A Small Model Theorem for Rectangular Hybrid Automata Networks

Taylor T. Johnson and Sayan Mitra

Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Email: {johnso99, mitras}@illinois.edu

Abstract. Rectangular hybrid automata (RHA) are finite state machines with additional skewed clocks that are useful for modeling real-time systems. This paper is concerned with the uniform verification of safety properties of networks with arbitrarily many interacting RHAs. Each automaton is equipped with a finite collection of pointers to other automata that enables it to read their state. This paper presents a small model result for such networks that reduces the verification problem for a system with arbitrarily many processes to a system with finitely many processes. The result is applied to verify and discover counterexamples of inductive invariant properties for distributed protocols like Fischer's mutual exclusion algorithm and the Small Aircraft Transportation System (SATS). We have implemented a prototype tool called Passel relying on the satisfiability modulo theories (SMT) solver Z3 to check inductive invariants automatically.

Keywords: hybrid automata, parameterized verification, small model theorem, uniform verification

1 Introduction

Distributed systems are naturally modeled as a collection of interacting building-blocks or *modules*. For example, distributed computing systems are built from communicating computing processes, distributed traffic control protocols involve the interaction of individual vehicles, and neural networks arise from the interaction of neurons. This paper presents a result for *parameterized verification* of systems composed of such modules. For parameterized verification, a property P and a module template $\mathcal{A}_i$ are given. The property must be independent of the number and the identities of the modules, and we must verify that P holds for any system built from arbitrarily many instances of $\mathcal{A}_i$. That is,

$$\forall\ N \in \mathbb{N},\ \mathcal{A}_1 \| \mathcal{A}_2 \| \ldots \| \mathcal{A}_N \models P, \tag{1}$$

where N is not fixed and the precise meaning of the parallel composition of operator $\|$ depends on the particular modeling framework.

The rectangular hybrid automaton (RHA) modeling framework [5,29] combines finite state machines with continuous variables. It has proven to be useful

for modeling protocols and control logics with timers, and for approximating more complex linear and nonlinear dynamics with piecewise constant dynamics. Several subclasses of RHA have been identified for which safety verification is decidable [29] and model checking tools have been developed [28,22,23].

In this paper, we consider parameterized verification of RHA networks where modules communicate by reading one another's state and through globally shared variables. A RHA may read the variables of another RHA through the use of a pointer variable tracking the identifier of that automaton. Such communication allows us to model distributed traffic control systems like the Small Aircraft Transportation System (SATS) [1,34,38]. In SATS, aircraft communicate by reading the valuations of discrete variables and continuous positions through pointer variables. The pointer variables allow us to model systems where the communication topology is dynamic. Thus, a variety of other distributed cyber-physical systems (DCPS) can be modeled in this manner through the use of pointer variables. For instance, in the automated highway system, a car may only need to keep track of the positions of the cars immediately ahead and behind it requiring two pointer variables, and similar scenarios arise in robotic swarm protocols in one-dimensional lanes [30]. However, at a four-way intersection of single lane roads, an autonomous car may need to track the positions of cars coming from every other direction, requiring three pointer variables. All of these scenarios fit into the communication model and verification framework developed in this paper.

To perform verification of an infinite number of such infinite-state RHA, we develop and use a small model result for RHA networks. Small model theorems are used to prove decidability of checking satisfiability of first-order logic (FOL) formulas. The philosophy behind applying them in verification is to identify classes of systems and specifications with the small model property, which reduces an infinite problem to a finite one. Many prefix classes of FOL were shown to be decidable by showing that the class has the *finite model property*: every satisfiable formula also has a finite model [11]. In many cases the proof of the finite model property comes with an explicit bound on the size of the finite structure that may satisfy (or violate) the property in question. For parameterized verification, small model theorems provide a finite threshold N_o such that if, for all $N \leq N_o$, $A^N \triangleq \mathcal{A}_1 \| \ldots \| \mathcal{A}_N \models P$, then Equation 1 also holds.

