Verification of SATS Landing Protocol: a Case Study

Dileep Kini

University of Illinois, Urbana Champaign

1 Introduction

The *Small Aircraft Transportation System*(SATS) was introduced by NASA[1] in order to manage increased throughput at small airports. Small airports usually have limited or no access to facilities like control towers and radar to help aircrafts land. SATS provides for a protocol which requires minimal assistance from the airport. The aircrafts involved get only landing sequence information from the airport. They cooridnate with other aircrafts to acquire rest of the information required to land.

The most obvious concern with any such protocol is safety of the participating aircrafts. In this scenario it is defined as maintaining a minimal separation between the aircrafts involved. The aim of this project is to formally model a simplified version of this system and verify it's correctness using the tools PVS and UPPAAL.

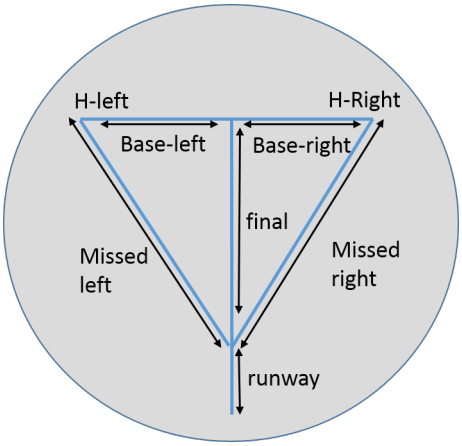
There have been previous attempts[2] [3] to tackle the verication of SATS protocol. This work is intended to be combinination of techniques and rediscovering proofs for a case study as part of a course project.

2 The SATS framework

2.1 Overview

The SATS framework describes a set of rules that need to be followed by the participanting aircrafts which are sufficient to ensure separation and safe landing. There are two components of the SATS model: (a) The Self Controlled Area (SCA) (b) The Airport Management Module (AMM). The aircrafts involved are assumed to be capable of communicating with other aircrafts in the SCA.

The framework that we consider here is simpler than the orignial one introduced by NASA. The original framework allowed for lateral entries into the SCA, where as we dont. This does not affect the continuous part of the aircraft in any way.



Top View

The Self Controlled Area consists of pre decided locations and areas which the aircrafts need to be in. To begin with all aircrafts are assumed to be flying outside the SCA. The aircraft is allowed to enter only at the holding locations at 3000 feet. An aircraft wishing to land requests the AMM for entry. When it is granted entry it is assigned a *sequence number* and *missed approach side*. The aircraft then depending upon location/sequence of other aircrafts in the SCA follows through the lower holding location, base segment, final segment and finally the runway. If for some reason an aircraft in the landing segments is unable to make the landing it goes to the missed approach segment assigned to it in the beginning. It subsequently enters one of the holding location at which point it is put at the end of the sequence.

The Airport Management Module should be in place at the airport, it should be capable of communicating with aircrafts in the SCA and those in the vicinity wanting to enter it. The AMM responsibility is limited to maintaining and assigning sequence numbers. The AMM is kept simple because it is assumed that SATS is catering to airports have limited facilities.

2.2 Formal Model

Each aircraft(identified by i) in the system is modeled as a hybrid automata with the states having the following components . The number i is the unique identifier of the aircraft (not it's sequence number). The aircraft's location is identified by the variable loc_i which varies over L (described in the next section). Its physical distance along the approach segment is measured with a continuous variable x_i. The variable m_i maintains the aircraft's missed apporach side and seq_i keeps it's landing sequence number. A discrete global variable c is maintained at the AMM which represents the highest sequence number held by any aircraft in the SCA, its default value is 0

The type of the variables is summarized in the following table

Variable ($i \in \mathbb{N}$)	Type
x_i	$\mathbb{R}_{\geq 0}$
seq_i	$\mathbb{N}_{>0}$
m_i	$\{l, r\}$
loc_i	L
c	$\mathbb{N}$

