Parametrized Verification of Distributed Cyber-Physical Systems: An Aircraft Landing Protocol Case Study

Taylor T. Johnson and Sayan Mitra
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
Email: {johnso99, mitras} @illinois.edu

***Abstract*—In this paper, we present the formal modeling and automatic parameterized verification of a distributed air traffic control protocol called the Small Aircraft Transportation System (SATS). Each aircraft is modeled as a timed automaton with (possibly unbounded) counters. SATS is then described as the composition of N such aircraft, where N is a parameter from the natural numbers. We verify several safety properties for arbitrary N, the most important of which is separation assurance, which ensures that no two aircraft may ever collide. The verification methodology relies on computing the set of backward reachable states from the set of unsafe states to a fixed point, and checking emptiness of the intersection of these reachable states and the initial set of states. We used the Model Checker Modulo Theories (MCMT) tool, which implements this technique.**

I. Introduction

Many current and future cyber-physical systems (CPS)—such as in automotive and air traffic control protocols—involve a complex interaction between software state of many independent agents to ensure physical safety. We call such systems distributed cyber-physical systems (DCPS) due to this distributed interaction of cyber and physical state. This paper presents a parameterized model of a distributed air traffic control protocol and automatically verifies several non-trivial properties for arbitrarily many participating aircraft. The Small Aircraft Transportation System (SATS) was developed with the goal of increasing access to small airports that potentially do not have control towers nor radar. Instead, the aircraft rely on (a) receiving landing sequence information from an automated airport management module (AMM) located at the airport, and (b) communicating with one another to determine landing orders and perform landings. The overall operation must satisfy a variety of safety properties—such as, between each aircraft, there is always a sufficiently large physical separation.

Previous work on formally modeling and analyzing SATS has relied on using verification of purely discrete models [1], [2], [3] and hybrid models [4], [5], [6]. Parts of these works relied on deductive verification, such as through the use of the interactive theorem prover PVS [7], supplemented with some automatic state space exploration [4]. In [4], for example, it is first shown using PVS that SATS can have at most four approaching aircraft, and then all automatic state exploration uses this fixed number of aircraft. Traditional model checking would require the number of aircraft involved in the system to be fixed to a natural number prior to computing the composition. In contrast, we present a parameterized model of a simplified version of SATS and automatically verify the key safety property of the protocol *regardless of the number of aircraft in the system*. The techniques we use could help scale verification of automated highways, peer-to-peer protocols, and other general aviation systems, such as landing protocols for unmanned aerial vehicles (UAVs).

Parameterized Model Checking: Consider an automaton specification $\mathcal{A}_i$, where i is a unique identifier chosen from some set. The *parametrized model checking problem (PMCP)* is, given a state transition system $\mathcal{A}_i$ and a temporal formula ϕ, prove $\forall N \in \mathbb{N}$ that the composition $\mathcal{A}^N \triangleq \|_{i\in\{1,\ldots,N\}}\mathcal{A}_i$ satisfies ϕ [8, Ch. 15]. Such a formulation allows one to conclude—irrespective of the number of components involved—whether a system composed of N copies of $\mathcal{A}_i$ satisfies the property ϕ. The PMCP naturally captures verification problems with arbitrarily many participating components. For instance, in a realization of the automated highway system, how could one automatically verify that any interaction between an arbitrary number of cars does not result in a collision? A variety of other interacting DCPS are naturally modeled as parameterized systems, such as automotive traffic protocols [9], swarm robotics and coordination [10], [11], industrial systems [12], and networked medical devices.

A non-parameterized automatic verification could compose the system for increasing numbers of cars, say starting with two, then three, etc., but this process will never end. Eventually a state explosion barrier will be reached, which depends on the complexity of the model of each car and how they may interact (communications, sensing, etc.). If parameterized verification is possible, then it verifies all such instances at once. This enables scalable verification for such DCPS that are increasingly being designed, built, and used.

One could argue that in physical scenarios there are physical and geometric restrictions preventing an arbitrary number

of cars or aircraft from interacting, and that instead at most a fixed, finite natural number C may interact. While this is true, parameterized verification enables scalable verification of such systems, as it may be infeasible to perform the composition of C such $\mathcal{A}_i$ due to the growth in the size of the composed $\mathcal{A}^N$. In particular, we found that using the approach implemented in traditional model checkers like Uppaal, we could not scale beyond a few aircraft. For some problems, a few $\mathcal{A}_i$ may suffice for verification, but what if the number that can feasibly be verified is smaller than C?

The PMCP is an infinite-state model checking problem and is undecidable, even when $\mathcal{A}_i$ is a finite state automaton [13]. However, it has some decidable subclasses that we review in more detail in Section IV. To remain decidable, restrictions must be placed on $\mathcal{A}_i$, the parallel composition operator $\|$ (and hence how the $\mathcal{A}_i$'s may communicate), and the property ϕ. We also note that for general classes of hybrid systems, checking if even a single $\mathcal{A}_i$ satisfies ϕ is undecidable, but it also has several decidable subclasses [14], such as when restrictions are placed on the continuous dynamics—like in initialized rectangular hybrid automata (IRHA)—or on the discrete dynamics [15].

Contributions: After introducing the formal modeling framework used to discuss the problem, we automatically verify a simplified version of SATS, presented in Section III. We modeled SATS in a new parameterized model checking tool called the Model Checker Modulo Theories (MCMT) [16] by manually translating the PVS specification from [4] and applying some manual abstraction. In particular, the model from [4] uses queues to record the zone of SATS an aircraft physically resides in, while we use individual control locations for each aircraft. A queue is also used to track the arrival sequence of aircraft into the system, but instead, for each aircraft, we keep track of only the aircraft immediately ahead (if one exists).

The simplifications in our model are that we use timed ($\dot{x} = 1$) instead of rectangular ($\dot{x} \in [a, b]$) dynamics and did not model lateral entries. The results are that we were able to automatically verify several discrete state properties previously verified in a series of papers [1], [2], [5], as well as the key physical safety property, separation assurance [4], [6]. We believe that SATS can serve as a benchmark for the verification of DCPS. The protocol has a complex interaction of continuous and discrete dynamics. While both the physical and cyber dynamics are modeled in a relatively simple manner—clocks and counters—the combination of the two yields interesting properties.

II. Small Aircraft Transportation System

First we give an informal overview of the Small Aircraft Transportation System (SATS). We discuss related work on verifying other DCPS like air traffic control systems and automotive systems and give an overview of the relevant literature on the PMCP later in Section IV.

A. SATS Overview

In SATS, each aircraft i has a one-dimensional position evolving with time, representing its distance from approach to the runway. There is a single runway where the aircraft need to land (see Figure 1 for an overhead view of the landing area). There are two entry points to the runway, coming from the left or right of it. Each aircraft begins flying and may enter either the left or right cyclic holding patterns called *holding zones*. In this step, an aircraft is assigned a *sequence number*, which is the order in which the aircraft should land. While in the holding pattern, the aircraft are assumed to have a safe separation, and hence the values of the continuous positions do not matter. However, an aircraft may attempt an approach to the runway, at which point it exits the holding zone, begins on a path toward the runway, and the values of the continuous positions become significant. Upon attempting to land, the aircraft may either land on the runway and subsequently taxi away, or it may *miss the approach* and return to a cyclic holding zone.

Communication and Sensing Requirements: Next we present the aircraft and airport communication and sensing requirements that will lead into our model of the operation of SATS. The Airport Management Module (AMM) is a ground-based automation system that would typically be located at an airport and provides sequencing information to pilots over a ground-to-air datalink [17]. The AMM is the main centralized communication component of SATS, and all other communication is decentralized and done either (a) between pilots over voice radio, or (b) between aircraft control software via air-to-air datalinks.

