Safety Verification of an Aircraft Landing Protocol: A Refinement Approach*

Shinya Umeno and Nancy Lynch

CSAIL, Massachusetts Institute of Technology, Cambridge MA, USA
{umeno,lynch}@theory.csail.mit.edu

Abstract. In this paper, we propose a new approach for formal verification of hybrid systems. To do so, we present a new refinement proof technique, *a weak refinement using step invariants*. As a case study of the approach, we conduct formal verification of the safety properties of NASA's *Small Aircraft Transportation System* (SATS) landing protocol. A new model is presented using the *timed I/O automata* (TIOA) framework [1], and key safety properties are verified. Using the new refinement technique presented in the paper, we first carry over the safety verification results from the previous discrete model studied in [2] to the new model. We also present properties specific to the new model, such as lower bounds on the spacing of aircraft in specific areas of the airspace.

1 Introduction

Hybrid systems are complex. In order to obtain a manageable mathematical model of a real hybrid system, a certain level of abstraction needs to be taken. A high-level abstraction of a system gives us a *discrete state-transition model*, where timing-dependent and continuous behavior of a real system are abstractly represented as discrete transitions. This high-level abstraction is particularly useful for a system that has algorithmically complex behavior. For instance, in [3], the initial start-up algorithm for the Time-Triggered Architecture [4] is formally verified using such a high-level abstraction. An important question here is whether the properties proved for the discrete abstraction hold for a real system, or for a refined, more realistic model.

In this paper, we propose a new approach to formally verify a given hybrid system. Basic concept of this approach is to use two levels of abstraction to verify a given hybrid system. The low-level *continuous model* includes descriptions of timing-dependent and continuous behavior, whereas in the high-level *discrete model*, timing-dependent and continuous behavior are abstracted away. Verification for these two models is done in the following steps.

1. First, the formal verification of the discrete model is conducted. This can be done either by the *invariant-proof technique*, or a *model-checking*.
2. Next, to carry over verification results from the discrete model to the continuous model, we prove a *refinement* from the continuous model to the discrete model.

* This project is supported by Air Force Contract FA9550-04-C-0084.

A. Bemporad, A. Bicchi, and G. Buttazzo (Eds.): HSCC 2007, LNCS 4416, pp. 557–572, 2007.

3. Finally, by using invariants carried over from the discrete model, we prove safety properties in the continuous model. Some of these properties immediately follow from the invariants carried over. On the other hand, some other properties can be expressed only in the continuous model, since they involve time-dependent or continuous behavior.

We technically contribute to the second step in the above stated approach. We often need some invariants of both the discrete model and the continuous model to prove a refinement. To make use of the invariants of the discrete model to larger extent than the existing techniques, we introduce a new refinement technique, called a *weak refinement using step invariants*. This technique differs from the existing techniques in that, by using this, we can use *invariants of the discrete model* in order to prove *invariants of the continuous model* needed for a refinement proof. Since we can assert the fact that invariants of the discrete model also hold for the continuous model *only after* proving a refinement between them, using the existing techniques causes a circular reasoning. Our new technique, a *weak refinement using step invariants*, resolves this problem.

As a case study of an application of the newly presented approach and refinement technique, we conduct a safety verification of the aircraft landing protocol that is part of NASA's *Small Aircraft Transportation System* (SATS) concept of operation [5]. Some formal verification studies for this protocol have been conducted so far. In [6], Dowek, Muñoz, and Carreño presented a state-transition model of the protocol. This model was a discrete model in that the airspace of airport is divided into several logical zones, and movements of aircraft are represented as discrete transitions Using this discrete model, safety verification of the model was done in [6], using a model-checking. The safety properties the authors model-checked were key upper bounds on the number of aircraft in the specific divided zones. In [7], Muñoz and Dowek extended their previous work [6] by presenting a *hybrid model* of the protocol, in which aircraft in a specific portion of the airspace of the airport exhibit continuous behavior, but movements of aircraft in the remaining portion are still discretized. Using this model, in [7], the authors verified key spacing properties of aircraft in the continuous portion of the hybrid model, using *symbolic model-checking* technique. We previously presented in [2] *invariant-proof-style verification* of the discrete model presented in [6]. In doing so, we first re-constructed the discrete model of [6] using an untimed *I/O automata* (IOA) framework, and verified key safety properties model-checked in [6] by using the invariant-proof technique. The proof for this case study has been mechanically checked using the PVS theorem prover [8].

In this paper, we present a new model of the protocol, $ContSATS$, which represents the continuous model in our new approach for this case study. This model more realistically reflects the dynamics of aircraft movement in a real system than the previous models presented in [6] and [7]. In contrast to the previous models, our new model captures continuous movements of aircraft in the entire airspace of the airport. The model is constructed using the *timed I/O automata* (TIOA) framework [1]. This framework and the *hybrid I/O automata* (HIOA)

framework [9][1] have been used successfully to model several hybrid systems, such as a helicopter controller [10], the Traffic Alert and Collision Avoidance System [11] and a Lego car [12]. We first carry over the result from the discrete model to $ContSATS$ by proving a refinement, and then prove key spacing properties of aircraft in $ContSATS$, which can be expressed by $ContSATS$, but not by the discrete model.

This paper is organized as follows: In Section 2, we briefly explain the TIOA framework, and introduce a new refinement technique. In Section 3, we quickly review the discrete model of [6], and present key invariants of the model proved in [2]. In Section 4, we introduce the new model $ContSATS$. Section 5 is devoted to proving a refinement from $ContSATS$ to the discrete model. In Section 6, we present lower bounds on the spacing of aircraft in $ContSATS$. These are obtained by using the results carried over from the discrete model by a refinement. Finally, in Section 7, we summarize the results, and discuss some future work.

2 Timed I/O Automata Framework

In this section, we explain some basics of the timed I/O automata (TIOA) framework [1]. In Section 2.2, we introduce a new refinement technique, a *weak refinement using step invariants*. We also present a theorem that states that this refinement from automaton A to B implies that invariants of B also hold in A in some specific sense.