The contributions of this paper are:

(a) A small model theorem for RHA networks that guarantees the existence of a bound N_o, such that, if an instantiation of A^N violating P exists for some $N > N_o$, then $\mathcal{A}^{N_o}$ must also violate P. Thus, the verification problem from Equation 1 is solved if, for all $N \leq N_o$, no instantiation A^N violates P.
(b) The theorem is applied in a software tool called Passel that we use to automatically check inductive invariants up to the bound N_o using the satisfiability modulo theories (SMT) solver Z3 [16].

The input to Passel is a hybrid automaton specification $\mathcal{A}_i$ and a candidate safety property P. Then, the bound N_o is computed from the syntactic descrip-

tion of $\mathcal{A}_i$ and P. In addition to these hybrid protocols, we have also verified several purely discrete algorithms (cache coherence protocols and mutual exclusion algorithms like the simplified bakery algorithm) studied in [18,2]. The experiments have indicated that it is feasible to develop automatic methods relying on our small model result that apply both to real-time distributed systems and classic distributed algorithms. The success of our experiments in part relies on the strengths of state-of-the-art SMT solvers like Z3, which allow for quantified formulas and have quantifier elimination and instantiation procedures for real and integer arithmetic [24,10].

Related Work To the best of our knowledge, the automatic parameterized verification problem has not been addressed previously for RHA, but there are several works addressing parameterized verification for networks of the special subclass of timed automata [4,3,25,14,21,13]. Parameterized verification of RHA networks is useful to show, for instance, that for arbitrarily many aircraft participating in a given distributed air traffic control protocol like the Small Aircraft Transportation System (SATS), no two aircraft ever collide [34,38,31]. We use a simplified version of SATS as a case study to illustrate the concepts and results developed in this paper. There are several partly manual works for parameterized verification of timed and hybrid systems using theorem provers [20,34,38,35,36,32].

An overview of automatic approaches for parameterized verification of discrete systems appears in [15, Ch. 15]. The parameterized verification problem is in general undecidable, even for finite-state modules [7]. However, for restricted classes of systems under various communications constraints, the problem has been shown to be decidable. For instance, parameterized verification is decidable for safety properties of timed networks considered in [4,14]. However, each timed automaton in the network is assumed to have either (a) a single real-valued clock, or (b) any finite number of discrete-valued clocks [3]. If the timed automata each have more than a single real-valued clock, then the problem is undecidable [3]. The previous undecidability result prevents using the standard initialized rectangular hybrid automata (IRHA)-to-timed automata conversion algorithm [6,29] because it adds two clocks for every continuous variable evolving with rectangular dynamics.

There are a variety of automatic and semi-automatic approaches to parameterized verification, but the most closely related are network invariants [40]. Finding invariants was automated with the invisible invariants approach [37,8], which provides a heuristic method to automatically compute inductive invariants, such as implemented in [9]. A small model theorem like the one presented in this paper was introduced in [37] for a class of discrete parameterized systems with bounded data. Network invariants have previously been developed for timed parameterized systems in [25]. Another automated approach for parameterized verification of timed systems is presented in [21]. One can view cutoffs—an instantiation of the system that has all the behaviors of additional compositions—like those in [26] like small model results, in that it is sufficient to check the composition of a protocol up to the cutoff or small model bound to verify the parameterized specification.

2 Modeling Framework

In this section, we present the syntax for a class of assertions we call *LH-assertions*, introduce a modeling framework for networks of rectangular hybrid automata (RHA), and then show how inductive invariants for such networks can be asserted in terms of LH-assertions. We will then develop the small model theorem for LH-assertions. Let $N \geq 2$ be a natural number. The set $[N] \triangleq \{1, \ldots, N\}$ is used for indexing RHA. For a set S, we define $S_\perp \triangleq S \cup \{\perp\}$.