Each aircraft in SATS is required to have the following sensing and communication capabilities [18]: (a) global positioning system (GPS) receiver, (b) air-to-ground datalink communication, for broadcast and receipt of AMM messages, (c) air-to-air datalink communication, (d) cockpit Display of Traffic Information, which provides a pilot the location of his/her aircraft and other nearby aircraft, (e) software to conduct the landing procedures, which informs a pilot who to follow and where to go by displaying sequencing information from AMM and uses Conflict Detection and Alerting Algorithms, and (f) voice communication radio.

A variety of desired properties are defined for SATS informally in the initial technical report describing the system [17]. The main safety property is *separation assurance*, that is, no two aircraft come closer than a pre-specified distance from one another, and hence never collide. There are also restrictions on the number of aircraft that may simultaneously be approaching the runway.

SATS Operation: We describe the protocol from the perspective of the i^{th} aircraft (refer to Figure 2 for states and refer to Subsection II-C for detailed definitions of the transitions). For aircraft i, q_i is its current mode. Aircraft i starts in the flying mode ($q_i = \mathbf{fly}$). It decides to land *nondeterministically* by entering the left or right holding

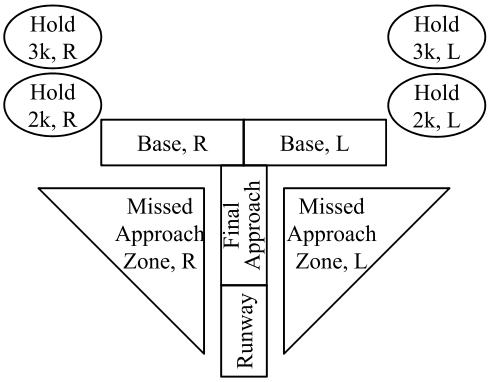

Figure 1. SATS viewed from above, except the holding zones at different elevations would be directly atop one another. There are two sides to approach the runway from, left (L) and right (R), and right and left are reversed in the image for the pilot's orientation. The safe spacing property matters when on the base, final, or missed zones, but not in other zones.

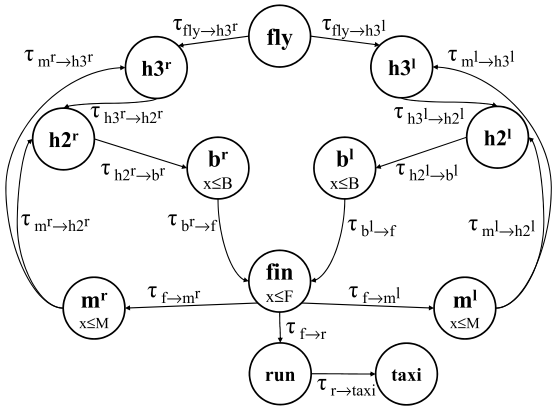

Figure 2. SATS Discrete Locations, Edges, and Invariants for an aircraft. Continuous variables x only matter in states $\mathbf{b^r}$, $\mathbf{b^l}$, $\mathbf{fin}$, $\mathbf{m^r}$, and $\mathbf{m^l}$, which correspond to when the aircraft is attempting to land on the runway by reaching location $\mathbf{run}$. Invariants for continuous variables in locations are captured by, for instance, $x \leq B$, etc., for some $B > 0$.

zone at 3000ft ($q_i \in \{\mathbf{h3^r}, \mathbf{h3^l}\}$). Upon entry, i is assigned a sequence number (0 if there are no other aircraft in the system, or $c+1$, where c is the value of the sequence number assigned to the last aircraft attempting to land). Subsequently, aircraft i *may* descend to the holding zone at 2000ft ($q_i \in \{\mathbf{h2^r}, \mathbf{h2^l}\}$). If there is enough spacing between i and the aircraft with sequence number one less than i's (if one exists), then i transitions to either the base left or right zone ($q_i \in \{\mathbf{b^r}, \mathbf{b^l}\}$). Aircraft i is never forced to transition from a holding pattern to an approach toward the runway. Rather, any aircraft *may* nondeterministically begin the approach, so long as the spacing condition is satisfied.

After traveling down the base zone for enough distance ($x_i \geq B$, where B is the length of the base zone), i moves to the final approach zone ($q_i = \mathbf{fin}$). Finally, after traversing the length of the final zone ($x_i \geq F$, where F is the length of the final zone), an aircraft *may* either: (i) land and go to the runway ($q_i = \mathbf{run}$), or (ii) miss its approach ($q_i \in \{\mathbf{m^r}, \mathbf{m^l}\}$). Then, after traversing the length of the missed zone ($x_i \geq M$, where M is the length of the missed zone), i restarts the process of moving from holding to final. If aircraft i misses its attempt to land, it is assigned a new sequence number at the end. Allowing aircraft to miss an approach is one reason that several of the properties to be introduced below are non-trivial to verify. The missed approach is initiated by the pilot if for any reason a safe landing cannot be assured (e.g., due to unforeseen weather changes, flying too fast to stop on the runway length, unknown obstacles on the runway, etc.). Also observe that the only locations where the continuous position x_i matters are the base, final, and missed zones.

B. Hybrid Automaton Model of an Aircraft

We first describe SATS using a hybrid automaton model for a single aircraft, and then compose $N \geq 2$ of the automata to yield the parameterized system. We write $[N] \triangleq \{1, \ldots, N\}$ and call it the *index set*. Our specification is a translation of [4] with the following two exceptions. First, we do not model lateral entry zones, where an aircraft could go from a lateral entry zone state to the holding zone at either 3000 or 2000ft, instead of having to start from the vertical entry to the 3000ft holding zone. Second, we use timed dynamics ($\dot{x} = 1$) instead of rectangular dynamics ($\dot{x} \in [a, b]$ for $a \leq b$). Modeling and verifying lateral entry zones would be relatively easy, but doing so for rectangular dynamics would require significant additional work.

Next we define some notation. For a set of variables V, and each variable $v \in V$ is associated with a type, which is the set of all the values v can take. A *valuation* or a *state* of a set of variables V is a function that maps each $v \in V$ to a value in its type, and is denoted by boldface $\mathbf{x}$, $\mathbf{v}$, etc. The set of all possible states for a set of variables V is denoted by $val(V)$. For a first-order logic formula ϕ, the valuation of the variables (appearing in ϕ) is written $[\![\phi]\!]$[1]. For some $T \in \mathbb{R}_{\geq 0}$, a T trajectory is a function $\gamma : [0, T] \to val(V)$, i.e., a trajectory maps intervals of time to states. The domain for a trajectory γ is denoted by $\gamma.\mathsf{dom}$. We define $\gamma.\mathsf{ltime}$ as the right endpoint of $\gamma.\mathsf{dom}$, $\gamma.\mathsf{lstate} \triangleq \gamma(\gamma.\mathsf{ltime})$, and $\gamma.\mathsf{fstate} \triangleq \gamma(0)$.

Next, we introduce the rest of the hybrid automata modeling framework in an on-the-fly fashion for our case study and refer the reader interested in additional technical details to [19], [20]. We are concerned with systems where a collection of hybrid automata evolve concurrently and communicate by reading each others' state variables. The hybrid automaton representing aircraft $i \in [N]$ is a tuple

[1]We are abusing notation—technically $[\![\phi]\!]$ is the denotation, or semantic value, of the formula ϕ. To relate valuations to denotations, we mean that $[\![\phi]\!]$ is the restriction of $[\![\phi]\!]$ on the set of variables V (which are not necessarily variables in the first-order logic sense appearing in ϕ). This is necessary as $[\![\phi]\!]$ would assign a value to each symbol appearing in ϕ, for instance, $N = 2$, $i = 1$, and $j = 2$.