2.1 Timed I/O Automata

A timed I/O automaton (TIOA) is a state transition machine with an extension of continuous behavior. Every discrete transition is defined in a precondition-effect style, and continuous behavior is defined using *trajectories*. A trajectory is a partial function from a time to the current values of the state components of the automaton. The domain of a trajectory must be some interval in the time domain, and the size of the domain represents the duration that elapses by that trajectory. A trajectory can be a *point trajectory*, whose domain is a point $[t, t]$, for some time t. The trajectories of an automata are specified by the **evolve** and the **stop when** statement in the trajectory definition. In the **evolve** statement, we state the rate of the value change of a real-time variable x by differential equations or inequalities in terms of $\mathsf{d}(x)$, the first derivative of x. Informally, the **stop when** statement specifies the time when we want the model to perform some discrete transition. An *execution fragment* of an automata is a (possibly infinite) alternating sequence of trajectories and discrete transitions $\tau_0 a_1 \tau_1 a_2 \tau_2 ...$ that satisfies the following three conditions: 1). Each trajectory satisfies the constraints defined by the **evolve** and the **stop when** statements;

[1] The latest version of the TIOA framework presented in [1] is the restricted version of the HIOA framework in that external analog variables cannot be used for TIOA. Since we do not have any analog variables in automata for our case study, the two frameworks are intrinsically same in this study.

2). a_{i+1} is enabled in $\tau_i.lstate$ (the last state of trajectory τ_i); 3). a_{i+1} represents the transition from $\tau_i.lstate$ to $\tau_{i+1}.fstate$ (the first state of trajectory τ_{i+1}). We call an execution fragment an *execution* if it starts with one of the designated *start states*. We say that state s is a reachable state if there is an execution α such that $\alpha.lstate = s$. Informally, the *trace* of an execution is the externally visible part of the execution. More formally, it is the alternating sequence of the duration that elapses by the trajectory and the *external* transitions, such that each duration matches up the duration of the corresponding trajectory in the execution, and all *internal* transitions are hidden.

Let A be a TIOA. Q_A denotes the set of the states of A. Θ_A denotes the set of the start states of A. $reachable(A)$ denotes the set of the reachable states of A. $traces_A$ denotes the set of the the traces of A. An *invariant* of automaton A is a predicate over Q_A that is satisfied for any $s \in reachable(A)$. A *step of A starting with state s* is an execution fragment of A starting with s that consists of either one discrete transition surrounded by two point trajectories, or one closed trajectory with no discrete transition.

2.2 Weak Refinement Using Step Invariants

A *refinement* is a proof technique that has been used to show trace inclusion between two automata A and B ($traces_A \subseteq traces_B$). Informally, the above stated trace inclusion tells us that the external behavior of A does not go beyond what we expect from B. In some cases, we want to use invariants of automata in a proof of a refinement. A *weak refinement*[2] has been used for such cases. These refinement techniques, and simulation relations (more general version of refinements) are well studies in the computer science community, and several kinds of such simulation techniques for TIOA are summarized in [13].

In some cases (as we will see in Section 5), we actually need *invariants of B* in order to prove some *invariants of A* needed in the proof of a refinement from A to B. Since we can assert the fact that invariants of B also hold for A *only after* proving a refinement from A to B, we end up with circular reasoning if we use an existing refinement technique. This is why we need our new technique, a *weak refinement using step invariants*. Informally, our solution to this problem is to prove only the inductive case of the invariant proof for such invariants of A, assuming some additional conditions. In the following, we present a new definition of invariants that captures the above informal discussion.

Definition 1. Let A be a TIOA. Let P_1 and P_2 be predicates over Q_A. We say that P_1 is a *step invariant of A using P_2*, or simply *a step invariant using P_2* when A is obvious from the context, if, for any reachable state s of A and any step α of A starting with s, the following condition holds.

$$P_1(\alpha.fstate) \wedge P_2(\alpha.fstate) \Rightarrow P_1(\alpha.lstate)$$

[2] This usage of the term "weak" here comes from [13]. We use this term since we have more assumptions (namely, invariants of automata) in some conditions of the definition of this refinement, than an ordinary refinement.

That is, to show that P_1 is a step invariant using P_2, we prove only the step condition of the invariant proof for P_1, assuming the additional condition P_2. The following lemma easily follows from the definition of a step invariant.

Lemma 2. *$P_1 \wedge P_2 \wedge ... \wedge P_n$ is a step invariant for automaton A using condition Q if P_1 is a step invariant of A using Q, and P_i, $2 \leq i \leq n$, is a step invariant of A using $Q \wedge P_1 \wedge ... \wedge P_{i-1}$.*

Now we define the new refinement. The main difference from the definition of an ordinary weak refinement[3] is that we assume an additional predicate P^* over Q_A in the step condition (Conditions 2) of the refinement. This P^* must be a step invariant using $\lambda s.P_B(r(s))$[4], where P_B is an invariant of B. This captures the above informal discussion: since we need invariant P_B of B in order to prove that P^* is an invariant of A, we just require P^* to be a step invariant using $\lambda s.P_B(r(s))$, invariant P_B "adapted" to A using mapping r.

Definition 3. Let A and B be TIOA. Let P_A be an invariant of A, and P_B be an invariant of B. Let r be a partial function from Q_A to Q_B. Let P^* be a step invariant of A using $\lambda s.P_B(r(s))$.

We say that r is a *weak refinement using P_A, P_B, and P^** if it satisfies the following two conditions for all states x_A and x_B of A and B, respectively.

1. If $x_A \in \Theta_A$ then $x_A \in dom(r)$, $r(x_A) \in \Theta_B$, and $P^*(x_A)$ hold.
2. If α is a step of A, and $\alpha.fstate \in dom(r)$, and

$$P_A(\alpha.fstate) \wedge P_B(r(\alpha.fstate)) \wedge P^*(\alpha.fstate)$$

holds, then $\alpha.lstate \in dom(r)$ and B has a closed execution fragment β with $\beta.fstate = r(\alpha.fstate)$, $trace(\beta) = trace(\alpha)$, and $\beta.lstate = r(\alpha.lstate)$.