The hybrid automaton for the ith aircraft is given by $\mathcal{A}_i = (X_i, Q_i, \Theta_i, E_i, \mathcal{D}_i, \mathcal{T}_i)$ where

$$
\begin{aligned}
X_i &= \{x_i, seq_i, m_i, loc_i\} \\
Q_i &= val(X_i) \\
\Theta_i &= (seq_i = 0 \wedge loc_i = fly) \\
\mathcal{T}_i &= \{t \in trajs(x_i) \mid \dot{t(x_i)} = 1\}
\end{aligned}
$$

In the following sections we describe the locations and the discrete transitions associated with this hybrid automata. Each discrete transition is considered to be an external action and hence we do not make any special mention of them.

Locations (L)

- fly: this is the single state in which every aircraft is assumed to be in before entering the SCA.
- h3r/h3l: these are the initial holding locations on either side of the runway. They are located 3000 feet above ground level. Aircrafts hold in this location before descending.
- h2r/h2l: like above these are the holding locations on either side which but are 2000 feet above ground. These are the last holding location before beginning the approach to the runway.
- br/bl: These are the inital segments of the approach to the runway. Starting from these states the value of the variable x_i becomes significant. x_i represents the distance along the approach and missed segment (if the aircraft misses the approach).
- fin: final segment is the end of the approach. It starts where the two base segments intersect at the T and goes along to the runway. The value. As stated above the value x_i is continued to be measured along this segment
- run: If the approach is successful it enters the this final state.
- mr/ml: These states represent the missed approach segment on either side of the runway. When the approach to runway is unsuccessfull the aircraft moves to here instead of run.

Actions and their Transitions

- fly-h3(d=r/l)(e=r/l): allows aircraft to move from fly to a holding location at 3000 feet
  - Guard: (i) no other aircraft occupies the location h3d it is going to enter (ii) there are less than 2 aircrafts in the SCA which are assigned e as their missed approach side
  - Update: $m_i := e$, $c := c + 1$, $seq_i := c$
- h3-h2(d=r/l): allows aircraft to move from the higher to lower holding zone on the same side
  - Guard: h2d is not occupied
- h2-b(d=r/l): move from holding and begin approach along base segment of the same side
  - Guard: $x_{p(i)} \geq 4$, where $p(i)$ is the aircraft with one lower sequence number
  - Update: $x_i := 0$
- b-fin(d=r/l): after covering length of base segment start the finish segment
  - Guard: $x_i = l(base)$
- fin-run: after covering length of finish segment land on runway
  - Guard: $x_i = l(base) + l(fin)$
- fin-m(d=r/l): if approach to runway unsuccessful move to md after covering length of the finish segment
  - Guard: $x_i = l(base) + l(fin) \wedge m_i = d$
- m-h2(d=r/l): after covering length of the missed approach zone enter holding zone at 2000 feet if both holding zones on that side are unoccuppied, and assume the last position in the sequence
  - Guard: $x_i = l(base) + l(fin) + l(miss)$ and h2,h3 at d are unoccupied
  - Update: ($\forall j \in$ SCA $seq_j := seq_j - 1$), $seq_i = c$
- m-h3(d=r/l): similar to above except it is sufficient for h3 at d to be free

3 Verification of SATS

The SATS system that we consider can have an arbitrarily large number of aircrafts. This makes the state space infinite and hard to verify the safety of any non-trivial property. The approach followed here is the same as in [Ref]. We prove that inspite of unbounded number of aircrafts in the system there can be atomst four aircrafts in the SCA. This is a property purely of the discrete transition system and is independent of the dynamics of it's continuous variables. Hence we use PVS to model it's discrete part (obtained by dropping the x_is) and mechanically prove this fact. The PVS specification is provided in the appendix. This specification is built on top of the simplemachine template.

To prove that there are atmost 4 aircrafts in the SCA we prove something slightly stronger: At any point there are atomst 2 aircrafts in the SCA which are assigned the same side for missed approach for either side. This property holds because when an aicraft gains entry into the SCA it is ensured that there are

fewer than 2 aircrafts assigned to the particular missed approach side.