$\mathcal{A}_i \triangleq \langle V_i, Q_i, \Theta_i, Edg_i, Grd_i, Rst_i, Flow_i, Inv_i \rangle$, where:

(A) V_i is a set of variables, and each variable $v \in V_i$ is associated with a type. Recall a valuation (or state) of a set of variables V_i is a function that maps each $v \in V_i$ to a value in its type, and is denoted by boldface $\mathbf{x}_i$, $\mathbf{v}_i$, etc. to indicate the dependence on i. A variable is either discrete or continuous. We write $X_i \subseteq V_i$ for the continuous variables. There are five variables: four discrete variables q_i, m_i, s_i, and c, and one continuous variable x_i. First is a finite control *location* q_i of type $Loc \triangleq \{\mathbf{fly}, \mathbf{h3^r}, \mathbf{h2^r}, \mathbf{h3^l}, \mathbf{h2^l}, \mathbf{b^r}, \mathbf{b^l}, \mathbf{fin}, \mathbf{m^r}, \mathbf{m^l}, \mathbf{run}, \mathbf{taxi}\}$, where each represent:
  (i) $\mathbf{fly}$: the aircraft is flying,
  (ii) $\mathbf{h3^r}, \mathbf{h2^r}, \mathbf{h3^l}, \mathbf{h2^l}$: the aircraft is in the holding zone on the right side at 3000 feet (2000 feet right side for $\mathbf{h2^r}$, and left for $\mathbf{h3^l}$ and $\mathbf{h2^l}$),
  (iii) $\mathbf{b^r}, \mathbf{b^l}$: attempting to land in the base segment on the right (left) side,
  (iv) $\mathbf{fin}$: on the final approach to the runway,
  (v) $\mathbf{m^r}, \mathbf{m^l}$: missed or aborted the landing attempt and on the right (left) side,
  (vi) $\mathbf{run}$: landed and on the runway, and
  (vii) $\mathbf{taxi}$: taxied off the runway to a gate.

  The next variable is the missed approach m_i of type $Side \triangleq \{left, right\}$ that indicates which side an aircraft will go to if it aborts its landing attempt. We abbreviate *left* as l and *right* as r when clear. The third variable is the sequence number s_i of type $\mathbb{N}$, which represents the sequence in which aircraft should land (where $s_i = 1$ would mean i should land first, and so on). The last discrete variable is a shared variable c of type $\mathbb{N}$ that is used as a counter tracking the last sequence number of an aircraft entering the system[2].

  There is a single continuous variable x_i of type $\mathbb{R}$ that represents the one-dimensional distance aircraft i has traveled from the physical location of a zone. In Figure 1, this is the distance measured along one dimension from the start of, to the end of, each of the base region, final approach region, and missed approach zones.

(B) $Q_i \triangleq val(V_i)$ is the set of states, which is the set of all valuations of the variables.

(C) Θ_i is the set of initial states and is $\{\mathbf{x}_i \in Q_i : q_i = fly \wedge x_i = 0 \wedge s_i = 0 \wedge c = 0\}$. The initial value of m_i is irrelevant, as it is updated before use.

(D) $Edg_i \subseteq Loc \times Loc$ is a set of *edges* corresponding to the discrete transitions of $\mathcal{A}_i$. There are discrete transitions between locations for SATS as shown in Figure 2.

(E) Rst_i is called a *reset function* and it associates with each edge a constant assignment for x_i. The resets are defined in detail in Subsection II-C.

(F) Grd_i is called a *guard function* and associates with each edge a predicate that defines the enabling condition for the transition. The guards are defined in detail in Subsection II-C. The guards for automaton $\mathcal{A}_i$ may contain constraints involving not only the local variables of $\mathcal{A}_i$, but also those of other automata.

  In our framework, these non-local guard constraints are the normal mode of communication between automata. In order to express such guards, we use *indexed constraints*, which are expressions with constraints over indexed state variables (e.g., q_i and x_j), such that all the indices are quantified and possibly constrained by comparison with one another (e.g., $i \neq j$). Some examples are,
  (i) $\forall\, i, j \in [N]\ (i \neq j \wedge x_i = c) \Rightarrow (x_j \neq c)$ and
  (ii) $(\forall\, j \in [N]\ (x_j \neq c)) \wedge (\exists\, j \in [N]\ (y_j = d))$.
  By UG_i, we refer to any universally quantified enabling condition corresponding to the edge, e.g., $x_j \neq c$ in the previous example(Fii).

(G) Inv_i is called an *invariant function* and associates each element of Loc a constraint (possibly empty) for x_i. The invariant function enforces a constraint on the value of x_i while q_i remains equal to a certain location. The non-empty invariants are shown for SATS in Figure 2 for the locations $\mathbf{b^r}$, $\mathbf{b^l}$, $\mathbf{fin}$, $\mathbf{m^r}$, and $\mathbf{m^l}$, and are used to model the physical lengths of the base, final, and missed zones.

(H) $Flow_i$ is called a *flow function* and associates each element of Loc a differential assignment for x_i. A *differential assignment for* x_i is an expression of the form: $\dot{x}_i := L$, where L is a constant. The variables that do not appear in a differential assignment are assumed to remain unchanged over the trajectory. In our model of SATS, $\dot{x}_i = 1$ in the locations $\mathbf{b^r}$, $\mathbf{b^l}$, $\mathbf{fin}$, $\mathbf{m^r}$, and $\mathbf{m^l}$, and $\dot{x}_i = 0$ in the other locations.

The parameterized system, denoted by $\mathcal{A}^N$, is defined as the composition of the automata $\mathcal{A}_1 \| \mathcal{A}_2 \| \ldots \| \mathcal{A}_N$, written in short as $\|_{i \in [N]} \mathcal{A}_i$. This composition is essentially the same as $N - 1$ parallel compositions of $\mathcal{A}_i$ considered in [19], [23]. Formally, $\mathcal{A}^N$ is a tuple $\langle V, Q, \Theta, Edg, Grd, Rst, Inv, Flow \rangle$, where:

(A) $V \triangleq \cup_{i \in [N]} V_i$ is the set of variables and $X = \cup \{x_i\} \subset V$ is the set of continuous variables (all the x_i's).

(B) $Q \triangleq val(V)$ is the set of states, that is, the valuations for the variables of each $i \in [N]$ in the composition.

(C) $\Theta \triangleq \wedge_{i \in [N]} \Theta_i$.

(D) $Edg \subseteq Loc^N \times Loc^N$. For every edge $e = (u, v)$ in Edg, there exists a unique $i \in [N]$ such that, for all other $j \in [N] \setminus \{i\}$, $u_j = v_j$ and $(u_i, v_i) \in Edg_i$. The unique component i for which the location changes is given by the function $trans : Edg \to [N]$, that is, $trans(e) = i$.

[2]We assume that all N automata have the same valuation of c, so we do not index it with a subscript. This fits within the timed network framework of [21], [22].

(E) Grd, defined as for each $e \in Edg$, $Grd(e) \triangleq Grd_i(e(i))$, where $i = trans(e)$.

(F) Rst is defined as, for any $(u, v) \in Edg$, $Rst((u, v)) \triangleq Rst_i((u_i, v_i))$, where $trans((u, v)) = i$. For all $j \in [N]$ where $j \neq i$, the variables remain unchanged.

(G) Inv is defined as, for any $a \in Loc^N$, $Inv(a) \triangleq \wedge_{\{i \in [N]: q_i = a_i\}} Inv_i(q_i)$, where a_i is the i^{th} component of the tuple $a \in Loc^N$. Note that this models whichever location each $i \in [N]$ is in. For instance, for the tuple $a = \langle \mathbf{fly}, \mathbf{h3^r}, \mathbf{run} \rangle$ and $\mathbf{x}.q = a$, at state $\mathbf{x}$, aircraft 1 is in $\mathbf{fly}$, 2 is in $\mathbf{h3^r}$, and 3 is in $\mathbf{run}$.