We can prove the following soundness theorem for this new refinement technique. A proof appears in the full version of this paper [14].

Theorem 4. *Let A and B be TIOA and r be a weak refinement from A to B, using P_A, P_B, and P^*. Then $traces_A \subseteq traces_B$.*

The existence of a refinement from A to B actually implies more than just trace inclusion. Due to space limitation, we cannot present general theorems about this close correspondence (they appear in [14]). Here we present one theorem regarding invariants of automata.

Theorem 5. *Let A and B be TIOA. Let r be a refinement, a weak refinement, or a weak refinement using step invariants, from A to B. Let P_B be an invariant of B. Then, the predicate $\lambda s.P_B(r(s))$ is an invariant of A.*

[3] Due to space limitation, we cannot give the definition of an ordinary refinement or that of a weak refinement in this paper. The definition appears in the full version of this paper [14].

[4] $\lambda s.P_B(r(s))$ is the function that, given $s_1 \in Q_A$, returns $P_B(r(s_1))$.

Theorem 5 is used in Section 5 to carry over the invariants of the discrete model that have been proved in [2] to our new continuous model presented in Section 4.

Related works: The new refinement introduced in this section has a flavor of *assume-guarantee* reasoning, which has also been applied to hybrid systems [15,16]. Assume-guarantee reasoning is used for compositional verification of a system. When we verify a composed system $S_1||S_2$, instead of verifying S_1 and S_2 separately, we sometimes want to assume some properties of the system to be composed with. For example, to prove that S_1 works correctly, we may have to assume that S_2 "well behaves" in some particular sense. Assume-guarantee techniques allows us to have deduction rules that if S_1 is correct assuming S_2 well behaves and S_2 is correct assuming S_1 well behaves, then, the composed final system $S_1||S_2$ is correct. In contrast to the existing assume-guarantee techniques, with our new technique, we can assume that the high-level abstraction behaves correctly in order to prove that the low-level abstraction has invariants needed to prove the refinement. To our best knowledge, we have not seen any other technique that uses assume-guarantee reasoning in the above sense.

3 Discrete Model

A discrete state-transition model of the SATS landing protocol is presented in [6]. In this model, the airspace of the airport is discretized, and every movement of the aircraft is represented as a transition of the model. In [2], we reconstructed the model using the I/O automata framework. Due to space limitation, we cannot present a formal description of the discrete model. However, we present a formal description of our new model in Section 4, and also discuss differences between the discrete model and the new model in the same section.

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

Logical zones: In the discrete model, the airspace of the airport is logically divided into 13 zones (see Fig. 1). Each zone is modeled as a first-in first-out queue of aircraft. A movement of aircraft is represented by moving an aircraft from the head of one queue to the end of another queue. We refer to the T-shaped area consists of base(right), base(left), intermediate, and final as *the approach area*. This area is where aircraft perform the final approach to the ground.

Landing sequence: When an aircraft enters the system, the system assigns its *leader* aircraft, or the aircraft it has to follow. This leader relation is used in the protocol as a guard that delays the aircraft's final approach initiation for safe landings: an aircraft cannot enter the approach area until its leader has done so. In our discrete model, we encode this notion of the leader aircraft as an explicit queue of aircraft, called the *landing sequence*. When an aircraft enters the logical zones, it is also added to the end of the landing sequence, and is removed when it finishes landing. We define the leader of aircraft a as the aircraft just in front of a in the landing sequence.

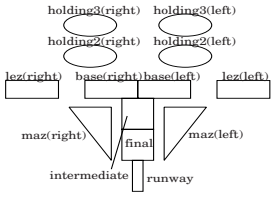


Fig. 1. 13 logical zones in the discrete model

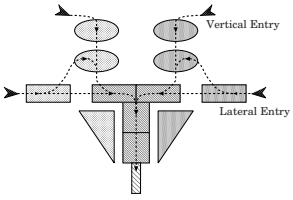


Fig. 2. Paths of aircraft

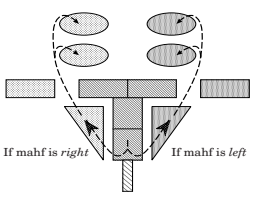


Fig. 3. Paths of aircraft that have missed the approach

Paths of aircraft: Here we present a high level picture of aircraft movements in the logical zones. All movements are represented by transitions, which are described in the precondition-effect style. A transition moves one aircraft from one zone to another in a way that it satisfies the rules specified in the protocol. The paths of aircraft are depicted in Fig. 2. An aircraft may miss the approach to the ground at the final zone. In such a case, it goes back to a holding fix (either holding3 or holding2), and makes the next try to land. An aircraft needs to determine the side of the holding fixes to which it goes in case it misses the approach. For this purpose, the assignment of the side, called the *missed approach holding fix (mahf)* is given to an aircraft when it enters the system. These paths of missed aircraft are depicted in Fig. 3.

Properties: In [6], some interesting properties of the discrete model that express safe separation of aircraft are presented and are exhaustively checked using an exhaustive exploration technique. In [2], using the invariant-proof technique, we proved key safety properties presented in [6]. Here we review some of the properties proved in [2]. The following condition Φ is defined as the conjunction of the listed seven conditions. An auxiliary predicate on_approach_qn(σ) checks if there is some aircraft assigned σ as its mahf in the approach area. In the rest of the paper, we refer to the first condition of Φ by Φ.1, the second condition by Φ.2, and so on. In Section 5.1, we present auxiliary invariants of the new model that is needed to prove a refinement as a step invariant using this Φ. It is worth to note here that, Conditions 3, 4, and 5 cannot be derived from the main safety properties taken from [6], but are derived from auxiliary lemmas to prove the main properties. Since we need these three conditions in Φ to prove a refinement in Section 5.2, this indicates that, by proving these auxiliary invariants, the assertional-style techniques give us more insight to how the system works, than an exhaustive exploration.

