ADVANCED VERIFICATION AND VALIDATION

Design of a Control System for Aircraft Traffic in Airport

2021-2022

Harshini AICH

Akash SHARMA

Table of Contents

1. Introduction

This project aims at designing a system to satisfy the given problem statement. The system is designed to operate in the context of a fully functional airport that has air-traffic going through it continuously.

The airport has resources to be managed (taxiways and runways) and there are aircraft that wish to land and takeoff from the airport. The management and control of these resources and aircraft is the context of the project.

The purpose of this document is to describe the Automatic Controller system designed for the above-mentioned airport. It details all the parameters, events, scenarios the system considers in order to create a complete and full-proof controller.

This document's contents will describe the intended use of the system, the various requirements it satisfies and the series of increasingly-complex models used to build it.

2. Presentation of the System

2.1. Intended Use of the System

The Automatic Controller system exists to **ensure efficient and safe operation of the airport**. It optimizes the usage of the airport resources (runways and taxiways), and prevents collision of aircraft due to poor management of traffic.

Without the proposed controller, the airport could be filled beyond capacity the runways could be congested. Such a controller is relevant for usage in any currently operational airport with air-traffic flowing through it.

The **global mission of the system** is to <u>coordinate the movement of aircraft between the air, the runways and the taxiways</u>.

The Controller performs the following sub-missions:

- Ensure airport operates within its capacity.
- Coordinates takeoff and landing.
- Prevents accidents due to aircraft collision on airport grounds.

The Controller moves the aircraft between the runways and the taxiway in a coordinated manner to achieve its purpose. It transforms planes "on the runway" to planes "on the taxiway" and vice versa, when it performs this movement.

The main **ways in which the Aircraft Traffic Controller fulfils its purpose** are as follows:

- Limits the number of aircraft on ground to a fixed number (20 planes).
- Allows landing/take-off only if runways available.
- Ensures each runway is only occupied by a single aircraft at any given instant.
- Allocates only operational runways to aircraft for takeoff/ landing.

The system **transforms the inputs of planes on ground, planes on runway and planes on taxiway**, by updating the associated variables after each event. The **runways** are also updated to keep track of which planes are where at any given instant. The events that transform these inputs correspond to actual airport operations of **take-off, landing, taxi in/out and the various authorization events**.

2.2. List of requirements

Requirements are the constraints imposed on the system by the problem statement. They are listed under.

REQUIREMENT DESCRIPTION	***IDENTIFIER***
The system is to control planes on ground: taxiway and runway	*FUN-1*
The system controls the access to the runway for take-off or landing	*FUN-2*
The number of planes on ground is limited to 20 max	*CON-1*
The airport has at least one runway	*CON-2*
Aircraft are permanently assigned the authorization to access the runway	*OPE-1*
A plane which is on one runway must be allowed to be there	*OPE-2*
A runway is not occupied by more than one aircraft at a time	*SAF-1*
A runway that is not usable for operation should not be assigned a clearance	*SAF-2*

3. System modelling

Event-B is a notation and method developed from the B-Method and is intended to be used with an incremental style of modelling. The machine part of an Event-B project allows us to define the interactions between the components of the system and add constraints on the system's operation.

The system is first designed keeping in mind only the fundamental requirements. Fulfilling these requirements is essential to ensure the system achieves the main goals. Each refinement addresses more requirements, introducing higher levels of complexity and sophistication to the system.

Event-B allows us to form an abstract model and directly expand the context and refine the machine to create subsequent refinements. It works based off set theory and Boolean logic to create the models. The consistency between models is preserved using mathematical proof rules.

In each model, we have the context which models static properties and the machine which models the dynamic properties. We also discuss the Proof Obligation (PO) and DeadLock Freedom (DLF) rules which determine whether any constraints have been violated or if any scenario has been reached from which there is no further transition.

3.1. Abstract model

The initial abstract model takes into account only the total number of planes on the ground; it does not make the distinction between planes on the runway and planes on the taxiway. The runway and taxiway elements are not present in the scope of the abstract model.

Two events are defined:

1. Landing, which increments the number of planes on the ground.
2. Take-off, which decrements the number of planes on the ground.

The requirements taken into consideration are:

1. FUN-1 (the system controls planes on ground).
2. CON-1 (the number of planes on ground is limited to 20 max).

CONTEXT:

The airport capacity as dictated by CON-1 is defined here. It is represented by the variable "nb_max", which is subsequently set to the positive integer value of 20 by axioms.

```
CONTEXT
  ATC_abstract_ct
CONSTANTS
  nb_max      //  The maximum possible number of planes on the ground i.e. airport capacity
AXIOMS
  axm1   :   nb_max ∈ N      //  The airport capacity is a non-negative integer
  axm2   :   nb_max = 20      //  CON-1: Number of planes on the ground is limited to 20 max
END
```

MACHINE:

The variable that is transformed by the machine is "nb_ground" which is representative of the number of planes that are present in the airport. This number cannot exceed the airport capacity defined by nb_max.

```
MACHINE
  ATC_abstract_mc
SEES
  ATC_abstract_ct
VARIABLES
  nb_ground      //  total number of planes on the ground at any given time
INVARIANTS
  inv1   :   nb_ground ∈ N      //  number of planes on the ground  must be a non-negative quantity
  inv2   :   nb_ground ≤ nb_max      //  CON-1: the number of planes on ground is limited to max capcity
  DLF   :   (nb_ground ∈ N1) ∨ (nb_ground < nb_max)      //  DeadLock Freedom Rule
```

The first event defined is the Initialization event. The number of planes on the ground is given an initial value. Here, it is set to zero, i.e., no planes on the ground at the start of operation.

```
EVENTS
  INITIALISATION  ≙
  STATUS
   ordinary
  BEGIN
   act1   :   nb_ground ≔ 0      //  initially there are no planes on the ground
  END
```

The next event defined is the Takeoff event. When there are planes on the ground, it allows for the aircraft to leave the airport. It updates the nb_ground variable by decrementing it by one aircraft each time the Takeoff event is triggered.