(H) $Flow$ is defined as, for any $a \in Loc^N$, $Flow(a) \triangleq \wedge_{\{i \in [N]: q_i = a_i\}} Flow_i(q_i)$, where a_i is again the i^{th} component of a.

To emphasize the dependence of these components on $[N]$, we may denote them by Θ^N, Q^N, Edg^N, Rst^N, etc.

The semantics of $\mathcal{A}^N$ is defined in terms of executions. An *execution fragment* of $\mathcal{A}^N$ is an alternating sequence $\alpha = \gamma_0, e_1, \gamma_1, \ldots$, where each γ_k is a trajectory for X and each e_k is an edge in Edg, such that the following conditions are satisfied:

(i) For each $i \in [N]$, for each $t \in \gamma_k.dom$, $\gamma_k(0).q_i = \gamma_k(t).q_i$, that is, the locations remain constant along trajectories. Here $\gamma_k.dom$ is the length of the k^{th} trajectory, and $\gamma_k(t)$ is a function mapping times t to Q, so $\gamma_k(0)$ is the first state fstate of the k^{th} trajectory and $\gamma_k(t)$ is the last state lstate.

(ii) For each $i \in [N]$, the derivative of x_i along γ_k satisfies the differential assignment $Flow_i(\gamma_k(0).q_i)$.

(iii) $\gamma_k.\mathsf{lstate}.X \in [\![Grd(e_{k+1})]\!]_I$, that is, the last state of each non-final trajectory satisfies the guard of the following edge. Recall that the notation $[\![\phi]\!]$ means the valuation of the variables (appearing in ϕ) satisfying the first-order logic formula ϕ.

(iv) For each $i \in [N]$, if $i \neq trans(e_{k+1})$ then $\gamma_{k+1}.\mathsf{fstate}.x_i = \gamma_k.\mathsf{lstate}.x_i$, otherwise $\gamma_{k+1}.\mathsf{fstate}.x_i \in [\![Rst(e_{k+1})]\!] = [\![R_i(e_i[k+1])]\!]$. That is, the first state of each non-initial trajectory satisfies the reset associated with the preceding edge. Since the edge e_{k+1} corresponds to a change in location for a unique $\mathcal{A}_i$, for which $i = trans(e_{k+1})$, this states that all the continuous variables in $X \setminus X_i$ remain constant over the transition, and the continuous variables in X_i are reset by $Rst_i(e_i)$.

An execution fragment is an execution, if in addition $\gamma_0.\mathsf{fstate} \in \Theta$. A state in Q is said to be reachable if there exists an execution terminating in it. The set of all reachable states of $\mathcal{A}^N$ is denoted by $\mathsf{Reach}_{\mathcal{A}^N}$, and if some state $\mathbf{x} \in \mathsf{Reach}_{\mathcal{A}^N}$, we say $\mathcal{A}^N$ reaches $\mathbf{x}$.

A *parametric safety property* ϕ is an indexed constraint. For example, the safety property $\forall i, j \in [N]\ i \neq j \wedge q_i = \mathbf{fin} \wedge q_j = \mathbf{fin} \Rightarrow x_i < b \wedge x_j > c$, asserts that for any two distinct aircraft that are in the same location $\mathbf{fin}$, there should be some minimal separation between the valuations of their continuous variables. Given a parametric safety property ϕ, the composition $\mathcal{A}^N$ is said to *satisfy* ϕ if, for all $N \in \mathbb{N}$, $\mathsf{Reach}_{\mathcal{A}^N} \subseteq [\![\phi]\!]$, which is written as $\mathcal{A}^N \models \phi$.

C. SATS Transitions and Trajectories

We now go through each of the transitions in the operation of SATS. The specification below is simplified for presentation (e.g., we use counters tracking the number of aircraft in certain zones), but the specification files showing these details are available on request. All aircraft begin flying, and may enter the system by transitioning from $\mathbf{fly}$ to the left or right holding zone at 3000ft. We drop the subscript i in all the following definitions of Grd, Rst, etc. for brevity, but they correspond to aircraft i. For aircraft i, for an edge from $\mathbf{fly}$ to $\mathbf{h3^r}$ (and symmetrically for $\mathbf{h3^l}$), we have:

$$
\begin{aligned}
Grd(\mathbf{fly}, \mathbf{h3^r}) &\triangleq q_i = \mathbf{fly},\\
Rst(\mathbf{fly}, \mathbf{h3^r}) &\triangleq q_i' = \mathbf{h3^r} \wedge s_i' = c + 1 \wedge c' = c + 1,\\
UG(\mathbf{fly}, \mathbf{h3^r}) &\triangleq q_j \neq \mathbf{h3^r} \wedge (j \neq i \Rightarrow q_j' = q_j \wedge s_j' = s_j).
\end{aligned}
$$

Note that we simplify notation and not write identity resets for any variable not appearing in a reset, although these must appear in the formulas (e.g., we dropped $m_j' = m_j$ and $x_j' = x_j$ from $UG(\mathbf{fly}, \mathbf{h3^r})$ and $x_i' = x_i$ from Rst). A first-order logic formula is obtained using quantifiers as: $\exists i \in [N].(Grd(\mathbf{fly}, \mathbf{h3^r}) \wedge Rst(\mathbf{fly}, \mathbf{h3^r}) \wedge \forall j \in [N].UG(\mathbf{fly}, \mathbf{h3^r}))$, where we observe that UG is a function of i. This is equivalent to the formula used by MCMT:

$$
\begin{aligned}
\tau_{\mathbf{fly} \to \mathbf{h3^l}} \triangleq \forall j \in [N].(q_j \neq \mathbf{h3^r}) \wedge \exists i \in [N].(q_i = \mathbf{fly} \wedge\\
q' = \lambda j.(\text{if } j = i \text{ then } \mathbf{h3^r} \text{ else } q_j) \wedge s' =\\
\lambda j.(\text{if } j = i \text{ then } c + 1 \text{ else } s_j) \wedge c' = c + 1),
\end{aligned}
$$

where the λj notation is equivalent to $\forall j$ and is used to ensure every component of (the vectors) q and s are defined.

We now describe the remaining transitions without going through this syntactic translation. Once in a (left or right) holding zone at 3000ft, an aircraft may descend to the holding zone on the same side at 2000ft. For the right side (and symmetrically for the left), this is described as:

$$
\begin{aligned}
Grd(\mathbf{h3^r}, \mathbf{h2^r}) &\triangleq q_i = \mathbf{h3^r}, Rst(\mathbf{h3^r}, \mathbf{h2^r}) \triangleq q_i' = \mathbf{h2^r},\\
UG(\mathbf{h3^r}, \mathbf{h2^r}) &\triangleq q_j \neq \mathbf{h2^r} \wedge (j \neq i \Rightarrow q_j' = q_j).
\end{aligned}
$$

Once in a (left or right) holding zone at 2000ft, an aircraft may begin the approach to the runway by transitioning to the base zone on the same side. For the right side (and symmetrically for the left), this is defined as:

$$
\begin{aligned}
Grd(\mathbf{h2^r}, \mathbf{b^r}) \triangleq q_i = \mathbf{h2^r} \wedge (s_i = 1 \vee (s_h = s_i - 1 \wedge\\
x_h - x_i \geq S)),\\
Rst(\mathbf{h2^r}, \mathbf{b^r}) \triangleq q_i' = \mathbf{b^r} \wedge x_i' = 0,\\
UG(\mathbf{h2^r}, \mathbf{b^r}) \triangleq (j \neq i \Rightarrow q_j' = q_j).
\end{aligned}
$$

In this transition, the identifiers i and h are both existentially quantified, but observe the variables of h are not reset.