Condition Φ

1. $\forall \sigma : \text{side},\ \text{length}(\mathsf{holding3}(\sigma)) \leq 1 \wedge \text{length}(\mathsf{holding2}(\sigma)) \leq 1$
2. $\forall \sigma : \text{side},\ \neg\text{empty\_qn}(\mathsf{lez}(\sigma)) \Rightarrow$
   $\text{empty\_qn}(\mathsf{holding2}(\sigma)) \wedge \text{empty\_qn}(\mathsf{holding3}(\sigma)) \wedge \text{empty\_qn}(\mathsf{maz}(\sigma))$
3. $\text{first}(\mathsf{final}) = \text{first}(\mathsf{landing\_seq})$
4. $\forall \sigma : \text{side},\ (\text{on\_approach\_qn}(\sigma) \wedge \neg\text{empty\_qn}(\mathsf{maz}(\sigma))) \Rightarrow \text{empty\_qn}(\mathsf{holding3}(\sigma))$

5. $\forall \sigma : \text{side}, \text{on\_approach\_qn}(\sigma) \Rightarrow \ \text{length}(\mathsf{holding2}(\sigma)) + \text{length}(\mathsf{holding3}(\sigma)) \leq 1$
6. $\forall \sigma : \text{side}, \mathit{length}(\mathsf{maz}(\sigma)) \geq 2 \Rightarrow \text{empty\_qn}(\mathsf{holding2}(\sigma)) \wedge \text{empty\_qn}(\mathsf{holding3}(\sigma))$
7. $\forall \sigma : \text{side}, \neg\text{empty\_qn}(\mathsf{maz}(\sigma))) \Rightarrow \ \text{length}(\mathsf{holding2}(\sigma)) + \text{length}(\mathsf{holding3}(\sigma)) \leq 1$

4 Our New Continuous Model

In this section, we present our new continuous model, *ContSATS*, which more realistically reflects the dynamics of the aircraft movement in a real system than the discrete model or the *hybrid model* presented in [7]. In the hybrid model of [7], the movement of the aircraft in the approach area and the missed approach zones is modeled as continuous behavior. These areas are modeled as abstract lines[5] representing paths of aircraft on which aircraft continuously move according to their velocity vectors. Now a discrete transition for aircraft in the approach area and the maz zones is performed when an aircraft reaches the intersection points of the lines, in order to reassign the line on which that aircraft move.

To describe continuous dynamics of aircraft in the entire airspace of the airport, we use the same strategy as used for the hybrid model of [7]: in *ContSATS*, we model the paths of aircraft predetermined by the protocol as a collection of lines, with aircraft moving on them according to their velocity. (see Fig. 4, and compare it with Fig. 2 and 3). In the new model, analogous to the hybrid model of [7], we use transitions to re-assign the line on which aircraft move. The pre-determined paths in *ContSATS* include holding points (holding3hold and holding2hold in Fig. 4), where aircraft hover until the condition for the next procedure (transition) is satisfied.

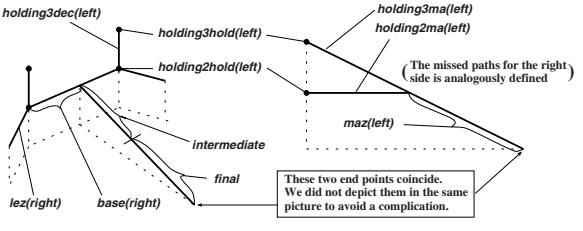


Fig. 4. Our new continuous model: *ContSATS*

4.1 Formal Specification for *ContSATS*

In this subsection, we present formal code for *ContSATS*, written in the TIOA specification language [17]. We explain auxiliary constants and functions first.

The line on which a specific aircraft currently moves is specified by a new attribute of aircraft, line. We use the prefix "LINE_" for the line names; for example, the final zone as a line is represented as LINE_final. The position of

[5] These lines forms "trajectories" of aircraft flying on the pre-determined paths. However, we avoid using the term "trajectories", and instead use "lines", since the term is also used in the TIOA framework and thus two usages may confuse the reader.

a specific aircraft in the line is specified by another new attribute of aircraft, pos. Using both the line value and pos value of a particular, we can uniquely determine on which line, and at what position in that line that aircraft is now.

Another new attribute of aircraft is t, which is used to express a time bound for some specific transitions to be performed. When one of the designated transitions becomes enabled, the aircraft a corresponding to that transition (the aircraft that will move by the transition) has its t value set to the current value of now. By the **stop when** clause in the trajectory definition, $ContSATS$ is guaranteed to fire the transition corresponding to aircraft a either before or at the time the value of $\mathsf{now} - a.\mathsf{t}$ reaches the pre-determined time bound for that transition. T_3, T_2, and $\mathsf{T}_{\mathsf{Tax}}$ represents the time bounds for StartDescending, VerticalApproachInitiation, and Taxiing, respectively. We use function T that maps the name of a zone to the above specified time bounds for aircraft in that zone. We set t of aircraft outside of the holding zones or of the runway to -1, indicating that a timer is not set for those aircraft.

For simplicity, we assume that the lines are exactly symmetric on the right and left sides of the airport. $\mathsf{L}_{\mathsf{3dec}}$, $\mathsf{L}_{\mathsf{3ma}}$, L_{B}, L_{I}, L_{F}, and L_{M} respectively represents the lengths of holding3dec, holding3ma, base, intermediate, final, and maz. We use the function L to represent the length of the line for a given line. We denote by L_{T} the length aircraft fly in the entire approach area, that is, $\mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + \mathsf{L}_{\mathsf{F}}$. We use the function D to represent the distance a specific aircraft has flown in the approach area, and then in the missed approach zone; for example, if aircraft a is in final, $\mathsf{D}(a) = \mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + a.\mathsf{pos}$, and if a is in $\mathsf{maz}(\sigma)$, $\mathsf{D}(a) = \mathsf{L}_{\mathsf{B}} + \mathsf{L}_{\mathsf{I}} + + \mathsf{L}_{\mathsf{F}} + a.\mathsf{pos}$. If an aircraft is not in the approach area nor in the missed approach zones, the D function returns 0.

The velocity of the aircraft is bounded by some constants. This constraint is specified in the **evolve** statement in the trajectory definition.

