:z_name):
           z1/=z2 IMPLIES
           (in_queue?(a,zones(z1)) IMPLIES NOT in_queue?(a,zones(z2))))

     uniqueness_lemma: LEMMA
       (FORALL (s:states):
         reachable(s) => uniqueness(zones(s)))

 %% What is implied by the uniqueness? %%%%%%%%%%%%%
     facts_from_uniqueness3 : LEMMA
        (FORALL (s:states):
          reachable(s) =>
            (FORALL (z:z_name): NOT empty?(zones(s)(z)) IMPLIES
                                       NOT in_queue?(first(zones(s)(z)), rest(zones(s)(z)))))


     landing_seq_ID_uniqueness_first_final_queue_correspondence: LEMMA
        (FORALL (s:states):
         reachable(s) =>
               (ID_queue_uniqueness(landing_seq(s)) AND
                (LET final_approach_area = append(final(s),intermediate(s)) IN
                 FORALL (i:nat):
                   i<length(landing_seq(s)) AND i<length(final_approach_area) IMPLIES
                     nth(landing_seq(s),i) = nth(final_approach_area,i)) AND
                (FORALL (a:Aircraft): on_zones?(s,a) IMPLIES in_queue?(a, landing_seq(s)))))


     %% ID uniqueness lemma for the landing sequence
     landing_seq_ID_uniqueness_lemma: LEMMA
        (FORALL (s:states):
         reachable(s) => ID_queue_uniqueness(landing_seq(s)))

     %% uniqueness lemma for the landing sequence
     landing_seq_uniqueness_lemma: LEMMA
        (FORALL (s:states):
         reachable(s) => queue_uniqueness(landing_seq(s)))
 %% What is implied by the uniqueness? %%%%%%%%%%%%%%
     facts_from_uniqueness : LEMMA
        (FORALL (s:states, side:Side, a1,a2:Aircraft):
          reachable(s) =>
            (((on?(side,a1,s) and on?(opposite(side),a2,s))   OR
              (on?(side,a1,s) and on_approach?(s, a2))        OR
              (on?(side,a1,s) and NOT empty?(final(s))        AND a2 = first(final(s)))       OR
              (on?(side,a1,s) and NOT empty?(intermediate(s)) AND a2 = first(intermediate(s)))  )
                IMPLIES  a1/=a2))

     facts_from_uniqueness2 : LEMMA
        (FORALL (s:states):
          reachable(s) =>
            NOT empty?(landing_seq(s)) IMPLIES
                  (NOT in_queue?(first(landing_seq(s)), rest(landing_seq(s)))))

     facts_from_ID_uniqueness : LEMMA
        (FORALL (s:states):
          reachable(s) =>
            (NOT empty?(landing_seq(s)) IMPLIES
             NOT in_queue?(reassign(s,first(landing_seq(s))), rest(landing_seq(s)))))

     facts_from_ID_uniqueness2 : LEMMA
        (FORALL (s:states, a,b:Aircraft):
          reachable(s) =>
            ((on_zones?(s,a) AND on_zones?(s,b) AND a/=b)  IMPLIES
                a‘id /= b‘id))


%% the first ac in the final zone is also the first in the landing_seq
     first_final_lemma: LEMMA
            (FORALL (s:states):
              reachable(s) =>
                (NOT empty?(final(s)) IMPLIES first(final(s)) = first(landing_seq(s))))


%% If an aircraft is on some zone, then it is also in the landing sequence.
     queue_correspondence_lemma: LEMMA
       (FORALL (s:states, a:Aircraft):
         reachable(s) =>
         (on_zones?(s,a) IMPLIES in_queue?(a, landing_seq(s))))


%% Auxiliary function to compute the number of aircraft assigned to one mahf in the landing seq.
     assigned_seq(seq:queue, side:Side): RECURSIVE nat =
        IF empty?(seq) THEN 0
        ELSIF mahf(first(seq)) = side THEN 1+assigned_seq(rest(seq),side)
        ELSE assigned_seq(rest(seq),side)
        ENDIF
        MEASURE length(seq)

     df_for_assigned_seq: LEMMA
           (FORALL (seq:queue,side:Side,ac:Aircraft):
              assigned_seq(add(seq,ac),side) = IF ac‘mahf = side THEN assigned_seq(seq,side) + 1
                                                                 ELSE assigned_seq(seq,side)     ENDIF)