Once an aircraft on the left or right base and on approach to the runway has traveled the length B of the base zone, it must enter the final approach zone. This must requirement makes this an urgent transition. The transition from the right (and symmetrically, left) is:

$$\begin{aligned} Grd(\mathbf{b^r}, \mathbf{fin}) &\triangleq q_i = \mathbf{b^r} \wedge x_i \geq B, \\ Rst(\mathbf{b^r}, \mathbf{fin}) &\triangleq q_i' = \mathbf{fin} \wedge x_i' = 0, \\ UG(\mathbf{b^r}, \mathbf{fin}) &\triangleq (j \neq i \Rightarrow q_j' = q_j \wedge x_j' = x_j). \end{aligned}$$

Once an aircraft on final approach travels the length F of the final approach zone, it must either miss the approach and enter the missed approach zone on the appropriate side, or it may land on the runway. Note that the side the aircraft misses to is defined by the value of the variable m_i, and is not necessarily the same side on which the aircraft initiated the approach to the final zone. The transition from the final approach to the right missed zone (and analogously, left) is:

$$\begin{aligned} Grd(\mathbf{fin}, \mathbf{m^r}) &\triangleq q_i = \mathbf{fin} \wedge x_i \geq F \wedge m_i = r, \\ Rst(\mathbf{fin}, \mathbf{m^r}) &\triangleq q_i' = \mathbf{m^r} \wedge x_i' = 0, \\ UG(\mathbf{fin}, \mathbf{m^r}) &\triangleq (j \neq i \Rightarrow q_j' = q_j \wedge x_j' = x_j). \end{aligned}$$

The transition from the final approach to the runway is:

$$\begin{aligned} Grd(\mathbf{fin}, \mathbf{run}) &\triangleq q_i = \mathbf{fin} \wedge x_i \geq F \wedge m_i = r, \\ Rst(\mathbf{fin}, \mathbf{run}) &\triangleq q_i' = \mathbf{m^r} \wedge x_i' = 0, \\ UG(\mathbf{fin}, \mathbf{run}) &\triangleq q_j \neq \mathbf{run} \wedge (j \neq i \Rightarrow q_j' = q_j \wedge x_j' = x_j). \end{aligned}$$

An aircraft on the runway may then taxi away, defined as:

$$\begin{aligned} Grd(\mathbf{run}, \mathbf{taxi}) &\triangleq q_i = \mathbf{run}, \\ Rst(\mathbf{run}, \mathbf{taxi}) &\triangleq q_i' = \mathbf{taxi}, \\ UG(\mathbf{run}, \mathbf{taxi}) &\triangleq (j \neq i \Rightarrow q_j' = q_j). \end{aligned}$$

After traveling the length M of the missed zone, an aircraft must transition to the lowest altitude holding zone without any aircraft in it on the same side as the missed zone. For the right missed approach zone (and symmetrically, left), this is defined by two transitions:

$$\begin{aligned} Grd(\mathbf{m^r}, \mathbf{h3^r}) &\triangleq q_i = \mathbf{m^r} \wedge x_i \geq M, \\ Rst(\mathbf{m^r}, \mathbf{h3^r}) &\triangleq q_i' = \mathbf{h3^r} \wedge x_i' = 0 \wedge s_i' = c, \\ UG(\mathbf{m^r}, \mathbf{h3^r}) &\triangleq q_j \neq \mathbf{h3^r} \wedge (j \neq i \Rightarrow \\ &\qquad q_j' = q_j \wedge x_j' = x_j \wedge s_j' = s_j - 1), \\ Grd(\mathbf{m^r}, \mathbf{h2^r}) &\triangleq q_i = \mathbf{m^r} \wedge x_i \geq M, \\ Rst(\mathbf{m^r}, \mathbf{h2^r}) &\triangleq q_i' = \mathbf{h2^r} \wedge x_i' = 0 \wedge s_i' = c, \\ UG(\mathbf{m^r}, \mathbf{h2^r}) &\triangleq q_j \neq \mathbf{h3^r} \wedge q_j \neq \mathbf{h2^r} \wedge (j \neq i \Rightarrow \\ &\qquad q_j' = q_j \wedge x_j' = x_j \wedge s_j' = s_j - 1). \end{aligned}$$

In the previous, we modeled the update values of m_i nondeterministically, which departs from the actual SATS specification. Also, observe that we use UG to decrease the value of the sequence numbers of the other aircraft.

Trajectories are defined according to the following first-order logic formula,

$$\begin{aligned} &\exists \epsilon \in \mathbb{R}_{\geq 0} \forall j \in [N]. T_j \wedge x_j' = x_j + \epsilon, \text{ where} \\ T_j \triangleq &(q_j \in \{\mathbf{b^r}, \mathbf{b^l}\} \Rightarrow x_j \leq B) \wedge (q_j = \mathbf{fin} \Rightarrow x_j \leq F) \\ &\wedge (q_j \in \{\mathbf{m^r}, \mathbf{m^l}\} \Rightarrow x_j \leq M). \end{aligned}$$

Here T_j captures urgency. This formula models that time elapses in the same amount ϵ for every aircraft, and thus their continuous positions evolve according to trajectories of the same length. Observe that this models many trajectories.

III. Verification of SATS

Given a set U of unsafe states, an automaton $\mathcal{A}$ is said to be safe with respect to U if it is never reached by $\mathcal{A}$. Sometimes the complement of the unsafe set U is specified as a formula involving the variables of $\mathcal{A}$ and it is called a safety property P. For SATS, one such safety property is separation assurance—that is, no two aircraft ever come too close together—which can be written as:

$$P \triangleq \forall i, j \in [N]. i \neq j \wedge s_i = s_j - 1 \Rightarrow x_i - x_j \geq S,$$

where $s_i = s_j - 1$ indicates that aircraft i is ahead of aircraft j in the landing sequence, and $S > 0$ is the minimum separation desired between aircraft i and j.

The general methodology for establishing that an automaton $\mathcal{A}$ is safe with respect to a property P is to show that the set P is invariant, that is, P contains all the reachable states of the system. Alternatively, one can define the unsafe property as the negation $\neg P$, which for separation assurance would assert that there are two aircraft too close together. Then one can prove safety by showing that the set of executions leading to this bad set of states cannot begin in the set of initial states. Thus, to prove a safety property automatically, it suffices to take the negation of the safety property, and ensure the set of backward reachable states have an empty intersection with the initial set of states. This is the method used by the verification tool we used, the Model Checker Modulo Theories (MCMT) [24], [16], [22], [25]. If the intersection of the backward reachable states and the initial states is empty and the backward reachability process terminates—that is, the backward reachability computation reaches a fixed point and no new states are added on a preimage computation—then the system is proven safe.

Under the assumption that the desired safety property is an indexed constraint, the preimage computation from the set of unsafe states is not much different than that for a non-parameterized system. For instance, consider the negation of the separation assurance property, which states there are two

$k := 0$
$\phi_k := \phi_B$, for $\phi_B \equiv \neg\phi_S$
$\rho_k := \phi_k$
$\sigma_k := \emptyset$

while $true$
  if $\rho_k \wedge \phi_I$ *satisfiable* // safety check
    return *unsafe* $\wedge \sigma_k$ *counterexample*
  $k := k + 1$
  for each $\chi \in \tau(\mathbf{x}, \mathbf{x}')$
    $\phi_k := \phi_k \cup Pre_\chi(\phi_{k-1}) \equiv \phi_k \cup \exists \mathbf{x}'.\, (\tau_\chi(\mathbf{x}, \mathbf{x}') \wedge \phi_{k-1}(\mathbf{x}'))$
    $\sigma_k(\chi) := \sigma_k(\chi) \cup \tau_\chi$ // keep tree of valid executions
  end
  $\rho_k := \rho_{k-1} \vee \phi_k$

  if $\neg\ (\rho_k \implies \rho_{k-1})$ *unsatisfiable* // fixed point check
    return *safe*
end

Figure 3. Basic backward reachability algorithm used by MCMT.

aircraft less than the safety distance apart,

$$\neg P \triangleq \exists i, j \in [N].i \neq j \wedge s_i = s_j - 1 \wedge x_i - x_j < S.$$

Observe that $\neg P$ is defined in terms of two aircraft being in a particular state. The preimage computation will return a formula with the same existential quantifiers[3]. For instance, the preimage of the formula $\exists i \in [N].q_i = \mathbf{run}$ is roughly—we omit some details to present the intuition—the formula $\exists i \in [N].q_i = \mathbf{fin}$, since the only way for an aircraft to reach the runway is from the final approach zone $\mathbf{fin}$. Observe that this does not increase the number of quantified index variables appearing in the formula—that is, the preimage of $\exists i \in [N].Q(i)$ is not of the form $\exists i, j \in [N].Q(i, j)$.