We present formal code for $ContSATS$ in the following. Due to space limitation, we only show the definitions of three transitions (VerticalApproachInitiation, MissedApproach, and LowestAvailableAltitude), and the trajectory definition. The full specification appears in [14]. The above three transitions are chosen because of the following three reasons. 1: VerticalApproachInitiation is one of the most interesting transitions, which represents an initiation of the aircraft's final approach to the ground. The precondition of the transition represents the guard so that an aircraft cannot initiate its approach until its leader has done so and the separation between the aircraft and its leader becomes at least S_0. 2: MissedApproach is also an interesting transition, which represents missed approaches of aircraft. As we can see from the precondition, this transition is preformed nondeterministically whenever an aircraft reaches the end point of the final line ($a.\mathsf{pos} = \mathsf{L}_{\mathsf{F}}$). 3: In addition to the extra structure needed to represent the continuous behavior (such as now, pos, or the trajectory definition), we also modified three transitions inherited from the discrete model, in order to more faithfully represent a real system (how we modified them is explained in [14]). These are LowestAvailableAltitude, Landing, and HoldingPatternDescend. Due to this modification, we need some nontrivial auxiliary invariants of $ContSATS$ to prove

a refinement from $ContSATS$ to the discrete model. As we will see in Section 5.1, these invariants are proved as step invariants using Φ (Corollary 7). In this paper, we focus on LowestAvailableAltitude among the three transitions.

We use three effects set_pos, set_line, and set_t to re-assign the pos, line, and t attributes of aircraft, respectively. The code of the automaton imports a *vocabulary*, ContSatsVocab, where auxiliary functions used in $ContSATS$ are defined. We do not have a space to explain all these functions (it appears in [14]), but will explain those we need for the lemma statement and the proof. leader(a, landing_seq) represents the leader of aircraft a in the landing sequence. The predicate on_approach_qn(a) where a is an aircraft checks if a is in the approach area. The predicate on_approach_qn(σ) where σ is a side checks if there is some aircraft assigned to σ as its mahf in the approach area. The predicate on_zone_qn(z, a) checks if aircraft a is in zone z.

---

automaton $ContSATS$
imports ContSatsVocab

%% All original discrete transitions are considered as the output transitions.
%% We added four new internal transitions, as well as the trajectory definition.
signature
output
VerticalEntry(ac:Aircraft, id:ID, $side$:Side), LateralEntry(ac:Aircraft, id:ID, $side$:Side),
HoldingPatternDescend(ac:Aircraft,$side$:Side), VerticalApproachInitiation(ac:Aircraft,$side$:Side),
LateralApproachInitiation(ac:Aircraft,$side$:Side), Merging(ac:Aircraft,$side$:Side),
Exit(ac:Aircraft), FinalSegment(ac:Aircraft), Landing(ac:Aircraft), Taxiing(ac:Aircraft),
MissedApproach(ac:Aircraft), LowestAvailableAltitude(ac:Aircraft,$side$:Side),
internal
StartHolding2(ac:Aircraft,$side$:Side), StartHolding3(ac:Aircraft,$side$:Side),
StartDescending(ac:Aircraft,$side$:Side), SetTime
states
zones : zone_map, % mapping from a zone name to a zone
nextmahf : Side, % Next missed approach holding fix
landing_seq : Zone % landing sequence is defined as a queue
now : AugumentedReal % the time elapsed from the initial state
initially
zones = initialZones ∧ nextmahf = right ∧ landing_seq = empty ∧ now = 0

%% Definitions of auxiliary functions are not shown in this code due to space limitation.

transitions

output VerticalEntry($a, id, side$)

output LateralEntry($a, id, side$)

internal StartDescending($a, side$)

output HoldingPatternDescend($a, side$)

output VerticalApproachInitiation($a, side$)
pre ¬(empty_qn(holding2($side$))) ∧
a = first(holding2($side$)) ∧
length(base(opposite($side$))) ≤ 1 ∧
(first_in_seq_qn(a) ∨
(on_approach_qn(leader(a,landing_seq)) ∧
D(leader(a,landing_seq)) ≥ S_0))
eff set_line(a, AC_base($side$)); set_pos(a, 0);
set_t(a, -1);
zones := move(holding2($side$),base($side$),zones)

output LateralApproachInitiation($a, side$)

internal SetTime

output Merging($a, side$)

output Exit(a)

output FinalSegment(a)

output Landing(a)

output Taxiing(a)

output MissedApproach(a)
pre ¬(empty_qn(final)) ∧ ¬(empty_qn(landing_seq))
∧ a = first(final) ∧ a.pos = L_F
eff set_line(a, AC_maz(a.mahf)); set_pos(a, 0)
zones:= assign(zones, final, rest(final));
zones:= assign(zones, maz(a.mahf),
add(maz(a.mahf),reassign(a)));
landing_seq := add(rest(landing_seq),reassign(a));
nextmahf := opposite(reassign(a).mahf);

output LowestAvailableAltitude(a, $side$)
pre ¬(empty_qn(maz($side$))) ∧
a = first(maz($side$)) ∧ a.pos = $\mathsf{L_M}$;
eff IF empty_qn(holding3($side$)) ∧
empty_qn(holding2($side$))
THEN set_line(a, AC_holding2ma($side$));
set_pos(a,0);
zones := move(maz($side$),holding2($side$),zones);
ELSE set_line(a, AC_holding3ma($side$));
set_pos(a,0)
zones := move(maz($side$),holding3($side$),zones);
FI

internal StartHolding3(a, $side$)

internal StartHolding2(a, $side$)

trajectories
stop when
(∃ a:Aircraft,
($\exists$ z:Zone, on_zone_qn(z, a)) ∧
a.pos ≥ L(a.line))
∨ (∃ a:Aircraft,
($\exists$ z:Zone, on_zone_qn(z, a)) ∧
a.t ≠ −1 ∧ now − a.t ≥ T(a.line))
∨ (∃ a:Aircraft,
($\exists$ z:Zone, on_zone_qn(z, a)) ∧
((a.line = holding2L ∨ a.line = holding2R)∧
a.t = −1 ∧ ¬first_in_seq_qn(a) ∧
on_approach_qn(leader(a,landing_seq)) ∧
D(leader(a,landing_seq)) = $\mathsf{S_0}$))
evolve
d(now) = 1
∀ a: Aircraft
IF (a.line=holding3decL ∨ a.line=holding3decR)
THEN ($\mathsf{V_{d\_min}} \le d(a.\mathsf{pos}) \le \mathsf{V_{d\_max}}$)
ELSE ($\mathsf{V_{min}} \le d(a.\mathsf{pos}) \le \mathsf{V_{max}}$) FI

In order to obtain a refinement, we have to assume the following condition: $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_3dec}}{V_{d\_min}})\mathsf{V_{max}} < \mathsf{L_T} + \mathsf{L_M}$. This is used in the refinement proof in the case of the LowestAvailableAltitude transition.