Now we claim that the same property will hold for the transition system extended with continuous variables. To prove this it is sufficient to provide a forward simulation from the hybrid automaton to it's discrete counterpart

Once the four aircraft property is established we model check the desired safety property against a system consisting of 4 aircrafts. In this case study we assume that the speed of all aircrafts through the various segments is constant (= 1). This allows us to model it as a timed automata. We use UPPAAL to model our system as parallel composition of 4 aircrafts. UPPAAL allows for model checking TCTL properties.

For properties that are universally quantified over two aicrafts we need only consider two of them, because the only thing different between the aircrafts is their id to which the protocol is oblivious. We model check the system against the following properties by encoding them in TCTL. UPPAAL allows for a restricted form of TCTL in which temporal properties are not allowed to be nested. The safety properties which we are interested in can be easily encoded in it.

- Any two aircrafts in the segments approaching the runway are always separated by a distance of 4 units.

  $$\forall\Box((\text{onApp}_0 \wedge \text{onApp}_1) \rightarrow (4 \leq |x_0 - x_1|))$$

  where $onApp_i$ is a macro which is true iff the aircraft i is on the approach. So the formula can be read as : For all paths from the starting state it is always true that if aircraft 0 and aircraft 1 are on the approach then the distance between them is atleast 4.
- The above property states that two aircrafts on the approach should be separated but it does not say anything about their order. What we also want is the following: For two aircrafts in the segments approaching the runway , the one with lower sequence number is always ahead of the other.

  $$\forall\Box((seq_0 = 1) \wedge (seq_1 = 2) \wedge \text{onApp}_0 \wedge \text{onApp}_1 \rightarrow (x_0 > x_1))$$

  This formula can be read as : for all paths from the starting states if aircraft A0 has sequence 1 and aircraft A1 has sequence 2, then A0 is ahead of A1.

4 Conclusion

In this case study we have tried to understand the working of the Air traffic control protocol called SATS. We showed the correctness of the landing protocol under assumptions of timed dynamics of the speeds of the aircraft. The correctness that we were interested in pertains to separation of the aircrafts and maintaing sequence. We were interested in proving these properties for arbitrarily large number of aircrafts in the system and hence followed a two-fold

approach. We first used a theorem proving tool PVS to mechanically prove that we need to consider a restriced state space. And then we used a model checking tool UPPAAL to prove the required properties of the continuous variables on the restricted state space.

References

1. G. Dowek, C. a. Muoz, and V. Carreo, "Abstract model of the sats concept of operations : Initial results and recommendations," *NASA Techinal Report - 2004*.
2. T. T. Johnson and S. Mitra, "Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study," *ICCPS*, pp. 161–170, Apr. 2012.
3. C. a. Muoz, G. Dowek, and V. Carreo, "Modeling and verification of an air traffic concept of operations," *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis - ISSTA '04*, pp. 175–175, 2004.