The main complication is ensuring that the sequence of preimage computations terminates. A detailed theory of when this preimage computation will terminate has been developed for parameterized systems [27], [26], and parameterized timed systems [21], [28]. Our formulation of SATS is undecidable—that is, a fixed point may not be reached—for two main reasons. First, we model urgent trajectories, that is, we prevent trajectories from continuing once some condition becomes true [21]. Second, we use universally quantified index variables in some transition guards [29].

Now, why should we expect this process to ever reach a fixed point? For instance, why is it not possible for new aircraft to continually enter the system? For example, observe that the unsafe property is of the form $\exists i \in [N].P(i)$. With this form of property, all that matters is whether there is some aircraft in the system satisfying $P(i)$. Again, observe that the preimage computation will return a formula like $\exists i \in [N].Q(i)$. It is essential for termination that the preimage computation does not always add existentially quantified index variables to the formula. If the preimage is $\exists i \in [N].Q(i)$, where $Q(i)$ corresponds to a formula not implied by $P(i)$ (or any of the formulas corresponding to already reached states), then we cannot terminate, but otherwise we can. Likewise, if the property is of the form $\exists i, j \in [N].P(i, j)$, then all that matters is whether there are two processes satisfying the formula, etc.

MCMT takes four inputs: the initial set of states, the unsafe set of states, the transition rules for one automaton $\mathcal{A}_i$, and some auxiliary axioms that hold for the parameterized system $\mathcal{A}^N$ (which are useful for asserting data type constraints and already proved safety properties). These inputs are essentially specified as formulas in a restricted subclass of first-order logic. For example, a safety property of SATS is that there is never more than a single aircraft in the left holding zone at 3000ft, $\mathbf{h3^l}$, and hence the unsafe states are those where there are two or more aircraft in $\mathbf{h3^l}$. More formally, the parametric safety property is $\forall i, j \in [N].i \neq j \Rightarrow (q_i \neq \mathbf{h3^l} \vee q_j \neq \mathbf{h3^l}$, and the parametric unsafe property is $\exists i, j \in [N].i \neq j \wedge q_i = \mathbf{h3^l} \wedge q_j = \mathbf{h3^l}$. The initial set of states are those where all aircraft are flying and have not yet entered the approach to the runway, $\forall i.q_i = \mathbf{fly} \wedge x_i = 0 \wedge s_i = \perp$. The representation of the transition rules and trajectories as first-order logic formulas was presented previously in Subsection II-C.

Next we describe how this procedure is implemented by MCMT in the algorithm shown in Figure 3. The parameterized system $\mathcal{A}^N$ evolves according to the set of transition rules $\tau(\mathbf{x}, \mathbf{x}')$, where the pre-state is $\mathbf{x}$ and post-state is $\mathbf{x}'$. The state $\mathbf{x}$ can be thought of as a vector of length $N \geq 2$, with corresponding state values $\mathbf{x}.x_i$ for say the continuous variable x_i of aircraft i. We previously showed how to syntactically convert the Grd and Rst functions for SATS to the τ transition rules in Subsection II-C. The entire transition relation $\tau(\mathbf{x}, \mathbf{x}')$ is defined as the disjunction of all transitions (e.g., $\tau_{\mathbf{fly}\to\mathbf{h3^r}} \cup \tau_{\mathbf{fly}\to\mathbf{h3^l}} \cup \ldots$) from Subsection II-C. The algorithm implemented in MCMT processes first-order logic formulas that describe sets of states. For backwards reachability, the first formula describes a bad (that is, unsafe or illegal) set of states, denoted by ϕ_B. For each $N \geq 2$, consider the composition $\mathcal{A}^N$, then we have $\mathbf{x}_B \equiv [\![\phi_B]\!]$.

The system will be safe if the algorithm reaches a fixed point and the constraints describing that set of states which may reach $\mathbf{x}_B$ do not intersect with the initial set of states, described by ϕ_I. Let $\mathbf{x}_I \equiv [\![\phi_I]\!]$ for any $N \geq 2$. Let ρ_k be the sequence of formulas starting from ϕ_B. Defined inductively, $\rho_0 \triangleq \phi_B$ and $\rho_k \triangleq \rho_{k-1} \vee Pre(\phi_{k-1})$. Pre is the preimage of a formula, defined as $Pre_\chi(\phi_{k-1}) \equiv \exists \mathbf{x}'.\, (\tau_\chi(\mathbf{x}, \mathbf{x}') \wedge \phi_{k-1}(\mathbf{x}'))$ for each transition $\chi \in \tau$. Thus, ρ_k represents the set of states that can reach the bad set of states in k iterations of the algorithm.

It is desired that a fixed point is eventually reached, that is, so that $\rho_k \equiv \rho_{k-1}$. The problem is in general undecidable, so it may be the case that no fixed point is reached. To check if a fixed point has been reached, one checks if $[\![\rho_k]\!] \subseteq [\![\rho_{k-1}]\!]$. This is equivalent to checking satisfiability of $\neg(\rho_k \implies \rho_{k-1})$. Conditions for decidability of the safety and fixed

[3]In general this is not true, but see [26] for when it is.

point checks are given in [26], as are conditions for when a fixed point is guaranteed to exist.

A. Properties Verified

SATS has previously been manually verified, and we specify and verify several of the same safety properties [4], [5]. We leave out a couple regarding lateral entry that we are not modeling. The initial states are specified as the first-order formula $\phi_I \triangleq \forall i. q_i = \mathbf{fly} \wedge m_i \in \{left, right\} \wedge x_i = 0 \wedge s_i = 0 \wedge c = 0$. Observe that all of the properties verified are in essence mutual exclusion properties. Some properties state that are no more than a single aircraft in a state, while others specify no more than two are in a state, etc. Separation assurance can be viewed as a sort of physical mutual exclusion property.

We describe all properties as parametric (or indexed) safety properties.

(A) There are no more than four aircraft attempting to land, that is, the total number of aircraft in any states besides flying and landed is 4 (but there may be arbitrarily many aircraft flying or landed). Let $T = \{\mathbf{fly}, \mathbf{taxi}\}$, and let $F = Loc \setminus T$ be the set of discrete locations without the flying or taxi states, then the property is specified as:

$$\phi_S \triangleq \forall i, j, k, l, m. (i \neq j \neq k \neq l \neq m \wedge q_i \in F \wedge q_j \in F \wedge q_k \in F \wedge q_l \in F) \implies q_m \in T.$$

In the implementation, we actually use counters to track this property, e.g., we count the number of aircraft in the system and verify this counter is bounded from above by 4. We note that the SATS specification allows only 4 aircraft to be on approach at a given time [17], but the number of aircraft involved in the protocol could potentially be expanded by adding additional "sides" with corresponding holding, base, and missed zones (e.g., SATS is designed to have only left and right sides, but one can imagine a system with more entry points).