5 Carrying over the Results from the Discrete Model Using a Refinement

In [2], we formally verified the safe separation of aircraft in the discrete model, by proving bounds on the number of aircraft in the logical zones. If we can carry over these results to $ContSATS$, the properties carried over tell us important spacing properties in $ContSATS$. For example, from the property that there is at most one aircraft in one holding3(σ), we can guarantee that two aircraft would never get close in the holding3 line in $ContSATS$. On the other hand, we cannot guarantee spacing properties of two aircraft on two adjacent lines from the properties of the discrete model. Some of these properties are actually proved as auxiliary lemmas for the refinement. We also examine several spacing properties in Section 6.

To make the discrete model (an ordinary IOA) comparable to $ContSATS$ (a TIOA), we first construct $ExtSATS$, a natural extension of the discrete model to a TIOA. This extension can be done in the following generic way: Given an ordinary IOA A, we construct A' that is an timed extension (TIOA version) of A. First, in A', we add a new now state component to A which evolves at rate 1 (d(now) = 1). There is no **stop when** statement for A', and all discrete transitions are exactly the same as before the extension. From this straightforward extension, it is easy to see that all invariants of A are also invariants of A'. From Theorem 5, if we prove a refinement from $ContSATS$ to $ExtSATS$, any invariant of $ExtSATS$ is guaranteed to be an invariant of $ContSATS$.

One straightforward refinement mapping to consider (and actually the one we use for the refinement proof) is the following mapping r from a state of $ContSATS$ to a state of $ExtSATS$: for all $s \in Q_{ContSATS}$, $r(s) = t$ such that

$zones\_equal(s.\mathsf{zones}, t.\mathsf{zones}) \ \wedge \ s.\mathsf{nextmahf} = t.\mathsf{nextmahf} \ \wedge$
$queue\_equal(s.\mathsf{landing\_seq}, t.\mathsf{landing\_seq}) \ \wedge \ t.\mathsf{now} = s.\mathsf{now},$

where $zones\_equal$ and $queue\_equal$ represent the equalities for two zone maps and two aircraft queues, respectively, defined by ignoring the new attributes of aircraft in $ContSATS$, such as pos (formal definitions appear in [14]). This mapping r maps a state of $ContSATS$ to a state of $ExtSATS$ so that every component of the state of $ContSATS$ matches the corresponding component of the state of $ExtSATS$. Note that such a state $r(s)$ in $ExtSATS$ is uniquely determined for every state s of $ContSATS$, since the above conditions specify all components of $ExtSATS$.

It turns out that we have to use a weak refinement using step invariants introduced in Section 2.2 for this mapping r. This is because in order to prove some invariants of $ContSATS$ needed to prove a refinement, we actually need some invariants of $ExtSATS$ that have been verified.

5.1 Auxiliary Invariants

In this subsection, we present the auxiliary invariants needed for the refinement proof. Due to space limitation, we cannot present a proof for these auxiliary invariants (it appears in [14]). We use Condition Φ defined in Section 3 as a state proposition of $ContSATS$.

Lemma 6. *Consider the following conditions A_1, A_2, B, C_1, and C_2.*

$(\mathbf{A_1}) : \forall a, b : \mathrm{Aircraft}, \forall \sigma : \mathrm{side}, \ \mathrm{on\_approach\_qn}(a) \wedge$
$a.\mathsf{mahf} = \sigma \wedge \mathrm{on\_zone\_qn}(\mathsf{holding3}(\sigma), b) \Rightarrow (1) \wedge (2) \wedge (3)$

(1) $b.\mathsf{line} = \mathsf{LINE\_holding3ma}(\sigma) \Rightarrow \mathsf{D}(a) \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$.

(2) $b.\mathsf{line} = \mathsf{LINE\_holding3hold}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}}$.

(3) $b.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}}$.

$(\mathbf{A_2}) : \forall a, b : \mathrm{Aircraft}, \forall \sigma : \mathrm{side}, \ \mathrm{on\_zone\_qn}(\mathsf{maz}(\sigma), a) \wedge$
$\mathrm{on\_zone\_qn}(\mathsf{holding3}(\sigma), b) \Rightarrow (1) \wedge (2) \wedge (3)$

(1) $b.\mathsf{line} = \mathsf{LINE\_holding3ma}(\sigma) \Rightarrow \mathsf{D}(a) \leq \frac{b.\mathsf{pos}}{\mathsf{V_{min}}} \cdot \mathsf{V_{max}}$.

(2) $b.\mathsf{line} = \mathsf{LINE\_holding3hold}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + (\mathsf{now} - b.\mathsf{t})) \cdot \mathsf{V_{max}}$.

(3) $b.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma) \Rightarrow \mathsf{D}(a) \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{b.\mathsf{pos}}{\mathsf{V_{d\_min}}}) \cdot \mathsf{V_{max}}$.

$(\mathbf{B}) : \forall a : \mathrm{Aircraft}, \quad \forall \sigma : \mathrm{side},$
$(\mathrm{on\_zone\_qn}(\mathsf{holding3}(\sigma)) \wedge a.\mathsf{line} = \mathsf{LINE\_holding3dec}(\sigma)) \Rightarrow \mathrm{empty\_qn}(\mathsf{holding2}(\sigma))$.