In the event of no planes on the ground (nb_ground = 0), this event is locked and cannot be executed. Once any plane lands, it is able to be triggered. This is ensured by the guard conditions chosen.

```
  TakeOff  ≙
  STATUS
   ordinary
  WHEN
   grd1   :   nb_ground ∈ N1      //  there must be atleast one plane on the ground
  THEN
   act1   :   nb_ground ≔ nb_ground − 1      //  the number of planes on the ground is decremented
  END
```

The last event in the abstract model is the Landing event. When the planes on the ground are less than the airport capacity, new planes are able to enter the airport by landing. This event updates the nb_ground variable by incrementing it by one aircraft each time the event is executed.

Once the airport capacity is filled up (nb_ground = nb_max = 20), the guard conditions ensure that the Landing event is disabled. The event is re-enabled once and plane takes off.

```
  Landing  ≙
  STATUS
    ordinary
  WHEN
    grd1   :   nb_ground < nb_max      //   the number of planes already on the ground must be < airport capacity
  THEN
    act1   :   nb_ground ≔ nb_ground + 1      //   the number of planes on the ground is incremented
  END

END
```

VALIDATION OF PROOF OBLIGATION (PO) INVARIANT RULES & DLF RULE:

A proof obligation is something that has to be proven to show the consistency of the machine, the correctness of theorems, etc. A proof obligation consists of a label, a number of hypothesis that can be used in the proof and a goal – a predicate that must be proven.

Axioms Invariants ⊢ Modified Invariant	INV

Sequent:

A sequent is a formal statement describing something we want to prove. Sequents are of the following form where H is the set of hypotheses (predicates) and G is the goal that can be proved from the predicates. The statement can be read as follows: Under the hypotheses H, prove the goal G.

$$\boxed{H \vdash G}$$

The abstract model has 2 invariants and 2 events. Therefore, a total of 4 Proof Obligation rules need to be applied.

Take_off / inv_1 / INV	*Take_off* / inv_2 / INV
landing / inv1 / INV	*landing* / inv2 / INV

Sequence – 1

Take_off / inv1/ INV:

Axiom	**axm_1**
Axiom	**axm_2**
Invariant	**inv_1**
Invariant	**inv_2**
⊢	
Modified Invariant **inv_1**	

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊢

$nb\_ground - 1 \in \mathbb{N}$

After applying the proof –

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊢

$nb\_ground - 1 \in \mathbb{N}$

MON

$nb\_ground \leq nb\_max$

⊢

$nb\_ground - 1 \in \mathbb{N}$

MON stands for monotonicity of hypotheses. Through the MON hypothesis rule, the unwanted hypothesis will be taken out, leaving only the necessary hypothesis.

Sequence - 2

Take_off / inv2 / INV:

Axiom	**axm_1**
Axiom	**axm_2**
Invariant	**inv_1**
Invariant	**inv_2**
⊢	
Modified Invariant **inv_1**	

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊢

$nb\_ground - 1 \leq nb\_max$

After applying the proof –

nb_max $\in \mathbb{N}$ nb_max = 20 nb_ground $\in \mathbb{N}$ nb_ground $\leq$ nb_max $\vdash$ nb_ground – 1 $\leq$ nb_max	MON	nb_ground $\leq$ nb_max $\vdash$ nb_ground-1 $\leq$ nb_max

Sequence - 3

Landing / inv1 / INV:

Axiom **axm_1** Axiom **axm_2** Invariant **inv_1** Invariant **inv_2** $\vdash$ Modified Invariant **inv_1**		nb_max $\in \mathbb{N}$ nb_max = 20 nb_ground $\in \mathbb{N}$ nb_ground $\leq$ nb_max $\vdash$ nb_ground + 1 $\in \mathbb{N}$

After applying the proof –

nb_max $\in \mathbb{N}$ nb_max = 20 nb_ground $\in \mathbb{N}$ nb_ground $\leq$ nb_max $\vdash$ nb_ground + 1 $\in \mathbb{N}$	MON	nb_ground $\in \mathbb{N}$ $\vdash$ nb_ground + 1 $\in \mathbb{N}$	P2

Sequence - 4

Landing / inv2 / INV:

Axiom **axm_1**
Axiom **axm_2**
Invariant **inv_1**
Invariant **inv_2**
⊦
Modified Invariant **inv_1**

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊦

$nb\_ground + 1 \leq nb\_max$

After applying the proof –

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊦

$nb\_ground + 1 \leq nb\_max$

MON

$nb\_ground \leq nb\_max$

⊦

$nb\_ground + 1 \leq nb\_max$

Deadlock Freedom:

The DLF rule exists to prevent the possibility of a scenario occurring when no events can be triggered, i.e., no state transition that can occur. The rule ensures that at least one set of guard conditions of the events is always fulfilled.

Axioms Invariants ⊦ Disjunction of the guards	DLF

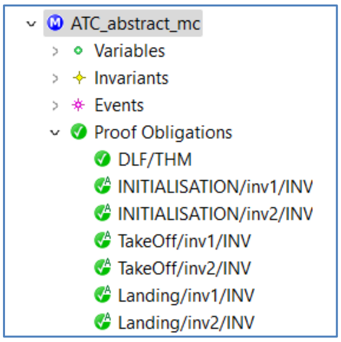
Axiom **axm_1**
Axiom **axm_2**
Invariant **inv_1**
Invariant **inv_2**
⊦
Disjunction of the guards **grd1 V grd2**

$nb\_max \in \mathbb{N}$
$nb\_max = 20$
$nb\_ground \in \mathbb{N}$
$nb\_ground \leq nb\_max$

⊦

$nb\_ground > 0 \lor nb\_ground < nb\_max$

Through this rule, one of the guard conditions will always remain true.