2.1 LH-Assertions

LH-assertions are built-up from constants, variables, arrays, and terms of several different types. We introduce LH-assertions first because we will use LH-assertions for specifying the syntactic description of individual RHA and RHA networks below. Throughout, natural numbers are used for indexing arrays. The numbers 1 and N are *constants* of type $\mathbb{N}$ and $L = \{L_1, \ldots, L_k\}$ is a fixed finite type. The signature of LH-assertions involves a finite number of variables of the following types: (a) *Index variables*: $i_1, \ldots, i_b : [N]_\perp$, (b) *Discrete variables*: $l_1, \ldots, l_c : L$, (c) *Real variables*: $x_1, \ldots, x_d, t_1, \ldots, t_d : \mathbb{R}$, (d) *Index-valued array variables*: $\bar{p}_1, \ldots, \bar{p}_e : [N] \to [N]_\perp$, (e) *Discrete array variables*: $\bar{l}_1, \ldots, \bar{l}_f : [N] \to L$, and (f) *Real array variables*: $\bar{x}_1, \ldots, \bar{x}_g : [N] \to \mathbb{R}$.

The grammar for constructing LH-assertions is defined as follows.

$$\begin{aligned} \mathsf{ITerm} &::= 1 \mid N \mid i_j \mid \bar{p}_k[\mathsf{ITerm}] \\ \mathsf{DTerm} &::= L_j \mid l_k \mid \bar{l}_j[\mathsf{ITerm}] \\ \mathsf{RTerm} &::= x_j \mid \bar{x}_k[\mathsf{ITerm}] \end{aligned}$$

An index term (ITerm) is either (a) one of the constants 1, N, or an index variable i_j, or (b) an index array $\bar{p}_k$ referenced at 1, N, or i_j. We use the notation $\bar{y}[i]$ to denote the valuation of the array $\bar{y}$ at the value of the index variable i. Discrete terms (DTerm) and real terms (RTerm) are defined as specified above. Here, L_j is constant from L, l_k is a discrete variable, $\bar{l}_j$ is a discrete array, x_j is a real variable, and $\bar{x}_k$ is a real array. Using these terms, formulas are defined next.

$$\begin{aligned} \mathsf{Atom} &::= \mathsf{ITerm}_1 < \mathsf{ITerm}_2 \mid \mathsf{DTerm} = L_k \mid a_1\mathsf{RTerm}_1 + a_2\mathsf{RTerm}_2 + a_3 < 0 \\ \mathsf{Formula} &::= \mathsf{Atom} \mid \neg\mathsf{Formula} \mid \mathsf{Formula}_1 \wedge \mathsf{Formula}_2 \end{aligned}$$

Here, a_1, a_2, and a_3 are real-valued numerical constants used to specify some real linear arithmetic constraint, and $\mathsf{Formula}_1$ and $\mathsf{Formula}_2$ are shorter formulas that are joined by boolean operators to obtain a longer formula. By combining the boolean operators $\wedge$ and $\neg$ with the $<$ operator, other comparison operators, such as $=$, $\neq$, $\leq$, $>$, and $\geq$, can be expressed in formulas for both indices and reals. For example, $\bar{p}_1[i_j] = \bar{p}_2[i_k]$ can be written as $\neg(\bar{p}_1[i_j] < \bar{p}_2[i_k]) \wedge \neg(\bar{p}_2[i_k] < \bar{p}_1[i_j])$.

Given a formula Formula, a *sentence*—a formula with no free variables—is obtained by quantifying all the free index and real variables. An LH-assertion is a sentence of the form $\forall t_1 \in \mathbb{R} : \forall i_1, \ldots, i_k \in [N] : \exists t_2 \in \mathbb{R} : \exists j_1, \ldots, j_m \in [N] : \varphi$,

where φ is a formula. We mention that t_1 and t_2 are only used in practice to respectively model an elapse of time of length t_1 and enforcing invariants for all trajectories of lengths $0 \leq t_2 \leq t_1$. We provide several example quantified sentences and LH-assertions:

$$\forall i, j : i \neq j \implies (\bar{l}[i] = \bar{l}[j] \implies |\bar{x}[j] - \bar{x}[i]| > a), \tag{2}$$

$$\forall i, j : i = j \ \vee \ \bar{l}[i] = \bar{l}[j] \ \vee \ (\bar{x}[j] - \bar{x}[i] - a < 0) \ \vee \ (\bar{x}[i] - \bar{x}[j] - a < 0), \tag{3}$$

$$\forall i \ \exists j : \bar{p}[i] = j \wedge |\bar{x}[i] - \bar{x}[\bar{p}[i]]| > a. \tag{4}$$