$(\mathbf{C_1}) : \forall a : \mathrm{Aircraft}, (\mathrm{on\_approach\_qn}(a) \wedge \neg \mathrm{first\_in\_seq\_qn}(a)) \Rightarrow$
$\mathsf{D}(\mathrm{leader}(a, \mathsf{landing\_seq})) - \mathsf{D}(a) \geq \mathsf{S_0} - \frac{\mathsf{D}(\mathrm{leader}(a, \mathsf{landing\_seq})) - \mathsf{S_0}}{\mathsf{V_{min}}} (\mathsf{V_{max}} - \mathsf{V_{min}})$.

$(\mathbf{C_2}) : \forall a, b : \mathrm{Aircraft}, (\mathrm{on\_zone\_qn}(\mathsf{runway}, \mathrm{a}) \wedge \mathrm{on\_approach\_qn}(\mathrm{b})) \Rightarrow$
$\mathsf{now} - a.\mathsf{t} \geq \frac{\mathsf{D}(b) - (\mathsf{L_T} - \mathsf{S_T})}{\mathsf{V_{max}}}$

The following conditions hold:

1. A_1, B, *and* C_1 *are step invariants using* Φ.
2. A_2 *is a step invariant using* Φ *and* A_1.
3. C_2 *is a step invariant using* Φ *and* C_1.

From Lemmas 2 and 6, we have the following corollary.

Corollary 7. *The conjunction* $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ *forms a step invariant of* $ContSATS$ *using* Φ.

Conditions A_2, B, and C_2 are used in the refinement proof (Theorem 8) for transitions LowestAvailableAltitude, HoldingPatternDescend, and Taxiing, respectively. Recall that these three transitions are modified from those in the original discrete model, so that $ContSATS$ more realistically models a real system. This is why we need these nontrivial conditions A_2, B, and C_2 in the refinement proof, in order to show that the modified transitions of $ContSATS$ matches with the original transitions of the discrete model. In the proof sketch of Theorem 8, we demonstrate how A_2 is used in the case of LowestAvailableAltitude in the refinement proof.

5.2 Refinement Proof

Now we prove a refinement from $ContSATS$ to $ExtSATS$. We use the mapping r defined in the beginning of Section 5. We use Inv_{Cont}, some auxiliary invariants of $ContSATS$ proved in [14], and Inv_{Ext}, invariants of the discrete model (and thus of $ExtSATS$) proved in [2]. We use $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ as a step invariant using Inv_{Ext} (since Inv_{Ext} implies Φ).

Theorem 8. *The function* r *is a weak refinement from* $ContSATS$ *to* $ExtSATS$ *using* Inv_{Cont}, Inv_{Ext}, *and* $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$.

Proof sketch: Condition 1 is easy to prove.
Condition 2: Suppose α is a step of A. We refer to $\alpha.fstate$ as s and $\alpha.lstate$ as s' in the following. It is easy to see that $s' \in dom(r)$ since r is a total function. We also assume invariants of $ContSATS$, Conditions Φ, and $A_1 \wedge A_2 \wedge B \wedge C_1 \wedge C_2$ hold in s. We demonstrate how a proof goes for Condition 2 by proving the case of the LowestAvailableAltitude(σ) transition. We use Condition A_2 for this case.

Suppose α consists of one LowestAvailableAltitude(σ) transition. From the precondition of the transition, there is at least one aircraft in maz(σ) in s, and thus also in $r(s)$. It follows that LowestAvailableAltitude(σ) is enabled in $r(s)$, and thus an execution fragment β of $ExtSATS$ starting with $r(s)$ that consists of one LowestAvailableAltitude(σ) is a valid execution fragment of $ExtSATS$. It is easy to see $trace(\alpha) = trace(\beta)$. Now we prove $\beta.lstate = r(s')$. If holding3(σ) is empty in s, LowestAvailableAltitude(σ) actually has the exact same effects in $ContSATS$ and $ExtSATS$ (see [14]). Hence it is sufficient to prove that holding3(σ) is empty in s. From the precondition, there is an aircraft a such that

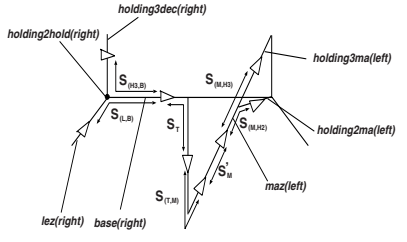
$\mathsf{a.pos} = \mathsf{L_M}$, and $\mathsf{a.line} = \mathsf{LINE\_maz}(\sigma)$. From Condition A_2 and an invariant of $ContSATS$: $\forall b : \mathsf{Aircraft}, b.\mathsf{pos} \leq \mathsf{L}(b.\mathsf{line})$ (this can be easily proved by induction), if $\mathsf{holding3}(\sigma)$ is not empty, then $\mathsf{a.x} = \mathsf{L_M} \leq (\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_3dec}}{V_{d\_min}})\mathsf{V_{max}} - \mathsf{L_T}$. This contradicts the assumption that $(\frac{\mathsf{L_{3ma}}}{\mathsf{V_{min}}} + \mathsf{T_3} + \frac{\mathsf{L_3dec}}{V_{d\_min}})\mathsf{V_{max}} < \mathsf{L_T} + \mathsf{L_M}$. □

From Theorems 8 and 5, we have the following corollary.

Corollary 9. *Let P be an invariant of $ExtSATS$. Then $\lambda s.P(r(s))$ is an invariant of $ContSATS$.*

6 Spacing Properties of Aircraft in *ContSATS*

In the previous section, by using a refinement technique, we proved as Corollary 9 that all invariants of the discrete model of SATS that have been proved in [2] are also invariants of $ContSATS$. For example, from $\Phi.1$ (the number of aircraft in each vertical fix is at most one), we can guarantee two aircraft would never get close in holding2 and holding3 zones in $ContSATS$. This kind of spacing properties of $ContSATS$ are derivable from the invariants of the discrete model, and they express the safe separation of aircraft in one specific zone (represented by a line in $ContSATS$). However, one might be interested in the safe separation of aircraft in two consecutive zones. In this section, we conclude the analysis of safe separation properties for $ContSATS$ in this paper, by presenting such spacing properties for all pairs of consecutive zones in $ContSATS$. The *spacing* between two aircraft is defined as the distance of the two aircraft with respect to the pre-determined paths of $ContSATS$.