%% The number of ac assigned to one mahf on zones coinsides with the number of it in the landing seq.
     assigned_ac_coincides_lemma: LEMMA
             (FORALL (s:states, side:Side):
              reachable(s) =>
                (assigned2fix(s,side) = assigned_seq(landing_seq(s),side)))


%% The mahf to an aircraft is alternately assigned to right and left.
     alternate_assignment_lemma: LEMMA
             (FORALL (s:states, side:Side):
              reachable(s) =>
               NOT empty?(landing_seq(s)) IMPLIES
```

```
                LET first_mahf = first(landing_seq(s))`mahf IN
                LET length_seq = length(landing_seq(s)) IN
                (assigned_seq(landing_seq(s),side) =
                        IF even?(length_seq) THEN length_seq/2
                        ELSIF first_mahf = side
                             THEN (length_seq+1)/2
                             ELSE (length_seq-1)/2  ENDIF                          ) AND
                (FORALL (i:nat): i<length(landing_seq(s)) IMPLIES
                  nth(landing_seq(s),i)`mahf =
                              IF even?(i)          THEN first_mahf
                                                   ELSE opposite(first_mahf) ENDIF ) AND
                (s`nextmahf = IF even?(length_seq) THEN first_mahf
                                                   ELSE opposite(first_mahf) ENDIF ))

%% The order of entering the approach area agrees with the order in the landing sequence..
     order_preserve_lemma: LEMMA
       (FORALL (s:states, a,b:Aircraft):
         reachable(s) =>
         (precedes?(b,a,landing_seq(s)) AND on_approach?(s,a) IMPLIES
          on_approach?(s,b)))


%% blocked_by?, blocked_opposite_side?, blocked_except_for_one?, ac_ready_to_approach?
     %% Aircraft a cannot enter the approach area since it is blocked (preceded)
     %% by aircraft b, or its mahf is assigned to the opposite side, so
     %% it will not go to maz of this side.
     blocked_by?(a,b:Aircraft, side:Side, s:states):bool =
        mahf(a) = side OR
        precedes?(b, a, landing_seq(s))

     %%
     blocked_side?(b:Aircraft, side:Side, s:states):bool =
        Forall (a:Aircraft):
           on?(side,a,s) IMPLIES blocked_by?(a,b,side,s)

     %% Every aircraft on the opposite side is blocked by aircraft b, or its mahf is
     %% opposite side.
     blocked_opposite_side?(b:Aircraft, side:Side, s:states):bool =
        blocked_side?(b, opposite(side), s)
        %Forall (a:Aircraft):
        %   on?(opposite(side),a,s) IMPLIES blocked_by?(a,b,side,s)

     %%
     blocked_side_minus_1?(b:Aircraft, side:Side, s:states):bool =
        EXISTS (c:Aircraft):
           on?(side,c,s) AND mahf(c) = opposite(side) AND
           FORALL (a:Aircraft):
             on?(side,a,s) AND a /= c IMPLIES blocked_by?(a,b,side,s)

     %% Every aircraft on the opposite side is blocked by aircraft b, or its mahf is
     %% opposite side, except for one aircraft.
     blocked_except_for_one?(b:Aircraft, side:Side, s:states):bool =
        %blocked_side_minus_1?(b, opposite(side), s)
        EXISTS (c:Aircraft):
           on?(opposite(side),c,s) AND mahf(c) = side AND
           FORALL (a:Aircraft):
             on?(opposite(side),a,s) AND a /= c IMPLIES blocked_by?(a,b,opposite(side),s)

   %% this auxiliary predicate is used in lemma 2, case 6/7.
     %% Is there an aircraft that is ready to go to the approach
     %% and goes to opposite side if it missed the approach?
     %% Here, 'ready' means that it precedes all aircrafts on the opposite side
     ac_ready_to_approach?(side:Side, s:states): bool =
           (EXISTS (a:Aircraft):
              mahf(a)=side AND
              on?(opposite(side),a,s) AND
              (FORALL (b:Aircraft):
                 on?(side,b,s) IMPLIES precedes?(a,b,landing_seq(s))))
```

```
%% action specific lemmas
     %% When NOT on_approach?(s,side) holds, an aircraft that missed appraoch goes
     %% to the opposite side of maz, thus maz of this side does not change.
     MissedApproach_going_opposite : LEMMA
           (FORALL (s:states, a:actions, side:Side):
              MissedApproach?(a) AND
              NOT on_approach?(s,side) AND
              enabled(a,s)
                IMPLIES
              maz(side,trans(a,s)) = maz(side,s))