We only use LH-assertions with t_1 and t_2 for checking continuous transitions as shown in Subsection 3.1. Reading these assertions as statements about networks of automata, the first one states that all automata with identical values of the discrete variable $\bar{l}$ have a minimum gap of a between the values of their $\bar{x}$ variables. The first assertion is an abbreviation of the second. The last assertion states every automaton has a pointer $\bar{p}$ to another automaton and that there is a minimum separation of a between its $\bar{x}$ value and the $\bar{x}$ value of the automaton it points to.

Models for Sentences and Assertions A *model* for an assertion provides interpretation to the elements appearing in the assertion. Specifically, an n-model M for an LH-assertion ψ is denoted $M(n, \psi)$ and provides an interpretation of each the *free* variables in ψ as follows:

- the constants 1 and N are assigned the values 1 and n,
- each index variable is assigned a value in the set $\{1, \ldots, n\}$,
- each discrete variable is assigned a value in L,
- each real variable is assigned a value in $\mathbb{R}$, and
- each index, discrete, and real array is assigned respectively a $\{1, \ldots, n\}_\perp$-valued, L-valued, and real-valued array of length $[1, \ldots, n]$.

For example, a 2-model for the assertion of Equation 4 is specified by the assignments $N = 2$, $\bar{p} = \langle 2, 1 \rangle$, and $\bar{x} = \langle 0, 10.0 \rangle$, assuming any choice for the real constant $a < 10$. Given an assertion ψ and a model $M(n, \psi)$, if ψ evaluates to true with the interpretations of the free variables given by $M(n, \psi)$, then $M(n, \psi)$ is said to *satisfy* ψ. If all models of ψ satisfy it, then the assertion is said to be *valid*. If there exist models that satisfy ψ, then the assertion is said to be *satisfiable*. Fixing $a = 9.0$, the above model satisfies assertion Equation 4, but it is not valid because the 1-model $\bar{p} = \langle 1 \rangle, \bar{x} = \langle 10.0 \rangle$ does not satisfy it, since $|\bar{x}[1] - \bar{x}[1]| = 0 \not> a$.

2.2 Networks of Rectangular Hybrid Automata

Informally, in the context of networks of rectangular hybrid automata (RHA), the arrays of discrete, real, and index variables respectively represent discrete, continuous, and pointer variables of individual automata, while the ordinary

(non-array) variables represent globally shared variables[1]. In this section, we introduce the syntax of a language for specifying networks of rectangular hybrid automata, and then introduce the semantics of the language and show how the networks can be modeled with LH-assertions.

A RHA $\mathcal{A}_i$ is a (possibly nondeterministic) finite state machine augmented with skewed real-valued clocks. A *RHA Network* is a collection of RHA in which the transitions of each RHA can depend on the the state of certain other RHA. In particular, these certain other RHA are specified through the use of pointer (index-valued) variables. In this paper, we consider RHA Networks that are composed of *arbitrarily many* identical RHA. If a module contains several RHA subsystems $\mathcal{A}_1$, $\mathcal{B}_1$, $\mathcal{C}_1$, ..., the composition of these subsystems can be taken first prior to composing the RHA network.

Syntax of Individual RHA Syntactically, a RHA $\mathcal{A}_i$ for $i \in [N]$ is specified by the following components: (a) a list of variable names Var_i, (b) a list of action names Act_i, (c) a list of mode names Mode_i, (d) an assertion Init_i on Var_i, (e) a collection of precondition-effect statements—one for each element in Act_i, and (f) a collection of invariant-stop-flow statements—one for each element in Mode_i. For example, Figure 1 shows a complete specification for a RHA modeling the simplified SATS air traffic control protocol.