$\mathsf{S_{(H3,B)}}$	$\mathsf{L_{3dec}} - \frac{\mathsf{L_{3dec}}}{\mathsf{V_{maz}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$
$\mathsf{S_{(L,B)}}$	$\mathsf{L_l} - \frac{\mathsf{L_l}}{\mathsf{V_{maz}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$
$\mathsf{S_T}$	$\mathsf{S_0} - \frac{\mathsf{L_T} - \mathsf{S_0}}{\mathsf{V_{min}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$
$\mathsf{S_{(T,M)}}$	$\mathsf{S_0} - \frac{\mathsf{L_T}}{\mathsf{V_{max}}}(\mathsf{V_{max}} - \mathsf{V_{min}})$
$\mathsf{S'_M}$	$2\mathsf{S_0} - (\mathsf{L_T} + \mathsf{L_M} - \mathsf{S_0})\Delta$
$\mathsf{S_{(M,H2)}}$	$(1 + \frac{\mathsf{V_{min}}}{\mathsf{V_{max}}})\mathsf{S_0} - \frac{\mathsf{V_{max}} - \mathsf{V_{min}}}{\mathsf{V_{max}}}(\mathsf{L_T} + \mathsf{L_M})$
$\mathsf{S_{(M,H3)}}$	$\mathsf{L_M} + \mathsf{L_T} - \mathsf{L_{3ma}}\Delta.$

Fig. 5. Lower bounds on the spacing of aircraft in two consecutive zones in $ContSATS$

An overview of the spacing properties of aircraft in two consecutive zones that we have proved in [14] is depicted in Figure 5. Each bi-directional arrow in the picture represents a lower bound on the spacing of aircraft. We have proved these properties by induction over the length of the execution of $ContSATS$. To do so, we used invariants carried over from the discrete model to $ContSATS$, by Corollary 9. Among these spacing properties, $\mathsf{S_T}$ and $\mathsf{S'_M}$ are the ones model-checked in [7] (we actually obtained a better bound for $\mathsf{S'_M}$ than [7], using some reasonable assumption stated in [14]).

7 Conclusion

In this paper, we presented a new approach to verify a given hybrid system. For the new approach, we introduced a new refinement proof technique, a *weak refinement using step invariants*. To demonstrate how the approach can be used, we conduct formal verification of NASA's SATS aircraft landing protocol. We believe that this approach is highly applicable to other hybrid systems as well. Proving the soundness of the abstraction used in [3] for a start-up algorithm of TTA by this approach appears one possible interesting future work.

Acknowledgment. we thank anonymous reviewers for their fruitful comments on an earlier version of this paper.

References

1. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers (2006)
2. Umeno, S., Lynch, N.: Proving safety properties of an aircraft landing protocol using I/O automata and the PVS theorem prover: a case study. In: FM 2006: Formal Methods. Volume 4084 of Lecture Notes in Computer Science., Hamilton, Ontario Canada (2006) 64 – 80
3. Steiner, W., Rushby, J., Sorea, M., Pfeifer, H.: Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation. In: Proc. of the 2004 International Conference on Dependable Systems and Networks, Florence, Italy, IEEE Computer Society (2004) 189–198
4. Kopetz, H., Bauer, G.: The time-triggered architecture. Proceedings of The IEEE **91; PART 1** (2003) 112–126
5. T.Abbott, Jones, K., Consiglio, M., Williams, D., Adams, C.: Small Aircraft Transportation System, High Volume Operation concept: Normal operations. Technical Report NASA/TM-2004-213022, NASA Langley Research Center, NASA LaRC,Hampton VA 23681-2199, USA (2004)
6. Dowek, G., Muñoz, C., Carreño, V.: Abstract model of the SATS concept of operations: Initial results and recommendations. Technical Report NASA/TM-2004-213006, NASA Langley Research Center, NASA LaRC,Hampton VA 23681-2199, USA (2004)
7. Muñoz, C., Dowek, G.: Hybrid verification of an air traffic operational concept. In: Proceedings of IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation, Columbia, Maryland (2005)
8. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In Kapur, D., ed.: 11th International Conference on Automated Deduction (CADE). Volume 607 of Lecture Notes in Computer Science., Saratoga, NY (1992) 748 – 752
9. Lynch, N., Segala, R., Vaandraager, F.: Hybrid I/O automata. Information and Computation **185(1)** (2003) 105–157
10. Mitra, S., Wang, Y., Lynch, N., Feron, E.: Safety verification of model helicopter controller using hybrid Input/Output automata. In: HSCC'03, Hybrid System: Computation and Control, Prague, the Czech Republic (2003)

11. Livadas, C., Lygeros, J., Lynch, N.A.: High-Level Modeling and Analysis of the Traffic Alert and Collision Avoidance System (TCAS). Proceedings of the IEEE, Special Issue on Hybrid Systems: Theory & Applications **88**(7) (2000) 926–948
12. Fehnker, A., Zhang, M., Vaandrager, F.: Modeling and verifying a lego car using hybrid I/O automata. In: Third International Conference on Quality Software (QSIC 2003), Dallas, Texas, USA, IEEE Computer Society Press (2003)
13. Lynch, N., Vaandrager, F.: Forward and backward simulations – part II: Timing-based systems. Information and Computation **128(1)** (1996) 1 – 25
14. Umeno, S.: Proving safety properties of an aircraft landing protocol using timed and untimed I/O automata: a case study. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA (2006)
15. Henzinger, T.A., Minea, M., Prabhu, V.: Assume-guarantee reasoning for hierarchical hybrid systems. In: Proc. of HSCC'01, Hybrid Systems: Computation and Control. Volume 2034 of Lecture Notes in Computer Science. (2001) 275 – 290
16. Frehse, G., Han, Z., Krogh, B.: Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In: CDC 2004: IEEE Conference on Decision and Control. (2004)
17. Garland, S.: TIOA User Guide and Reference Manual. (2005)