     VerticalEntry_LateralEntry_lemma : LEMMA
           (Forall (s:states, a:actions):
              (VerticalEntry?(a) OR LateralEntry?(a)) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
              (NOT in_queue?(ac(a), landing_seq(s))) AND
              (FORALL (ac:Aircraft): on_zones?(s,ac) => ac/=ac(a)))


     HoldingPatternDescend_lemma : LEMMA
           (FORALL (s:states, a:actions):
              HoldingPatternDescend?(a) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                 (length(holding3(side(a),trans(a,s))) = length(holding3(side(a),s)) - 1) AND
                 (length(holding3(side(a),s)) <= 1 IMPLIES empty?(holding3(side(a),trans(a,s)))) AND
                 (first(holding3(side(a),s)) = first(holding2(side(a),trans(a,s)))) AND
                 (NOT empty?(holding2(side(a),trans(a,s))))                                        )

     VerticalApproachInitiation_lemma : LEMMA
           (FORALL (s:states, a:actions):
              VerticalApproachInitiation?(a) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                 length(holding2(side(a),trans(a,s))) = length(holding2(side(a),s)) - 1 AND
                 (FORALL (side:Side) : NOT on_approach?(s, side) IMPLIES
                                       assigned_approach(trans(a, s), side) <= 1) AND
                 (FORALL (side:Side) : (NOT mahf(ac(a)) = side) IMPLIES
                                        on_approach?(trans(a,s), side) = on_approach?(s, side)) AND
                 (length(holding2(side(a),s)) + length(holding3(side(a),s)) <= 1 IMPLIES
                  empty?(holding2(side(a),trans(a,s))) AND empty?(holding3(side(a),trans(a,s)))) AND
                 (((NOT (empty?(holding2(side(a), trans(a, s)))) OR
                    NOT (empty?(holding3(side(a), trans(a, s))))) AND
                   length(holding3(side(a), s)) <= 1 AND
                   length(holding2(side(a), s)) <= 1)
                    IMPLIES
                     (NOT (empty?(holding2(side(a), s))) AND
                      NOT (empty?(holding3(side(a), s))))) AND
                 (length(holding2(side(a), s)) <= 1 IMPLIES empty?(holding2(side(a), trans(a,s)))) AND
                 (FORALL (b: Aircraft): on?(opposite(side(a)),b,s) AND mahf(ac(a)) = opposite(side(a))
                   IMPLIES (blocked_except_for_one?(b,opposite(side(a)),s) =>
                            blocked_opposite_side?(b,opposite(side(a)),trans(a,s))))               AND
                 (on_approach?(trans(a,s),ac(a))                                                     )


      LateralApproachInitiation_lemma : LEMMA
           (FORALL (s:states, a:actions):
              LateralApproachInitiation?(a) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                 (FORALL (side:Side) : NOT on_approach?(s, side) IMPLIES
                                       assigned_approach(trans(a, s), side) <= 1) AND
                 (empty?(holding2(side(a),s)) AND
                  empty?(holding3(side(a),s)) AND
                  empty?(maz(side(a),s)) AND
                  length(lez(side(a),s)) <= 1
                      IMPLIES
                  ((NOT on_approach?(s,opposite(side(a))) AND
                   on_approach?(trans(a,s),opposite(side(a)))) IMPLIES
                     (FORALL (b:Aircraft):
                         (blocked_opposite_side?(b,opposite(side(a)),trans(a,s)))) AND
                     (ac_ready_to_approach?(opposite(side(a)), s))                                ) AND
                  (NOT (length(base(opposite(side(a)),s)) <= 1 AND
                       (first_in_seq?(s,ac(a)) OR on_approach?(s,leader(ac(a),landing_seq(s)))))
                    IMPLIES
                     (FORALL(b:Aircraft): blocked_except_for_one?(b, opposite(side(a)),s) IMPLIES
                                          blocked_except_for_one?(b, opposite(side(a)),trans(a,s))) AND
                     (ac_ready_to_approach?(opposite(side(a)),trans(a,s))
                         IMPLIES ac_ready_to_approach?(opposite(side(a)),s)))))