The DLF rule is as below:

```
DLF   :   (nb_ground ∈ N1) ∨ (nb_ground < nb_max)      //   DeadLock Freedom Rule
```

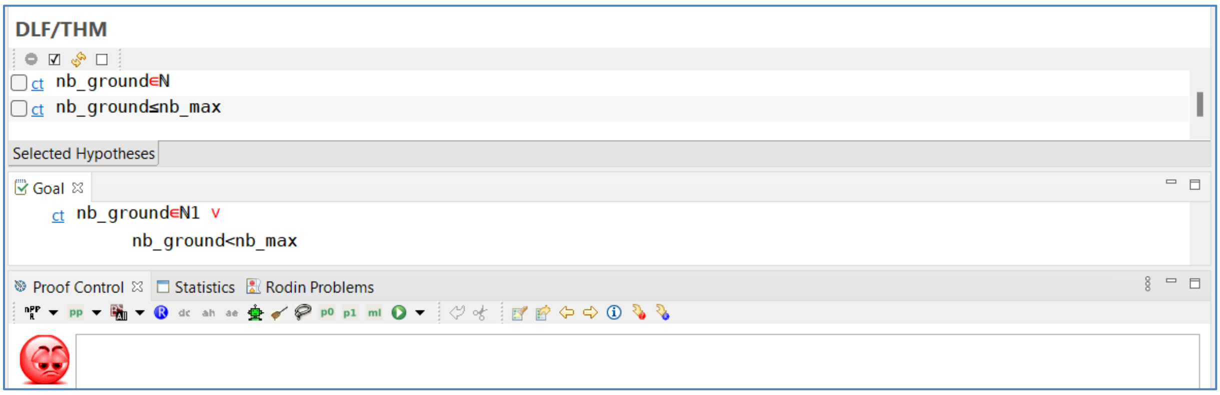
*NB: subsidiary **comment on removing axiom "nb_max=20"***

The axiom "nb_max = 20" fixes the capacity of the airport at a user-defined value. It ensures a fixed quantity of 20 to be used in the guard conditions and invariant checks. To check the necessity of this rule to have the DLF rule be true, we remove it from the context.

```
CONTEXT
  ATC_abstract_ct
CONSTANTS
  nb_max      //  The maximum possible number of planes on the ground i.e. airport capacity
AXIOMS
  axm1   :   nb_max ∈ N      //   The airport capacity is a non-negative integer
END
```

Now, nb_max is randomly assigned, and if nb_max = 0, the system enters into a deadlock. There can be no landing events as nb_ground < nb_max is always false (since now nb_ground = nb_max = 0). And since there are no planes on the ground (nb_ground = 0 from initialization), takeoff can also never occur. Thus, the system is stuck in a deadlock.

The expression (nb_ground ϵ ℕ1) *OR* (nb_ground < nb_max) is continuously false in the DLF rule and thus the rule is seen to be undischarged.



3.2. First refinement

The first refinement adds the distinction between the Runways and the Taxiways in the airport. The previous variable of nb_ground is replaced by three new variables of nb_takeoff, nb_landing and nb_taxi. They represent the number of planes on the runways that want to takeoff, the number of planes on the runways that have landed on the runways and the number of planes on the taxiway respectively.

To transfer planes between the taxiway and the runway, two additional functions are defined; Taxi_In, to move a plane on the runway into the taxiway, and Taxi_Out, to move a plane on a taxiway into a runway.

The requirements taken into consideration are:

1. FUN-1 (the system controls planes on ground).
2. CON-1 (the number of planes on ground is limited to 20 max).
3. FUN-2 (the system controls the access to the runway for take-off or landing).
4. CON-2 (the airport has at least one runway).
5. SAF-1* (a runway is not occupied by more than one aircraft at a time).

* This requirement is considered at a very basic level and not fully satisfied. For this refinement, it is only ensured that *number of planes taking off* + *number of planes landing* = *number of runways*.

CONTEXT:

To account for the addition of the runways component of the airport, a new quantity called "nb_R" is defined which has a value equal to the number of runways present in the airport.

```
CONTEXT
  ATC_ct_1st
EXTENDS
  ATC_abstract_ct
CONSTANTS
  nb_R      //  Represents the number of runways the airport is equipped with
AXIOMS
  axm1   :   nb_R ∈ N1      //   The number of runways is a positive number and there is at least one runway
  axm2   :   nb_R = 5      //   The number of runways is set to a user-defined value
END
```

Here, the user defines the airport as having 5 runways.

MACHINE:

The machine now has three new variables to transform; nb_takeoff, nb_landing and nb_taxiway, that count the number of planes ready to takeoff, just landed and on the taxiway, respectively.

The glue invariant linking this refinement to the abstract model is “nb_ground” which is now equal to the sum of the new variables mentioned above. This new expression of nb_ground is used in the refined definition of the CON-1 invariant.

```
MACHINE
  ATC_mc_1st
REFINES
  ATC_abstract_mc
SEES
  ATC_ct_1st
VARIABLES
  nb_taxiway     //  total number of planes on taxiways
  nb_ground      //  total number of planes on the ground i.e. taxiways and runways
  nb_takeoff     //  total number of planes that are on runways ready to takeoff
  nb_landing     //  total number of planes that have landed on runways
INVARIANTS
  inv2  :  nb_taxiway ∈ N     //  number of planes on the taxiway must be non-negative
  inv3  :  nb_takeoff + nb_landing + nb_taxiway ≤ nb_max     //  CON-1: number of planes on the ground is 20 max
  inv5  :  nb_takeoff + nb_landing ≤ nb_R     //  SAF-1: a runway is not occupied by more than one aircraft at a time (basic version, not fully satisfied)
  inv6  :  nb_ground = nb_takeoff + nb_landing + nb_taxiway     //  -> Glue Invariant
  inv7  :  nb_takeoff ∈ N     //  number of planes taking-off must be non-negative
  inv8  :  nb_landing ∈ N     //  number of planes landing must be non-negative
  DLF   :  (nb_taxiway>0∧nb_takeoff+nb_landing<nb_R)∨nb_takeoff>0∨(nb_takeoff+nb_landing<nb_R∧nb_takeoff+nb_landing+nb_taxiway<nb_max)     //  DeadLock Freedom
           ∨nb_landing>0
```

The new DLF rule is defined by the disjunction of the guard conditions of the events that will be described following this.