Now we describe the syntactic structure of each of these components. Each variable in Var_i is associated with a type. The type can be (a) *a finite set* L, (b) *the set of indices* (augmented with the special element $\perp$) $[N]_\perp$, or (c) *the set of reals*. Each variable in Var_i has a name of the form $\langle variable\_name \rangle[i]$. For example, $l[i] : L$, $p[i] : [N]_\perp$, and $x[i] : \mathbb{R}$, define discrete, index, and real variables in Var_i. Looking ahead, this naming convention will be consistent with the syntax of a class of assertions called LH-assertions, that are used for defining the RHA network. One array type variable $\bar{l}$ will encode valuation of all the $l[i]$ variables in the network. That is, the state variables of a RHA network will be represented by arrays of discrete, continuous, and index variables.

The initial assertion Init_i is specified by a sentence involving the variables in Var_i. IN SATS for instance, $\mathsf{Init}_i : l[i] = F \wedge next[i] = \perp \wedge last = \perp$, where *last* is a global variable. For each action $\mathsf{a} \in \mathsf{Act}_i$, the precondition—denoted by $\mathbf{pre}(\mathsf{a}, i)$—is a sentence involving the variables in Var_i with possibly additional quantified index variables. The effect, denoted by $\mathbf{eff}(\mathsf{a}, i)$, is a sentence involving both the variables in Var_i as well as their primed versions. For example, the

[1] A real-typed variable may be updated continuously and/or discretely, while variables of other types are only updated discretely—we do not explicitly partition the sets of real and real array variables for simplicity of presentation, but will mention it when defining the semantics.

following are precondition-effect statements for some labels setPtr and chkGap:

$$\begin{aligned}
\mathbf{pre}(i, \mathsf{setPtr}) &: \forall j : l[i] = L_1 \ \wedge \ (j \neq i \implies j \neq p[i]), \\
\mathbf{eff}(i, \mathsf{setPtr}) &: \exists j : l'[i] = L_2 \ \wedge \ x'[i] = 0 \ \wedge \ j \neq i \ \wedge \ p'[i] = j, \\
\mathbf{pre}(i, \mathsf{chkGap}) &: l[i] = L_2 \ \wedge \ x[p[i]] > 20, \\
\mathbf{eff}(i, \mathsf{chkGap}) &: l'[i] = L_3 \ \wedge \ x'[i] = 0.
\end{aligned}$$

Note that the precondition and effect sentences may have additional quantified index variables apart from i with the $\exists\ \forall$ quantifier ordering.

For each mode $\mathsf{m} \in \mathsf{Mode}_i$, the invariant—denoted by $\mathbf{inv}(\mathsf{m}, i)$—and the stopping condition—denoted by $\mathbf{stop}(\mathsf{m}, i)$—are sentences involving the variables in Var_i with possibly additional quantified index variables. For each $\mathsf{m} \in \mathsf{Mode}_i$, the flow statement associates two real constants with each real-valued, continuously updated variable in Var_i. For variable $x[i]$, these constants are denoted by $\mathbf{lflowrate}(\mathsf{m}, i, x)$ and $\mathbf{uflowrate}(\mathsf{m}, i, x)$. This allows for modeling the usual rectangular dynamics $\mathbf{flowrate}(\mathsf{m}, i, x) \in [a, b]$ for $a \leq b$, where a equals the lower rate and b equals the upper. If $a = b$, we say the dynamics are timed. For example, for mode Base of SATS (see Figure 1),

$$\begin{aligned}
\mathbf{inv}(\mathsf{Base}, i) &: l[i] = B \ \wedge \ x[i] \leq L_B \\
\mathbf{stop}(\mathsf{Base}, i) &: l[i] = B \ \wedge \ x[i] = L_B \\
\mathbf{lflowrate}(\mathsf{Base}, i, x) &: v_{min} \ \wedge \ \mathbf{uflowrate}(\mathsf{Base}, i, x) : v_{max}
\end{aligned}$$

so that $\mathbf{flowrate}(\mathsf{Base}, i, x) \in [v_{min}, v_{max}]$.