      MissedApproach_lemma : LEMMA
            (FORALL (s:states, a:actions):
              MissedApproach?(a) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                 %% MissedApproach_going_opposite
                 (FORALL (side:Side) :
                    (NOT on_approach?(s,side) IMPLIES maz(side,trans(a,s)) = maz(side,s))) AND
                 LET ma_mahf = mahf(ac(a)) IN
                 (length(maz(ma_mahf,trans(a,s))) = length(maz(ma_mahf,s)) + 1) AND
                 (maz(opposite(ma_mahf),trans(a,s)) = maz(opposite(ma_mahf),s)) AND
                 (on_approach?(trans(a,s),opposite(ma_mahf)) = on_approach?(s,opposite(ma_mahf))) AND
                 (on_approach?(s,ma_mahf)) AND
                 (assigned_approach(s, ma_mahf) <= 1 IMPLIES NOT on_approach?(trans(a,s), ma_mahf)) AND
                 (assigned_approach(trans(a,s),ma_mahf) = assigned_approach(s,ma_mahf) -1)          AND
                 (on?(ma_mahf, reassign(s,ac(a)), trans(a,s)))                                         )

      LowestAvailableAltitude_lemma: LEMMA
            (FORALL (s:states, a:actions):
              LowestAvailableAltitude?(a) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                 (IF empty?(holding3(side(a),s)) AND empty?(holding2(side(a),s)) THEN
                      length(holding2(side(a),trans(a,s))) = length(holding2(side(a),s)) + 1 AND
                      holding3(side(a),trans(a,s)) = holding3(side(a),s) AND
                      first(holding2(side(a),trans(a,s))) = first(maz(side(a),s))
                  ELSIF empty?(holding3(side(a),s)) THEN
                      holding2(side(a),trans(a,s)) = holding2(side(a),s) AND
                      length(holding3(side(a),trans(a,s))) = length(holding3(side(a),s)) + 1 AND
                      first(holding3(side(a),trans(a,s))) = first(maz(side(a),s))
                  ELSE
                      length(holding2(side(a),trans(a,s))) = length(holding2(side(a),s)) + 1 AND
                      length(holding3(side(a),trans(a,s))) = length(holding3(side(a),s)) AND
                      (empty?(holding2(side(a),s)) =>
                           first(holding2(side(a),trans(a,s))) = first(holding3(side(a),s))) AND
                      (length(holding3(side(a),s))<=1 IMPLIES
                                first(holding3(side(a),trans(a,s))) = first(maz(side(a),s)))
                  ENDIF) AND
                 (length(maz(side(a),trans(a,s))) = length(maz(side(a),s)) - 1) AND
                 (length(maz(side(a),s)) >= 1 IMPLIES
                      first(maz(side(a),trans(a,s)))=first(rest(maz(side(a),s))))   )


%% Auxiliary predicates to express that every ac on the opposite side has the opposite side as the mahf
      opposite_mahf_opposite_side?(side:Side, s:states):bool =
        Forall (a:Aircraft):
           on?(opposite(side),a,s) IMPLIES a`mahf = opposite(side)

%% Auxiliary predicates to express that every ac except for one on the opposite side has
%% the opposite side as the mahf
```

```
      opposite_mahf_except_for_one?(side:Side, s:states):bool =
        EXISTS (c:Aircraft):
           on?(opposite(side),c,s) AND mahf(c) = side AND
           FORALL (a:Aircraft):
             on?(opposite(side),a,s) AND a /= c IMPLIES a`mahf = opposite(side)

%% What is implied by assigned2fix(s,side) <= 2 ?
      assigned2fix_lemma: LEMMA
            (FORALL (s:states, side:Side):
              assigned2fix(s,side) <= 2
               IMPLIES
                ((((EXISTS (a:Aircraft): on?(side,a,s) AND mahf(a) = side) AND
                   on_approach?(s,side)) OR
                  (assigned_approach(s,side) >= 2))
                      IMPLIES
                  opposite_mahf_opposite_side?(side,s)) AND
                (on_approach?(s,side) AND ac_ready_to_approach?(side, s)
                      IMPLIES
                  opposite_mahf_except_for_one?(side,s)))

%% What is implied by virtual(s,side) < 2 ?
      virtual_lt_2_lemma: LEMMA
           (FORALL (s:states, side:Side):
              virtual(s,side) < 2
                IMPLIES
                  ((EXISTS (ac:Aircraft): on?(opposite(side),ac,s) AND mahf(ac)=side)
                     IMPLIES
                       empty?(holding2(side,s)) AND opposite_mahf_except_for_one?(side, s)) AND
                  (NOT empty?(holding2(side,s)) IMPLIES opposite_mahf_opposite_side?(side, s)))