The first event is the Initialization event. As before, the number of planes on ground are initialized to zero. But this time the three components of the on ground planes must be initialized separately.

```
EVENTS
  INITIALISATION  ≙     //  total number of planes on ground (taxiways and runways) set to zero
   extended
  STATUS
   ordinary
  BEGIN
   act1  :  nb_ground ≔ 0     //  initially there are no planes on the ground
   act3  :  nb_taxiway ≔ 0
   act4  :  nb_takeoff ≔ 0
   act5  :  nb_landing ≔ 0
  END
```

The next event is the Take-Off event. The planes are now decremented from the planes on the take-off runway directly each time the event is run.

```
Take-off  ≙
 extended
STATUS
 ordinary
REFINES
 Take-off
WHEN
 grd1  :  nb_ground > 0     //  the must be atleast one plane on the ground
 grd4  :  nb_takeoff > 0     //  there must be planes on the runway ready to takeoff
THEN
 act1  :  nb_ground ≔ nb_ground − 1     //  the number of planes on the ground is decremented
 act2  :  nb_takeoff ≔ nb_takeoff − 1     //  the number of planes on the takeoff runway is decremented
END
```

The third event is the Landing event. The planes are incremented to the planes on the landing runway directly each time the event is run. The guard conditions for this event and the Take-Off event are the modified to include the new variables, but are the same logically.

```
Landing  ≙
 extended
STATUS
 ordinary
REFINES
 Landing
WHEN
 grd1  :  nb_ground < nb_max     //  the number of planes already on the ground must be less than the airport capacity
 grd2  :  nb_takeoff + nb_landing < nb_R     //  there must be a free runway
 grd3  :  nb_takeoff + nb_landing + nb_taxiway < nb_max     //  CON-1: number of planes on the ground is 20 max
THEN
 act1  :  nb_ground ≔ nb_ground + 1     //  the number of planes on the ground is incremented
 act2  :  nb_landing ≔ nb_landing + 1     //  the number of planes on the landing runway is incremented
END
```

The fourth event is a new event defined as the <u>Taxi-Out event</u>. This event moves planes that have just landed from the runway onto a taxiway and frees up the runway for use. The event is able to be executed once there are planes land on the runway.

```
Taxi-Out  ≙
STATUS
 ordinary
WHEN
 grd2   :   nb_taxiway > 0      //  there must be planes on the taxiway
 grd3   :   nb_takeoff + nb_landing < nb_R      //  there must be a free runway
THEN
 act1   :   nb_taxiway ≔ nb_taxiway − 1      //  the number of planes on the taxiway is decremented
 act2   :   nb_takeoff ≔ nb_takeoff + 1      //  the number of planes on the runway ready to takeoff is incremented
END
```

The fifth and final event is a new event defined as the <u>Taxi-In event</u>. This event moves planes that are on a taxiway to a runway in preparation for takeoff. The event is executable once there are planes land on the taxiway, and if there are runways free to be used.

```
Taxi_In  ≙
STATUS
 ordinary
WHEN
 grd1   :   nb_landing > 0      //  there must be planes that have landed on the runway
THEN
 act1   :   nb_taxiway ≔ nb_taxiway + 1      //  the number of planes on the taxiway is incremented
 act2   :   nb_landing ≔ nb_landing − 1      //  the number of planes on the landing runway is decremented
END

END
```

<u>Validation of the PO invariant rules and the whole DLF rule:</u>

- ATC_mc_1st
  - Variables
  - Invariants
  - Events
  - Proof Obligations
    - DLF/THM
    - INITIALISATION/inv2/INV
    - INITIALISATION/inv3/INV
    - INITIALISATION/inv5/INV
    - INITIALISATION/inv6/INV
    - INITIALISATION/inv7/INV
    - INITIALISATION/inv8/INV
    - Taxi-Out/inv2/INV
    - Taxi-Out/inv3/INV
    - Taxi-Out/inv5/INV
    - Taxi-Out/inv6/INV
    - Taxi-Out/inv7/INV
    - Take-off/inv3/INV
    - Take-off/inv5/INV
    - Take-off/inv6/INV
    - Take-off/inv7/INV
    - Landing/inv3/INV
    - Landing/inv5/INV
    - Landing/inv6/INV
    - Landing/inv8/INV
    - Taxi_In/inv2/INV
    - Taxi_In/inv3/INV
    - Taxi_In/inv5/INV
    - Taxi_In/inv6/INV
    - Taxi_In/inv8/INV

The DLF rule is below:

```
DLF  :  (nb_taxiway>0∧nb_takeoff+nb_landing<nb_R)∨nb_takeoff>0∨(nb_takeoff+nb_landing<nb_R∧nb_takeoff+nb_landing+nb_taxiway<nb_max)   //  DeadLock Freedom
        ∨nb_landing>0
```

No variants are required to complete this refinement.

3.3. Second refinement

The second refinement adds the notion of sets to the Airport Controller system model. This allows relations between set elements like Planes and Runways. Now, the requirement of SAF-1 can be fully satisfied (it was only partially satisfied in the first refinement).

This also allows us to uniquely identify planes and track the movement of a particular plane through the airport.

The requirements taken into consideration are:

1. FUN-1 (the system controls planes on ground).
2. FUN-2 (the system controls the access to the runway for take-off or landing).
3. CON-1 (the number of planes on ground is limited to 20 max).
4. CON-2 (the airport has at least one runway).
5. SAF-1 (a runway is not occupied by more than one aircraft at a time).

<u>CONTEXT</u>:

The second refinement implements sets and to achieve this a set PLANES is defined which contains all the possible planes that could operate in the airport as elements.

The set is finite and of size equal to the capacity of the airport.

```
CONTEXT
  ATC_ct_2nd
EXTENDS
  ATC_ct_1st
SETS
  PLANES     //  The set containing all planes that could operate within the airport and airspace
AXIOMS
  axm1  :  finite(PLANES)     //  The set of planes is a finite set
  axm2  :  card(PLANES) = nb_max     //  The number of elements (planes) in the set is 20 max (derived from CON-1)
  axm3  :  ∀s,p·(s ⊆ PLANES ∧ p ∈ s) ⇒ card(s\{p}) = card(s) − 1     //  Axiom directing Rodin how to process removal of an element plane from a set
END
```

An additional axiom is required to set the behavior of Rodin in the event of the negation of an element from a set. It tells Rodin that the cardinality of a set decreases by one when a single element is removed from the set. It is done to aid in the preservation of the invariants after they have been modified by events.