(B) The main property we are interested in is separation assurance, that two aircraft on approach to the runway are separated by a safety spacing $S > 0$. Let $F = \{\mathbf{b^r}, \mathbf{b^l}, \mathbf{fin}, \mathbf{m^r}, \mathbf{m^l}\}$, and the property is:

$$\phi_S \triangleq \forall i, j. (i \neq j \wedge q_i \in F \wedge q_j \in F \wedge s_i = s_j - 1) \Rightarrow x_i - x_j \geq S.$$

(C) No more than two aircraft are actually on either side (left or right). Let T^l be all the locations on the left side, $T^l = \{\mathbf{h3^l}, \mathbf{h2^l}, \mathbf{b^l}, \mathbf{m^l}\}$ and T^r be all the locations on the right side, and let $F^r = Loc \setminus T^l$ and $F^l = Loc \setminus T^r$ be the other states. The property for the left side (and symmetrically, right) is:

$$\phi_S \triangleq \forall i, j, k. (i \neq j \neq k \wedge q_i \in T^l \wedge q_j \in T^l) \Rightarrow q_k \in F^r.$$

(D) At most one aircraft is in each of the holding zones, for $\mathbf{h2^r}$ (and defined analogously for $\mathbf{h3^r}$, $\mathbf{h3^l}$, and $\mathbf{h2^l}$) this is:

$$\phi_S \triangleq \forall i, j. (i \neq j \wedge q_i = \mathbf{h2^r}) \Rightarrow q_j \neq \mathbf{h2^r}.$$

We actually could not verify the property for $\mathbf{h3^l}$ or $\mathbf{h3^r}$ due to a spurious execution from the stopping failures abstraction [16], so we assumed these two cases, and were able to establish the property for the others.

(E) No more than two aircraft are on a missed approach fix, for the right (and defined analogously for left), this is:

$$\phi_S \triangleq \forall i, j, k. (i \neq j \neq k \wedge q_i = \mathbf{m^r} \wedge q_j = \mathbf{m^r}) \Rightarrow q_k \neq \mathbf{m^r}.$$

Additionally, there is a liveness property proven in [4], but unfortunately, MCMT cannot verify liveness properties, only safety ones. The property would state that all aircraft eventually land and that they land in order according to their sequence numbers. Some very recent work attempts to allow verification of some classes of liveness properties [25], but general liveness properties for parameterized timed systems were shown to be undecidable in [21].

We used version 1.1.1 of MCMT and version 1.0.32 of Yices for the verification. MCMT has some capability to generate invariants, and we enabled the full invariant search for our verification (we used the options `-I` and `-S3`). All runtimes of verification attempts are reported in Table I, and were measured on a modern laptop with 8GB main memory and an Intel Core i7 quad-core processor running at 2.0GHz. However, we ran the verification in a virtual machine under Ubuntu, limited to the use of two cores and 1.5GB memory. We used the `memtime` utility from Uppaal [30] to measure runtimes and memory usage. We used an existing model of the system in Uppaal to verify some properties in a non-parameterized version for $N = 2, 3, 4, 5$ aircraft and to help us understand the protocol. We do not report verification runtimes for Uppaal, as we primarily used Uppaal as a simulation tool prior to encoding the SATS protocol in MCMT, but we could not verify beyond five aircraft in the system.

The order in which the properties are proved is important, as our process was first to attempt proving a property, and if it was established as an invariant, we would assume it as a lemma and continue the verification process. We used a Python script to automatically call MCMT and assert lemmas. Finally, we note that we also performed some sanity checks to see if certain states are reachable (e.g., there are 3 aircraft in the system, although this could be spurious due to the stopping failures abstraction).

Property	Runtime (s)	Memory Usage (MB)
A	25.95	10.19
B	283.08	32.49
C	24.50	5.80
D	0.81	4.56
E	491.61	274.44

Table I
RUNTIME IN SECONDS AND MAXIMUM MEMORY USAGE IN MEGABYTES REQUIRED TO VERIFY PROPERTIES OF SATS IN MCMT.

IV. RELATED WORK

Verification of safety critical traffic protocols like those seen in automotive and aerospace systems has been done for quite some time. For instance, techniques based on optimal control are used for verification of conflict resolution maneuvers in [31], [32] and automatic landing systems in [33]. Curved flight maneuvers have also been verified in [34]. Automotive protocols like those that may play a role in the automated highway system have been modeled and verified semi-automatically [9].

With regard to SATS, discrete abstractions capturing all behaviors of SATS are created in [2], [5]. The properties verified in these works include that there are at most four aircraft on the approach to the runway, and similar properties limiting the number of aircraft in certain zones of SATS. In [4], assuming this limited number of aircraft in the system, the authors automatically generated a set of lemmas corresponding to every combination of aircraft, and then discharged these lemmas semi-automatically, thus verifying the separation assurance safety property of the hybrid system. However, a more detailed hybrid systems model of SATS is developed in [6].

Parameterized Verification: There is a large amount of related work on automating the parameterized verification problem. The book [8, Ch. 15] includes an overview, and a nice survey is [35]. Some of the earliest works on parameterized verification appeared in [36], [37]. The verification of such compositions is also known as *uniform verification* [35]. We stated in the introduction that this problem is in general undecidable [13]. However, for restricted classes of systems under various communications constraints, the problem has been shown to be decidable.

For instance, the PMCP is decidable for safety properties of the networks of timed automata considered in [21], [22], where each automaton has a single real-valued clock. If the timed automata have more than a single real-valued clock, then checking safety properties is undecidable [28]. However, if the clocks are discrete-valued, each automaton may have any finite number of clocks. If the timed automata have urgent transitions, then checking safety properties is undecidable [21]. While checking general liveness properties is undecidable for these networks [21], some recent work develops methods for checking some liveness properties [25].

An alternative approach for parameterized verification uses interactive theorem proving. The system model and the properties are specified as a theory in the language of the theorem prover, and then these properties are discharged by invoking theorem prover commands on the proof goals. The granularity of these commands and the degree of automation varies from one system to another, but proving sophisticated invariant properties requires significant manual work. This approach has been successfully applied to verify (a) timed automata [38], [39] in PVS [7], (b) SATS [4], [5], [6] in PVS, (c) Fischer's mutual exclusion protocol [40] in SAL, (d) aircraft separation assurance in conflict avoidance maneuvers [34] in KeYmaera [41], [42], (e) automotive collision avoidance in adaptive cruise control [9] in KeYmaera, (f) and many other systems and tools. Despite these techniques using automated theorem provers being partially manual, we believe the strengths of deductive methods are that (a) they can handle nonlinear continuous dynamics and complex discrete dynamics with data structures, and (b) they could be used to specify and verify liveness properties.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we automatically verified several safety properties of a distributed air traffic landing protocol, regardless of the number of aircraft involved in that protocol. The air traffic protocol is a nontrivial distributed cyber-physical system (DCPS), and for this reason, we believe it could serve as a standard benchmark in the verification of DCPS. We believe other DCPS under development, like networks of autonomous vehicles or medical devices, can and should be verified using this approach to increase the assurance of reliability that the complex interaction of software and physical processes does not yield catastrophic failure for some instantiations of the system. While we were successful in the verification in this paper—in part because there exists a cutoff on the number of aircraft in the system as shown through the verification approach used in [4]—we are interested in investigating abstractions for solving this problem. For instance, the environment abstraction approach [43] tracks the number of processes satisfying some predicates, perhaps even abstracting the continuous variables in this way, and may provide a more tractable approach for some classes of DCPS.

ACKNOWLEDGMENTS

This paper is based upon work supported by the National Science Foundation under CAREER Grant No. 1054247. The authors wish to thank Cesar Muñoz for providing a PVS specification of SATS, Karthik Manamcheri for developing the Uppaal model, and helpful discussions with Madhusudan Parthasarathy. The authors also wish to thank their paper shepherd, Sung-Soo Lim, and the anonymous referees, for their helpful feedback.