2.3 Example: Simple Air Traffic Landing Protocol

We use a simplified version of the Small Aircraft Transportation System (SATS), a distributed air traffic control protocol, as a running example for the remainder of the paper [1,34,38,31]. SATS is a program aimed at increasing throughput at small airports without air traffic controllers by allowing aircraft to communicate between themselves along with a centralized communication component at the airport used to determine a landing sequence order [1,39]. Aircraft in SATS communicate by reading the continuous position of any aircraft immediately ahead of it in the landing sequence, where the aircraft immediately ahead is tracked using a pointer. The example is parameterized on the number of aircraft involved in the landing attempt, and is naturally modeled as a RHA network. The system models a single airport and N flying aircraft that are attempting to land. After determining the landing sequence order from a centralized airport management module (AMM) located at the airport—which we model using the global shared variable *last*—the remainder of the protocol is decentralized and each aircraft communicates with the aircraft immediately ahead of it (if one exists) to determine if it is safe to attempt landing.

All aircraft begin in the Flying mode, and when an aircraft is ready to attempt landing, it initiates the approach to the airport by making a discrete

1 **Var**: $l[i]$:{F,H,B,R}, $x[i]$:$\mathbb{R}$, $next[i]$:$[N]_\perp$
Global Var: $last$:$[N]_\perp$
3 **Init**: $l[i] = F \wedge next[i] = \perp \wedge last = \perp$
Mode: Fly, Holding, Base, Runway
5
Fly: **Inv**: $l[i] = F$
7 **Flowrate**: $\dot{x}[i] = 0$

9 Holding: **Inv**: $l[i] = H$
Flowrate: $\dot{x}[i] = 0$
11
Base: **Inv**: $l[i] = B \wedge x[i] \leq L_B$
13 **Stop**: $x[i] = L_B$
Flowrate: $\dot{x}[i] \in [v_{min}, v_{max}]$
15
Runway: **Inv**: $l[i] = R$
17 **Flowrate**: $\dot{x}[i] = 0$

Act: FtoH, HtoB, BtoH, BtoR
FtoH: **Pre**: $l[i] = F$ 20
Eff: $l'[i] = H \wedge next'[i] = last \wedge last' = i$
22
HtoB: **Pre**: $l[i] = H \wedge (next[i] = \perp$
$\vee\, l[\ next[i]\ ] = B \wedge x[\ next[i]\ ] \geq L_S)$ 24
Eff: $l'[i] = B \wedge x'[i] = 0$
26
BtoH: **Pre**: $l[i] = B$
Eff: $l'[i] = H \wedge x'[i] = 0 \wedge$ 28
$(last \neq i \Rightarrow next'[i] = last) \wedge last' = i$
$\wedge \forall\, j \neq i$ (**if** $next[j] = i$ **then** $next'[j] =$ 30
$\perp$**else** $next'[j] = next[j]$)
32
BtoR: **Pre**: $l[i] = B \wedge x[i] \geq L_B \wedge next[i] = \perp$
Eff: $l'[i] = R \wedge (last = i \Rightarrow last' = \perp)$ 34
$\wedge \forall\, j \neq i$ (**if** $next[j] = i$ **then** $next'[j] =$
$\perp$**else** $next'[j] = next[j]$) 36

Fig. 1. RHA $\mathcal{A}_i$ for simplified SATS protocol.