<u>MACHINE</u>:

The machine now transforms the sets corresponding to the variables previously used. The sets pl_rt, pl_rl and pl_t contains the elements counted by nb_takeoff, nb_landing and nb_taxiway respectively. These will be the glue invariants that link the second refinement to the first refinement.

```
MACHINE
  ATC_mc_2nd
REFINES
  ATC_mc_1st
SEES
  ATC_ct_2nd
VARIABLES
  nb_taxiway     //  total number of planes on taxiways
  nb_ground      //  total number of planes on the ground i.e. taxiways and runways
  pl_rt      //  set containing all planes that are on the runway ready to takeoff
  pl_rl      //  set containing all planes that have landed on the runway
  pl_t      //  set containing all planes that are on the taxiway
  nb_takeoff     //  total number of planes that are on runways ready to takeoff
  nb_landing     //  total number of planes that have landed on runways
INVARIANTS
  inv1  :  pl_rt ⊆ PLANES     //  Planes is the superset containing all planes in the airport, and the previously defined sets are subsets of it
  inv2  :  pl_rl ⊆ PLANES
  inv3  :  pl_t ⊆ PLANES
  inv4  :  card(pl_t) = nb_taxiway     //  -> Glue Invariant
  inv5  :  card(pl_rt) = nb_takeoff     //  -> Glue Invariant
  inv6  :  card(pl_rl) = nb_landing     //  -> Glue Invariant
  inv7  :  pl_rt ∩ pl_rl = ∅     //  The sets of planes on the take-off runways, the landing runways and the taxiway are all disjoint
  inv8  :  pl_rt ∩ pl_t = ∅     //  i.e. a plane can either be taking off, or be landing or be in the taxiway
  inv9  :  pl_rl ∩ pl_t = ∅
           ∃p·p ∈ PLANES ⇒ (card(pl_t) > 0 ∧ p ∈ pl_t ∧ (card(pl_rt) + card(pl_rl) < nb_R)) ∨ (p ∈ pl_rt ∧ card          //  DeadLock Freedom Rule
  DLF   :  (pl_rt) > 0) ∨ ((card(pl_rt) + card(pl_rl) + card(pl_t) < nb_max) ∧ (card(pl_rt) + card
           (pl_rl) < nb_R) ∧ p ∈ PLANES\(pl_rt ∪ pl_rl ∪ pl_t)) ∨ (card(pl_rl) > 0 ∧ p ∈ pl_rl)
```

We also declare the sets to be disjoint. This reflects the real-life situation where a plane can only do one of the following actions at a time: it can either takeoff, land or it can be taxiing. It cannot do two events simultaneously. This invariant helps in the PO rules discharging.

The DLF rule must be modified to not only include the new guards, but also to ensure the existence of an element p belonging to the superset PLANES.

The first event described is the Initialization event. The sets corresponding to the on-ground planes are initialized to be empty sets.

```
EVENTS
  INITIALISATION  ≙
   extended
  STATUS
   ordinary
  BEGIN
   act1  :  nb_ground ≔ 0     //  initially there are no planes on the ground
   act3  :  nb_taxiway ≔ 0
   act4  :  nb_takeoff ≔ 0
   act5  :  nb_landing ≔ 0
   act6  :  pl_rl ≔ ∅     //  The sets of planes on the ground are initialised to empty sets
   act7  :  pl_t ≔ ∅
   act8  :  pl_rt ≔ ∅
  END
```

The next event is the Take Off event. The guard conditions are redefined to include the set notations. In addition to numerical increments, a negation of sets facilitates removing of a plane from the takeoff runway and the airport.

```
TakeOff  ≙
 extended
STATUS
 ordinary
REFINES
 Take-off
ANY
 p      //  Parameter to denote an element of the sets Planes
WHERE
 grd1  :  nb_ground > 0      //  the must be atleast one plane on the ground
 grd4  :  nb_takeoff > 0      //  there must be planes on the runway ready to takeoff
 grd5  :  p ∈ pl_rt      //  A plane currently on the takeoff runway
 grd6  :  card(pl_rt) > 0      //  Takeoff runway planes is a non-empty set
THEN
 act1  :  nb_ground ≔ nb_ground − 1      //  the number of planes on the ground is decremented
 act2  :  nb_takeoff ≔ nb_takeoff − 1      //  the number of planes on the takeoff runway is decremented
 act3  :  pl_rt ≔ pl_rt\{p}      //  Remove the particular plane from the set of takeoff planes
END
```

The third event described is the Landing event. This adds planes from the superset PLANES into the airport space. Again, the guards are rewritten to include the set notation. A union of sets is used to add a landing plane into the landing runways set.

```
Land  ≙
 extended
STATUS
 ordinary
REFINES
 Landing
ANY
 p      //  Parameter to denote an element of the sets Planes
WHERE
 grd1  :  nb_ground < nb_max      //  the number of planes already on the ground must be less than the airport capacity
 grd2  :  nb_takeoff + nb_landing < nb_R      //  there must be a free runway
 grd3  :  nb_takeoff + nb_landing + nb_taxiway < nb_max      //  CON-1: number of planes on the ground is 20 max
 grd4  :  card(pl_rt) + card(pl_rl) < nb_R      //  The planes on the runways should be less than number of runways (i.e. a runway must be free)
 grd5  :  p ∈ PLANES\(pl_rt ∪ pl_rl ∪ pl_t)      //  A plane currently not on the ground
 grd6  :  card(pl_rt) + card(pl_rl) + card(pl_t) < nb_max      //  The number of planes on the ground should be less than 20 (CON-1 refined)
THEN
 act1  :  nb_ground ≔ nb_ground + 1      //  the number of planes on the ground is incremented
 act2  :  nb_landing ≔ nb_landing + 1      //  the number of planes on the landing runway is incremented
 act3  :  pl_rl ≔ pl_rl ∪ {p}      //  Add the particular plane to the set of landing planes
END
```