REFERENCES

[1] C. Muñoz, G. Dowek, and V. Carreño, "Modeling and verification of an air traffic concept of operations," *Software Engineering Notes*, vol. 29, no. 4, pp. 175–182, 2004.

[2] 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.

[3] V. Carreño and C. Muñoz, "Safety verification of the Small Aircraft Transportation System concept of operations," in *Proceedings of the AIAA 5th Aviation, Technology, Integration, and Operations Conference, AIAA-2005-7423*, Arlington, Virginia, 2005.

[4] C. Muñoz, V. Carreño, and G. Dowek, "Formal analysis of the operational concept for the small aircraft transportation system," in *Rigorous Development of Complex Fault-Tolerant Systems*, ser. LNCS, M. Butler, C. Jones, A. Romanovsky, and E. Troubitsyna, Eds. Springer Berlin / Heidelberg, 2006, vol. 4157, pp. 306–325.

[5] S. Umeno and N. Lynch, "Proving safety properties of an aircraft landing protocol using i/o automata and the pvs theorem prover: A case study," in *Formal Methods*, ser. LNCS, J. Misra, T. Nipkow, and E. Sekerinski, Eds. Springer, 2006, vol. 4085, pp. 64–80.

[6] ——, "Safety verification of an aircraft landing protocol: A refinement approach," in *Hybrid Systems: Computation and Control*, ser. LNCS. Springer, 2007, vol. 4416, pp. 557–572.

[7] S. Owre, J. M. Rushby, , and N. Shankar, "PVS: A prototype verification system," in *11th International Conference on Automated Deduction (CADE)*, ser. LNAI, D. Kapur, Ed., vol. 607. Saratoga, NY: Springer-Verlag, jun 1992, pp. 748–752.

[8] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[9] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *Formal Methods*, ser. LNCS, M. Butler and W. Schulte, Eds. Springer, 2011.

[10] S. Gilbert, N. Lynch, S. Mitra, and T. Nolte, "Self-stabilizing robot formations over unreliable networks," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, pp. 1–17, Jul. 2009.

[11] T. T. Johnson and S. Mitra, "Safe flocking in spite of actuator faults using directional failure detectors," *Journal of Nonlinear Systems and Applications*, vol. 2, no. 1-2, pp. 73–95, Apr. 2011.

[12] T. T. Johnson, S. Mitra, and K. Manamcheri, "Safe and stabilizing distributed cellular flows," in *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS 2010)*, Genoa, Italy, June 2010.

[13] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Inf. Process. Lett.*, vol. 22, no. 6, pp. 307–309, 1986.

[14] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, pp. 94–124, 1998.

[15] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 13, pp. 1–21, 2000.

[16] S. Ghilardi and S. Ranise, "MCMT: A model checker modulo theories," in *Automated Reasoning*, ser. LNCS. Springer, 2010, vol. 6173, pp. 22–29.

[17] T. S. Abbott, K. M. Jones, M. C. Consiglio, D. M. Williams, and C. A. Adams, "Small aircraft transportation system, higher volume operations concept: Normal operations," Tech. Rep. NASA/TM-2004-213022, Aug. 2004.

[18] S. Viken and F. Brooks, "Demonstration of four operating capabilities to enable a small aircraft transportation system," in *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 2, Oct. 2005.

[19] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[20] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.

[21] P. A. Abdulla and B. Jonsson, "Model checking of systems with many identical timed processes," *Theoretical Computer Science*, vol. 290, no. 1, pp. 241–264, 2003.

[22] A. Carioni, S. Ghilardi, and S. Ranise, "MCMT in the land of parameterized timed automata," in *In Proc. of VERIFY 2010*, Jul. 2010.

[23] T. A. Henzinger, "The theory of hybrid automata," in *IEEE Symposium on Logic in Computer Science (LICS)*. Washington, DC, USA: IEEE Computer Society, 1996, p. 278.

[24] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli, "Towards smt model checking of array-based systems," in *Automated Reasoning*, ser. LNCS. Springer, 2008, vol. 5195, pp. 67–82.

[25] R. Bruttomesso, A. Carioni, S. Ghilardi, and S. Ranise, "Automated analysis of parametric timing based mutual exclusion protocols," in *Proc. of 4th NASA Formal Methods Symposium (NFM)*. Springer, 2012.

[26] S. Ghilardi and S. Ranise, "Backward reachability of array-based systems by smt solving: Termination and invariant synthesis," *Logical Methods in Computer Science*, vol. 6, no. 4, 2010.

[27] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!" *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63–92, 2001.

[28] P. A. Abdulla, J. Deneux, and P. Mahata, "Multi-clock timed networks," in *Proc. of 19th Annual IEEE Symposium Logic in Computer Science*, Jul. 2004, pp. 345–354.

[29] P. Abdulla, G. Delzanno, and A. Rezine, "Parameterized verification of infinite-state processes with global conditions," in *Computer Aided Verification*, ser. LNCS, W. Damm and H. Hermanns, Eds. Springer, 2007, vol. 4590, pp. 145–157.

[30] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal: a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. Lecture Notes in Computer Science, R. Alur, T. Henzinger, and E. Sontag, Eds. Springer Berlin / Heidelberg, 1996, vol. 1066, pp. 232–243.

[31] C. Tomlin, G. Pappas, and S. Sastry, "Conflict resolution for air traffic management: a study in multiagent hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 509–521, Apr. 1998.

[32] C. Tomlin, I. Mitchell, and R. Ghosh, "Safety verification of conflict resolution manoeuvres," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 2, no. 2, pp. 110–120, June 2001.

[33] A. Bayen, I. M. Mitchell, M. M. K. Oishi, and C. J. Tomlin, "Aircraft autolander safety analysis through optimal control-based reach set computation," *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, vol. 30, no. 1, Jan. 2007.

[34] A. Platzer and E. Clarke, "Formal verification of curved flight collision avoidance maneuvers: A case study," in *Formal Methods*, ser. LNCS, A. Cavalcanti and D. Dams, Eds. Springer, 2009, vol. 5850, pp. 547–562.

[35] L. Zuck and A. Pnueli, "Model checking and abstraction to the aid of parameterized systems," *Computer Languages, Systems & Structures*, vol. 30, no. 3-4, pp. 139–169, 2004.

[36] M. C. Browne, E. M. Clarke, and O. Grumberg, "Reasoning about networks with many identical finite state processes," *Information and Computation*, vol. 81, no. 1, pp. 13–31, 1989.

[37] S. M. German and A. P. Sistla, "Reasoning about systems with many processes," *J. ACM*, vol. 39, no. 3, pp. 675–735, 1992.

[38] H. Lim, D. Kaynar, N. Lynch, and S. Mitra, "Translating timed i/o automata specifications for theorem proving in pvs," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS, P. Pettersson and W. Yi, Eds. Springer, 2005, vol. 3829, pp. 17–31.

[39] M. Archer, H. Lim, N. Lynch, S. Mitra, and S. Umeno, "Specifying and proving properties of timed i/o automata using tempo," *Design Automation for Embedded Systems*, vol. 12, pp. 139–170, 2008.

[40] B. Dutertre and M. Sorea, "Timed systems in sal," SRI International, Tech. Rep. SRI-SDL-04-03, Oct. 2004.

[41] A. Platzer, "Quantified differential dynamic logic for distributed hybrid systems," in *Computer Science Logic*, ser. LNCS, vol. 6247, 2010, pp. 469–483.

[42] ——, "Quantified differential invariants," in *Proc. of the 14th ACM Intl. Conf. on Hybrid Systems: Computation and Control*. ACM, 2011, pp. 63–72.

[43] E. Clarke, M. Talupur, and H. Veith, "Environment abstraction for parameterized verification," in *Verification, Model Checking, and Abstract Interpretation*, ser. LNCS, E. Emerson and K. Namjoshi, Eds. Springer, 2006, vol. 3855, pp. 126–141.