%% Lemmas on blocked_by?, blocked_opposite_side?
     blocked_by?_unchanged: LEMMA
           (FORALL (s:states, a:actions, side:Side, ac1:Aircraft, ac2:Aircraft):
              (HoldingPatternDescend?(a)      OR
               VerticalApproachInitiation?(a) OR
               LateralApproachInitiation?(a)  OR
               Merging?(a)                    OR
               FinalSegment?(a)               OR
               Taxiing?(a)                    OR
               LowestAvailableAltitude?(a)      ) AND
              enabled(a,s)
             IMPLIES
            blocked_by?(ac1,ac2,side,trans(a,s)) = blocked_by?(ac1,ac2,side,s))

     blocked_opposite_side?_unchanged: LEMMA
           (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
              (HoldingPatternDescend?(a)      OR
               Merging?(a)                    OR
               FinalSegment?(a)               OR
               Taxiing?(a)                    OR
               LowestAvailableAltitude?(a)      ) AND
               enabled(a,s)
                 IMPLIES
              blocked_opposite_side?(ac,side,trans(a,s)) =
              blocked_opposite_side?(ac,side,s))

     blocked_opposite_side?_implication: LEMMA
           (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
              (VerticalEntry?(a)                      OR
               LateralEntry?(a)                       OR
               VerticalApproachInitiation?(a)         OR
               LateralApproachInitiation?(a)          OR
               ((Exit?(a) OR Landing?(a) OR MissedApproach?(a)) AND
                (on?(side,ac,s)))) AND
               on_zones?(s,ac) AND
               enabled(a,s) AND
               reachable(s)
                 IMPLIES
```

```
          (blocked_opposite_side?(ac,side,s) =>
           blocked_opposite_side?(ac,side,trans(a,s))))


  %% The following two lemmas state why we focus on blocked_opposite_side?
  blocked_assigned_approach : LEMMA
        (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
           (VerticalApproachInitiation?(a) OR
            LateralApproachInitiation?(a)    ) AND
           side(a) = opposite(side) AND
           blocked_opposite_side?(ac,side,s) AND
           enabled(a,s) AND
           reachable(s) AND
           on?(side,ac,s)
            IMPLIES
           assigned_approach(trans(a,s),side) = assigned_approach(s,side))

  blocked_on_approach : LEMMA
        (FORALL (s:states, a:actions, side:Side, ac:Aircraft):
           (VerticalApproachInitiation?(a) OR
            LateralApproachInitiation?(a)    ) AND
           side(a) = opposite(side) AND
           blocked_opposite_side?(ac,side,s) AND
           enabled(a,s) AND
           reachable(s) AND
           on?(side,ac,s)
            IMPLIES
           on_approach?(trans(a,s),side) = on_approach?(s,side))


%% Lemmas on ac_ready_to_approach? and blocked_except_for_one?
  ac_ready_to_approach?_unchanged: LEMMA
        (FORALL (s:states, a:actions, side:Side):
           (HoldingPatternDescend?(a)      OR
            Merging?(a)                    OR
            FinalSegment?(a)               OR
            Taxiing?(a)                    OR
            LowestAvailableAltitude?(a)      ) AND
           enabled(a,s) AND
           reachable(s)
              IMPLIES
           ac_ready_to_approach?(side,trans(a,s)) =
           ac_ready_to_approach?(side,s)           )

  ac_ready_to_approach?_implication: LEMMA
        (FORALL (s:states, a:actions, side:Side):
           (Exit?(a)                                    OR
            Landing?(a)                                 OR
            (MissedApproach?(a) AND mahf(ac(a)) = side) OR
            ((((VerticalEntry?(a) OR LateralEntry?(a)) AND
                side(a)=opposite(side)) OR
              (MissedApproach?(a) AND mahf(ac(a)) = opposite(side))) AND
             EXISTS (ac:Aircraft): on?(side,ac,s))      OR
            (VerticalApproachInitiation?(a) AND
             side(a) = opposite(side))                   ) AND
           enabled(a,s) AND
           reachable(s)
             IMPLIES
              (ac_ready_to_approach?(side,trans(a,s)) => ac_ready_to_approach?(side,s)))