The fourth event is the Enter Runway event. Here, planes are negated from the set of Taxiway planes and appended to the Takeoff Runway set by a union. The guards are rewritten to include sets, but the logic remains the same as the refined Taxi Out event.

```
Enter_rwy  ≙
 extended
STATUS
 ordinary
REFINES
 Taxi-Out
ANY
 p      //  Parameter to denote an element of the sets Planes
WHERE
 grd2  :  nb_taxiway > 0      //  there must be planes on the taxiway
 grd3  :  nb_takeoff + nb_landing < nb_R      //  there must be a free runway
 grd4  :  p ∈ pl_t      //  A plane currently on the taxiway
 grd5  :  card(pl_t) > 0      //  Taxiway planes is a non-empty set
 grd6  :  card(pl_rt) + card
          (pl_rl) < nb_R      //  The number of planes on the runways should be less than number of runways (i.e. a runway must be free)
THEN
 act1  :  nb_taxiway ≔ nb_taxiway − 1      //  the number of planes on the taxiway is decremented
 act2  :  nb_takeoff ≔ nb_takeoff + 1      //  the number of planes on the runway ready to takeoff is incremented
 act3  :  pl_rt ≔ pl_rt ∪ {p}      //  Add the particular plane to the set of takeoff planes
 act4  :  pl_t ≔ pl_t\{p}      //  Remove the particular plane from the set of taxiway planes
END
```

The fifth event is the Leave Runway event. Here, planes are negated from the set of Landed planes and appended to the Taxiways set by a union. The guards are rewritten to include sets, but the logic remains the same as the refined Taxi In event.

```
Leav_rwy  ≙
 extended
STATUS
 ordinary
REFINES
 Taxi_In
ANY
 p      //   Parameter to denote an element of the sets Planes
WHERE
 grd1   :   nb_landing > 0      //   there must be planes that have landed on the runway
 grd2   :   p ∈ pl_rl      //   A plane currently on the landing runway
 grd3   :   card(pl_rl) > 0      //   Landing runway planes is a non-empty set
THEN
 act1   :   nb_taxiway ≔ nb_taxiway + 1      //   the number of planes on the taxiway is incremented
 act2   :   nb_landing ≔ nb_landing − 1      //   the number of planes on the landing runway is decremented
 act3   :   pl_rl ≔ pl_rl∖{p}      //   Remove the particular plane from the set of landing planes
 act4   :   pl_t ≔ pl_t ∪ {p}      //   Add the particular plane to the set of taxiway planes
END

END
```

Validation of the PO invariant rules and the DLF rule:

- ATC_mc_2nd
  - Variables
  - Invariants
  - Events
  - Proof Obligations
    - inv4/WD
    - inv5/WD
    - inv6/WD
    - DLF/WD
    - DLF/THM
    - INITIALISATION/inv4/INV
    - INITIALISATION/inv5/INV
    - INITIALISATION/inv6/INV
    - INITIALISATION/inv7/INV
    - INITIALISATION/inv8/INV
    - INITIALISATION/inv9/INV
    - Enter_rwy/grd5/WD
    - Enter_rwy/grd6/WD
    - Enter_rwy/inv4/INV
    - Enter_rwy/inv5/INV
    - Enter_rwy/inv7/INV
    - Enter_rwy/inv8/INV
    - Enter_rwy/inv9/INV
    - TakeOff/grd6/WD
    - TakeOff/inv5/INV
    - TakeOff/inv7/INV
    - TakeOff/inv8/INV
    - Land/grd4/WD
    - Land/grd6/WD
    - Land/inv6/INV
    - Land/inv7/INV
    - Land/inv9/INV
    - Leav_rwy/grd3/WD
    - Leav_rwy/inv4/INV
    - Leav_rwy/inv6/INV
    - Leav_rwy/inv7/INV
    - Leav_rwy/inv8/INV
    - Leav_rwy/inv9/INV

The DLF Rule is as below:

```
DLF  :  ∃p·p ∈ PLANES ⇒ (card(pl_t) > 0 ∧ p ∈ pl_t ∧ (card(pl_rt) + card(pl_rl) < nb_R)) ∨ (p ∈ pl_rt ∧ card   //  DeadLock Freedom Rule
        (pl_rt) > 0) ∨ ((card(pl_rt) + card(pl_rl) + card(pl_t) < nb_max) ∧ (card(pl_rt) + card
        (pl_rl) < nb_R) ∧ p ∈ PLANES∖(pl_rt ∪ pl_rl ∪ pl_t)) ∨ (card(pl_rl) > 0 ∧ p ∈ pl_rl)
```

3.4. Third refinement

The final layer of refinement takes into account the notion of clearances, authorizations and the scenarios of runways being closed off due to construction work, vehicle occupancy etc.

All the requirements initially described are completely satisfied by this refinement. They are:

1. FUN-1 (the system controls planes on ground).
2. FUN-2 (the system controls the access to the runway for take-off or landing).
3. CON-1 (the number of planes on ground is limited to 20 max).
4. CON-2 (the airport has at least one runway).
5. SAF-1 (a runway is not occupied by more than one aircraft at a time).
6. SAF-2 (a runway that is not usable for operation should not be assigned a clearance).
7. OPE-1 (aircraft are permanently assigned the authorization to access the runway).
8. OPE-2 (a plane which is on one runway must be allowed to be there OPE-2).

4 additional events are defined-

1. Accept_clearance: parametric event enables to associate a plane with a runway (as one action); the other action adds this runway to rwy_occ. This event basically gives the go ahead to any plane wanting to enter a particular runway and books the runway.
2. Add_rwy_nok: parametric event enables to add a not cleared runway to rwy_nok. In the event of a runway being closed off, this event prevents any plane from getting authorization for that runway and consequently, the clearance.
3. Free_rwy_occ: parametric event enables to remove a pair of (plane, runway) from the clearance list (as one action); the other action removes this runway from rwy_occ. This event is triggered when an aircraft leaves the runway. It no longer has clearance for that runway and the runway is open to other aircraft.
4. Free_rwy_nok: parametric event enables to remove a runway from rwy_nok. This event is triggered when a closed off runway is ready to resume operation.

CONTEXT:

A new set RUNWAYS is introduced which holds all the runways of the airport as elements.

```
CONTEXT
  ATC_ct_3rd
EXTENDS
  ATC_ct_2nd
SETS
  RUNWAYS     //  Set containing all the runways that the airport has (functional and non-fuctional)
AXIOMS
  axm1   :   finite(RUNWAYS)      //  The set of runways is a finite dimension set
  axm2   :   card(RUNWAYS) = nb_R      //  The size of the runways set is equal to the number of runways the airport has
  axm3   :   RUNWAYS ≠ ∅      //  CON-2: The airport has at least one runway (i.e. RUNWAYS cannot be an empty set)
END
```

MACHINE:

In the final refinement, every movement to and from the runway is dictated by the clearance which is a variable set that associates one plane to one runway at a time. The variable *aut* relates the set PLANES to the set RUNWAYS. In other words, it maintains all possible clearances that any plane can have.