transition to the Holding mode. The Holding mode physically represents that the aircraft is flying in a cyclic holding pattern, and it is assumed the aircraft maintain a safe separation in this mode. On entering Holding, an aircraft is either designated as the first one in the landing sequence (and $next[i] = \perp$), or the aircraft is assigned the identifier of the last aircraft that began its approach to the runway (and $next[i] = last$). Subsequently, an aircraft may nondeterministically transition from the Holding mode to the Base mode and is now physically approaching the runway. The position of the i^{th} aircraft is modeled using a single continuous variable ($x[i]$) of real type, representing the position along a line measured starting from the geographic location of the cyclic holding zone (that is, the beginning of the base region). This transition is only enabled for aircraft i if there is at least L_S distance between its position $x[i]$ and the position of the aircraft ahead of it, $x[next[i]]$ (if one exists). Once in the Base mode, the aircraft is approaching the runway and after traversing L_B distance, the aircraft may either (a) cancel the landing attempt and return to the cyclic holding pattern in mode Holding, in which case it becomes the last aircraft in the sequence, or (b) the aircraft may succeed in landing and set its mode to Runway.

SATS Properties We checked the following properties for SATS with memory and time required as indicated in Table 1. The properties specifying a safe separation are D and E. Note that we checked property D only for rectangular dynamics (that is, Figure 1, line 14 is as written) and E only for timed dynamics (that is, the rectangular dynamics in Figure 1, line 14 are replaced by $\dot{x}[i] = 1$). This is because SATS with rectangular dynamics does not satisfy E. The properties are:

(A) $\forall i \in [N] : l[i] = F \Rightarrow last \neq i$,
(B) $\forall i, j \in [N] : next[j] = i \Rightarrow l[i] \neq F$,
(C) $\forall i, j \in [N] : l[i] = H \wedge next[j] = i \Rightarrow l[j] = H$,

(D) $\forall i, j \in [N] : l[i] = B \wedge l[j] = B \wedge next[j] = i \Rightarrow x[i] \geq L_S + (v_{max} - v_{min})\frac{L_B - x[j]}{v_{min}}$, and
(E) $\forall i, j \in [N] : i \neq j \wedge l[i] = B \wedge l[j] = B \wedge next[j] = i \Rightarrow x[i] - x[j] \geq L_S$.

2.4 Semantics of RHA Networks

Given the syntactic specification of a single RHA $\mathcal{A}_i$, the semantic definition of a RHA network, where arbitrarily many $\mathcal{A}_i$'s execute in parallel, is obtained as follows. We note this is essentially $N-1$ parallel compositions like those considered in [27]. For any $N \in \mathbb{N}$, the automaton A^N is the tuple $\langle V_N, Q_N, \Theta_N, D_N, T_N \rangle$. We define each of these components as follows.

V_N is a set of variables, with a real array $\bar{x}_1$ corresponding to each real variable x_1 in the Var_i list, an index array $\bar{p}_1$ corresponding to each index variable p_1 in the Var_i list, and so on. Additionally, V_N contains a set of global (non-array) variables of assorted types, corresponding to the non-array variables in the LH-assertions.

Q_N is the set of all possible valuations of the variables in V_N. Elements of Q_N are called *states* and are denoted by boldface $\mathbf{v}$, $\mathbf{v}'$, etc. At a state $\mathbf{v}$, the valuation of a particular array variable $\bar{p}$ is denoted by $\mathbf{v}.\bar{p}_1$, and $\mathbf{v}.g$ for some non-array variable g in V_N. The valuation of the variables in Var_i at state $\mathbf{v}$ is denoted by $\mathbf{v}[i]$. Given a state $\mathbf{v}$ and a quantified sentence Sent involving the arrays in V_N and index variables, we say that $\mathbf{v}$ *satisfies* Sent iff fixing the values of the variables in Sent yields an assertion that is valid. In that case, we write $\mathbf{v} \models \mathsf{Sent}$.

$\Theta_N \subseteq Q_N$ is called the *set of initial states*, and is the set of states that satisfy the assertion Init_i for every index $i \in [N]$,

$$\{\mathbf{v} \in Q_N \mid \forall\, i \in [N] : \mathbf{v}[i] \models \mathsf{Init}_i\}.$$