  blocked_except_for_one?_unchanged: LEMMA
        (FORALL (s:states, a:actions, side:Side, b:Aircraft):
           (HoldingPatternDescend?(a)      OR
            Merging?(a)                    OR
            FinalSegment?(a)               OR
            Taxiing?(a)                    OR
            LowestAvailableAltitude?(a)      ) AND
           enabled(a,s)
              IMPLIES
           blocked_except_for_one?(b,side,trans(a,s)) =
              blocked_except_for_one?(b,side,s)           )


     blocked_except_for_one?_implication: LEMMA
           (FORALL (s:states, a:actions, side:Side, b:Aircraft):
              ((Exit?(a)                             OR
                Landing?(a)                          OR
                (MissedApproach?(a) AND
                 mahf(ac(a)) = opposite(side))       OR
                (VerticalApproachInitiation?(a) AND
                 mahf(ac(a)) = opposite(side)   AND
                 side(a) = opposite(side)          ) OR
                ((VerticalEntry?(a) OR LateralEntry?(a)) AND
                   side(a)=opposite(side))           )AND
               on?(side,b,s)                          AND
               enabled(a,s)                           AND
               reachable(s)                              )
                 IMPLIES
              blocked_except_for_one?(b,side,s) =>
              blocked_except_for_one?(b,side,trans(a,s)))

     blocked_implies_not_ready : LEMMA
           (FORALL (s:states, a:actions, side:Side, b:Aircraft):
              blocked_opposite_side?(b, side, s) AND
              on?(side, b, s) AND
              reachable(s)
                IMPLIES
              NOT ac_ready_to_approach?(side,s))


      VAI_ac_ready: LEMMA
            (FORALL (s:states, a:actions):
              VerticalApproachInitiation?(a) AND
              mahf(ac(a)) = opposite(side(a)) AND
              enabled(a,s) AND
              reachable(s)
               IMPLIES
                ac_ready_to_approach?(opposite(side(a)), s))

END dt_lemmas
```

[sats_invariants.pvs]:

```
sats_invariants : THEORY

BEGIN

     IMPORTING dt_lemmas

%%%%% invariants from Cesar's paper %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

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

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

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

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

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


%%%%% Lemmas to prove invariants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %%Lemma 1
     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 2
     %% Case 1: two aircrafts 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 aircrafts in maz and some aircraft with mahf side is 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 in maz and some aircraft is 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 with mahf side is on approach, and
     %%         some aircraft is on 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: some aircraft with mahf side is on the opposite side and
     %%         it precedes an aircraft 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: one aircraft in maz and some aircraft with mahf side is on the opposite side.
     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 7 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)


%% define invariants, lemmas in the order of the proof %%%%%%%%%%%%%%%%

     Invariant_1: LEMMA ( FORALL (s:states): reachable(s) => Inv1(s));

     Invariant_7: LEMMA ( FORALL (s:states): reachable(s) => Inv7(s));

     Invariant_5: LEMMA ( FORALL (s:states): reachable(s) => Inv5(s));

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

     Invariant_6: LEMMA ( FORALL (s:states): reachable(s) => Inv6(s));

     %% The following lemma is used in the proof of Lemma_2, in order to prove Cases 6 and 7.
     Lemma_2_aux: LEMMA ( FORALL (s:states, a:actions, side:Side, ac:Aircraft):
            VerticalApproachInitiation?(a) AND
            side(a) = opposite(side) AND
            Inv3(s) AND Inv4(s) AND
            Lem2_case1(s,side) AND
            Lem2_case3(s,side) AND
            Lem2_case5(s,side) AND
            ac_ready_to_approach?(side,s) AND
            (EXISTS (ac:Aircraft): on?(side,ac,trans(a,s)) AND mahf(ac) = opposite(side)) AND
            reachable(s) AND
            enabled(a,s)
              IMPLIES
            blocked_except_for_one?(ac,opposite(side),trans(a,s)))

     Lemma_2: LEMMA ( FORALL (s:states): reachable(s) => Lem2(s));

     Invariant_3: LEMMA ( FORALL (s:states): reachable(s) => Inv3(s));

     Invariant_4: LEMMA ( FORALL (s:states): reachable(s) => Inv4(s));

     Invariant_2: LEMMA ( FORALL (s:states): reachable(s) => Inv2(s));

END sats_invariants
```