The clearance is checked for the events where the plane is going on the runway while it is removed when the plane leaves the runway.

```
MACHINE
  ATC_mc_3rd
REFINES
  ATC_mc_2nd
SEES
  ATC_ct_3rd
VARIABLES
  nb_taxiway      //  total number of planes on taxiways
  nb_ground      //  total number of planes on the ground i.e. taxiways and runways
  pl_rt     //  set containing all planes that are on the runway ready to takeoff
  pl_rl     //  set containing all planes that have landed on the runway
  pl_t     //  set containing all planes that are on the taxiway
  nb_takeoff      //  total number of planes that are on runways ready to takeoff
  nb_landing      //  total number of planes that have landed on runways
  rwy_occ      //  number of runways for which a plane has been given clearance
  rwy_nok      //  number of runways not suitable for operation
  aut     //  a subset of the cartesian product PLANES x RUNWAYS
  clear     //  variable associating a plane to a runway. One plane can only have a clearance for one runway at a time
INVARIANTS
  inv1   :   rwy_occ ⊆ RUNWAYS      //  Both rwy_occ and rwy_nok are subsets of the larger set RUNWAYS
  inv2   :   rwy_nok ⊆ RUNWAYS
  inv3   :   aut ⊆ PLANES × RUNWAYS      //  contains all possible mapping of planes to runways that are operational
  inv4   :   clear ∈ PLANES ⤔ RUNWAYS      //  clearance given to planes basically books a particular runway for them
  inv6   :   rwy_occ ∩ rwy_nok = ∅      //  None of the non-operational runways must be given clearance
```

The cartesian product contains all the possible pairings of planes and runways. Initially, all planes have authorization for all runways since no runways are in the NOK set.

```
EVENTS
  INITIALISATION  ≙
   extended
  STATUS
   ordinary
  BEGIN
   act1   :   nb_ground ≔ 0      //   initially there are no planes on the ground
   act2   :   nb_taxiway ≔ 0
   act3   :   nb_takeoff ≔ 0
   act4   :   nb_landing ≔ 0
   act5   :   pl_rl ≔ ∅      //   The sets of planes on the ground are initialised to empty sets
   act6   :   pl_t ≔ ∅
   act7   :   pl_rt ≔ ∅
   act8   :   rwy_occ ≔ ∅      //   The sets of runways initialised to empty sets
   act9   :   rwy_nok ≔ ∅
   act10   :   clear ≔ ∅      //   Initially, no plane has been assigned a clearance
   act11   :   aut ≔ PLANES × RUNWAYS
  END
```

Enter_rwy & Land : Clearance is checked

```
Enter_rwy  ≙
 extended
STATUS
 ordinary
REFINES
 Enter_rwy
ANY
 p     //  Parameter to denote an element of the sets Planes
 r     //  Parameter to denote an element of the set RUNWAYS
WHERE
 grd1  :  nb_taxiway > 0     //  there must be planes on the taxiway
 grd2  :  nb_takeoff + nb_landing < nb_R     //  there must be a free runway
 grd3  :  nb_taxiway > 0     //  there must be planes on the taxiway
 grd4  :  nb_takeoff + nb_landing < nb_R     //  there must be a free runway
 grd5  :  p ∈ pl_t     //  A plane currently on the taxiway
 grd6  :  card(pl_t) > 0     //  Taxiway planes is a non-empty set
 grd7  :  card(pl_rt) + card(pl_rl) < nb_R     //  The number of planes on the runways should be less than number of runways (i.e. a runway must be free)
 grd8  :  p↦r ∈ clear     //  Checking whether the maplet p --> r exists in the clearance set
THEN
 act1  :  nb_taxiway ≔ nb_taxiway − 1     //  the number of planes on the taxiway is decremented
 act2  :  nb_takeoff ≔ nb_takeoff + 1     //  the number of planes on the runway ready to takeoff is incremented
 act3  :  pl_rt ≔ pl_rt ∪ {p}     //  Add the particular plane to the set of takeoff planes
 act4  :  pl_t ≔ pl_t∖{p}     //  Remove the particular plane from the set of taxiway planes
END
```

```
Land  ≙
STATUS
 ordinary
REFINES
 Land
ANY
 p     //  Parameter to denote an element of the sets Planes
 r     //  Parameter to denote an element of the set RUNWAYS
WHERE
 grd1  :  nb_ground < nb_max     //  the number of planes already on the ground must be less than the airport capacity
 grd2  :  nb_takeoff + nb_landing < nb_R     //  there must be a free runway
 grd3  :  nb_takeoff + nb_landing + nb_taxiway < nb_max     //  CON-1: number of planes on the ground is 20 max
 grd4  :  card(pl_rt) + card(pl_rl) < nb_R     //  The planes on the runways should be less than number of runways (i.e. a runway must be free)
 grd5  :  p ∈ PLANES∖(pl_rt ∪ pl_rl ∪ pl_t)     //  A plane currently not on the ground
 grd6  :  card(pl_rt) + card(pl_rl) + card(pl_t) < nb_max     //  The number of planes on the ground should be less than 20 (CON-1 refined)
 grd7  :  p↦r ∈ clear
THEN
 act1  :  nb_ground ≔ nb_ground + 1     //  the number of planes on the ground is incremented
 act2  :  nb_landing ≔ nb_landing + 1     //  the number of planes on the landing runway is incremented
 act3  :  pl_rl ≔ pl_rl ∪ {p}     //  Add the particular plane to the set of landing planes
END
```