Appendix

.1 A PVS theory for SATS

```
SATS: THEORY BEGIN

Id : TYPE = nat
Zone : TYPE = upto(10) % Locations are numbered from 0 to 10
side: TYPE = { r , l }
Q : TYPE = [# loc:[Id -> Zone], m:[Id -> side] #]
IMPORTING finite_sets[Id]

non_fly(q:Q) : setof[Id] = { id:Id | loc(q)(id) /= 0 }
states: TYPE = { q:Q | is_finite(non_fly(q)) }

loc(i:Id, s:states): Zone = loc(s)(i)
m(i:Id, s:states): side = m(s)(i)

% Start State, all aircrafts are in flying zone
start(s: states) : bool = (FORALL (i:Id) : loc(i,s) = 0)

% Actions
actions: DATATYPE BEGIN
    fly_h3(i:Id, d:side, e:side): fly_h3?
    h3_h2(i:Id, d:side): h3_h2?
    h2_b(i:Id, d:side): h2_b?
    b_fin(i:Id, d:side): b_fin?
    fin_m(i:Id, d:side): fin_m?
    fin_run(i:Id): fin_run?
    m_h3(i:Id, d:side): m_h3?
```

```
    m_h2(i:Id, d:side): m_h2?
END actions

% Auxiliary functions for defining guards
occupied(z:Zone, s:states): bool = EXISTS (i:Id) : loc(i,s) = z % is a zone occupied
free(e:side, s:states): bool =
     FORALL (i:Id), (j:Id) : ((loc(i,s) /= 0)  and (loc(i,s) /= 10) and
                     (loc(j,s) /= 0) and (loc(j,s) /= 10) and (i /= j))
                   implies ((m(i,s) /= e) or (m(j,s) /= e))

% Guards
enabled(a:actions, s:states):bool =
CASES a OF
      fly_h3(i,d,e): (loc(i, s) = 0) and (d=r implies (not occupied(1, s))) and
          (d=l implies (not occupied(2, s))) and free(e,s),
      h3_h2(i,d) : (d=r implies (loc(i, s) = 1 and not occupied(3, s))) and
                  (d=l implies (loc(i, s) = 2 and not occupied(4, s))),
      h2_b(i,d)  : (d=r implies (loc(i, s) = 3)) and
                 (d=l implies (loc(i, s) = 4)),
      b_fin(i,d) : (d=r implies (loc(i, s) = 5)) and
          (d=l implies (loc(i, s) = 6)),
      fin_m(i,d) : (loc(i, s) = 9) and (d = m(i,s)),
      fin_run(i) : (loc(i, s) = 9),
      m_h3(i,d)  : (d=r implies (loc(i, s) = 7 and not occupied(1, s))) and
          (d=l implies (loc(i, s) = 8 and not occupied(2, s))),
      m_h2(i,d)  : (d=r implies (loc(i, s) = 7 and not (occupied(1, s)
                   or occupied(3, s)))) and (d=l implies (loc(i, s) = 8
                   and not (occupied(2, s) or occupied(4, s))))
ENDCASES

% Transitions
trans(a:actions, s:states):states =
   CASES a OF
      fly_h3(i,d,e): s WITH [loc := loc(s) WITH[(i) := (if d=r then 1 else 2 endif)],
                m := m(s) WITH[(i) := e]],
      h3_h2(i,d) : s WITH [loc := loc(s) WITH[(i) := (if d=r then 3 else 4 endif)]],
      h2_b(i,d)  : s WITH [loc := loc(s) WITH[(i) := (if d=r then 5 else 6 endif)]],
      b_fin(i,d) : s WITH [loc := loc(s) WITH[(i) := 9]],
      fin_m(i,d) : s WITH [loc := loc(s) WITH[(i) := (if d=r then 7 else 8 endif)]],
      fin_run(i) : s WITH [loc := loc(s) WITH[(i) := 10]],
      m_h3(i,d)  : s WITH [loc := loc(s) WITH[(i) := (if d=r then 1 else 2 endif)]],
      m_h2(i,d)  : s WITH [loc := loc(s) WITH[(i) := (if d=r then 3 else 4 endif)]]
   ENDCASES
```

```
IMPORTING simplemachine[states, actions, enabled, trans, start]

inSCA(i:Id, s:states): bool = (loc(i,s) /= 0) and (loc(i,s) /= 10)

% Atmost 2 aircrafts in zones other than fly and run which are assigned to a particular sid
atmost2(s:states) : bool = Forall (i,j,k:Id): (inSCA(i,s) and inSCA(j,s) and inSCA(k,s) and
                (m(i,s) = m(j,s)) and (m(j,s) = m(k,s)))
       implies
       ((i=j) or (j=k) or (i=k))

m_inv : LEMMA FORALL (i:Id, s:states, a:actions):
                   loc(i,s)/=0 and enabled(a,s) implies m(i, trans(a,s)) = m(i,s)

atmost2_inv : LEMMA FORALL (s:states, a:actions):
    reachable(s) and enabled(a,s) and atmost2(s) implies atmost2(trans(a,s))

END SATS
```