Leav_rwy & Takeoff : Clearance is removed

```
Leav_rwy  ≙
STATUS
 ordinary
REFINES
 Leav_rwy
ANY
 p     //  Parameter to denote an element of the sets Planes
 r     //  Parameter to denote an element of the set RUNWAYS
WHERE
 grd1  :  nb_landing > 0     //  there must be planes that have landed on the runway
 grd2  :  p ∈ pl_rl     //  A plane currently on the landing runway
 grd3  :  card(pl_rl) > 0     //  Landing runway planes is a non-empty set
 grd4  :  r ∈ rwy_occ     //  Runway should be occupied
THEN
 act1  :  nb_taxiway ≔ nb_taxiway + 1     //  the number of planes on the taxiway is incremented
 act2  :  nb_landing ≔ nb_landing − 1     //  the number of planes on the landing runway is decremented
 act3  :  pl_rl ≔ pl_rl∖{p}     //  Remove the particular plane from the set of landing planes
 act4  :  pl_t ≔ pl_t ∪ {p}     //  Add the particular plane to the set of taxiway planes
 act5  :  clear ≔ clear∖{p↦r}     //  Removing clearance
END
```

```
TakeOff  ≙
STATUS
 ordinary
REFINES
 TakeOff
ANY
 p     //  Parameter to denote an element of the sets Planes
 r     //  Parameter to denote an element of the set RUNWAYS
WHERE
 grd1  :  nb_ground > 0     //  the must be atleast one plane on the ground
 grd2  :  nb_takeoff > 0     //  there must be planes on the runway ready to takeoff
 grd3  :  p ∈ pl_rt     //  A plane currently on the takeoff runway
 grd4  :  card(pl_rt) > 0     //  Takeoff runway planes is a non-empty set
 grd5  :  r ∈ rwy_occ
THEN
 act1  :  nb_ground ≔ nb_ground − 1     //  the number of planes on the ground is decremented
 act2  :  nb_takeoff ≔ nb_takeoff − 1     //  the number of planes on the takeoff runway is decremented
 act3  :  pl_rt ≔ pl_rt∖{p}     //  Remove the particular plane from the set of takeoff planes
 act4  :  clear ≔ clear∖{p↦r}
END
```

Everytime a plane leaves a runway, it has to ask for a new clearance the next time it needs to access a runway. Clearance is provided through the Accept_clearance event as described below.

Accept_clearance: This event relates a particular plane to a particular runway while making sure no other plane has clearance for the same runway. It also books the runway for the plane by adding it to the rwy_occ set.

```
Accept_clearance  ≙
STATUS
 ordinary
ANY
 r      //  Parameter to denote an element of the set RUNWAYS
 p      //  Parameter to denote an element of the set PLANES
WHERE
 grd1   :   r ∈ RUNWAYS\(rwy_occ ∪ rwy_nok)     //  The runway should be suitable and unoccupied
 grd2   :   p ∈ PLANES\(pl_rt ∪ pl_rl)     //  The plane should either be in the taxiway or in the air
 grd3   :   p ∉ dom(clear)     //  Plane should not already have clearance
 grd4   :   r ∉ ran(clear)     //  Runway should not already be occupied or booked
THEN
 act1   :   clear(p) ≔ r     //  Giving clearance
 act2   :   rwy_occ ≔ rwy_occ ∪ {r}     //  Booking the runway for the particular plane
END
```

Add_rwy_nok: In the event a runway is not suitable for operation, this event moves it to the rwy_nok set. It removes a clearance for the runway (if any exist) and blocks the runway.

```
Add_rwy_nok  ≙
STATUS
 ordinary
ANY
 r      //  Parameter to denote an element of the set RUNWAYS
 p      //  Parameter to denote an element of the set PLANES
WHERE
 grd1   :   r ∈ RUNWAYS\(rwy_occ ∪ rwy_nok)     //  the runway should initially be suitable for operation
 grd2   :   p ∈ PLANES
THEN
 act1   :   rwy_nok ≔ rwy_nok ∪ {r}     //  Adding the runway to the NOK set
 act3   :   clear ≔ clear\{p↦r}     //  Removing clearance in case a plane has access to the runway
END
```

Free_rwy_occ: Everytime a plane leaves a runway, it has to be made available for other possible planes. This event removes the plane from the rwy_occ set.

```
Free_rwy_occ  ≙
STATUS
 ordinary
ANY
 r
WHERE
 grd1   :   r ∈ rwy_occ     //  Runway should be occupied
 grd2   :   card(rwy_occ) ≠ card(pl_rt) + card(pl_rl)     //  This is to make sure that the plane has physically left the runway
THEN
 act1   :   rwy_occ ≔ rwy_occ\{r}     //  Removing the runway from the occupied set
END
```

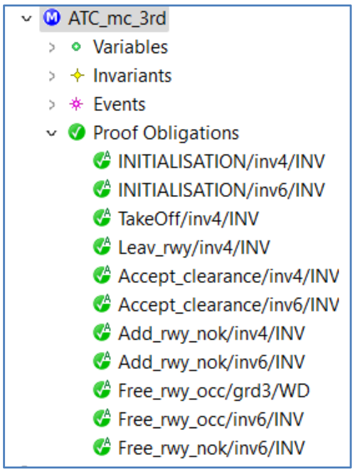
Free_rwy_nok: This event removes a runway form the rwy_nok set after it has been cleared for operation.

```
Free_rwy_nok  ≙
STATUS
 ordinary
ANY
 r      //  Parameter to denote an element of the set RUNWAYS
 p      //  Parameter to denote an element of the set PLANES
WHERE
 grd1   :   r ∈ rwy_nok     //  Runway should initially be closed off
 grd2   :   p ∈ PLANES
THEN
 act1   :   rwy_nok ≔ rwy_nok\{r}     //  Opening the runway for operation
END

END
```

Remark: If the Accept_clearance event is triggered, the planes in the taxiway (if any) are given priority for clearance over planes in the air wanting to land.

Validation **of the PO invariant rules:**



4. Conclusion

The initial problem statement has been satisfied using the incremental modelling style of Event-B. The models were expanded form simple numeric models to include seats and relations. This allowed us to simulate a system close to a real-life situation.

Event-B has many capabilities but there are some scenarios it cannot simulate. For example, simultaneous landing and takeoff, or random assigning of clearance to a new plane for landing while there are planes on the taxiway. Future works could include assigning runways based on direction of takeoff/ landing or sorting criteria of runways based on size of planes that can land on it.

5. References

1. Rodin User's Handbook v.2.8, Michael Jastram.
2. Chaudemar. J.C. « Basic Introduction to Systems Engineering ». 2018