$D_N \subseteq Q_N \times Q_N$ is called the set of *discrete transitions* and is defined as follows: $(\mathbf{v}, \mathbf{v}') \in D_N$ iff:

$$\exists\, i \in [N] :\ \exists\, \mathsf{a} \in \mathsf{Act}_i : \mathbf{v} \models \mathbf{pre}(\mathsf{a}, i) \ \wedge\ (\mathbf{v}, \mathbf{v}') \models \mathbf{eff}(\mathsf{a}, i) \wedge \forall\, j \in [N] :\ j \neq i \wedge j \notin M \implies \mathbf{v}'[j] = \mathbf{v}[j],$$

where M is a (possibly empty) subset of $[N]$ corresponding to any indices (excluding i) of the valuations of array variables modified by the $\mathbf{eff}(\mathsf{a}, i)$ statement. For instance, in SATS (see Figure 1, line 31), if i transitions from Base back to Hold, then if $next[m] = i$ for any $m \neq i$, then $next'[m] = \perp$, and $M = \{m\}$.

$T_N \subseteq Q_N \times Q_N$ is called the set of *trajectories*. To define T_N we first define the relation $\mathbf{flow}(\mathsf{m}, \mathbf{v}[i], t)$ that returns a set of valuations $\mathbf{v}'[i]$, such that for each $x \in \mathsf{Var}_i$, if x's type is not real (or is real, but is only updated discretely), then $\mathbf{v}'[i].x = \mathbf{v}[i].x$, but otherwise, $\mathbf{v}[i].x + \mathbf{lflowrate}(\mathsf{m}, i, x)t \leq$

$\mathbf{v}'[i].x \leq \mathbf{v}[i].x + \mathbf{uflowrate}(\mathsf{m}, i, x)t$. A pair $(\mathbf{v}, \mathbf{v}') \in T_N$ iff:

$$\begin{aligned}\exists t_1 \in \mathbb{R}_{\geq 0} : \ & \forall i \in [N] : \exists \mathsf{m} \in \mathsf{Mode}_i : \forall t_2 \in \mathbb{R}_{\geq 0} : t_2 \leq t_1 \wedge \\ & (\mathbf{flow}(\mathsf{m}, \mathbf{v}[i], t_2) \models \mathbf{inv}(\mathsf{m}, i) \wedge \mathbf{flow}(\mathsf{m}, \mathbf{v}[i], t_2) \models \mathbf{stop}(\mathsf{m}, i) \Rightarrow t_2 = t_1) \\ & \wedge \mathbf{v}'[i] \in \mathbf{flow}(\mathsf{m}, \mathbf{v}[i], t_1).\end{aligned}$$

Informally, a discrete transition from $\mathbf{v}$ to $\mathbf{v}'$ models the discrete transition of a particular RHA $\mathcal{A}_i$ by some action $\mathsf{a} \in \mathsf{Act}_i$. The precondition of action a may depend on the variables in Var_j for $j \neq i$. The effect is usually defined in terms of the variables in Var_i, but may also set variables in Var_j if j is the valuation of some index-valued variable of i. A trajectory models a transition from $\mathbf{v}$ to $\mathbf{v}'$ that occurs over some interval of time with length t_1. All the non-continuously updated variables (discrete variables, pointers, and any real variable updated only discretely) remain unchanged. For each $i \in [N]$ and each continuously updated real variable $x \in \mathsf{Var}_i$, $\mathbf{v}[i].x$ must evolve to the valuations $\mathbf{v}'[i].x$, in exactly t_1 time in some mode $m \in \mathsf{Mode}_i$. All intermediate states along the trajectory must also satisfy the invariant $\mathbf{inv}(\mathsf{m}, i)$, and if an intermediate state satisfies $\mathbf{stop}(\mathsf{m})$, then that state must be $\mathbf{v}'$ (that is, the end of a trajectory).

The behavior of a network of RHA is defined as sequences of states that are related by transitions and trajectories. An *execution* of A^N is a sequence of states $\mathbf{v}_0, \mathbf{v}_1, \ldots$ such that $\mathbf{v}_0 \in \Theta_N$, and for each index k appearing in the sequence, if k is even then $(\mathbf{v}_k, \mathbf{v}_{k+1}) \in T_N$, and otherwise $(\mathbf{v}_k, \mathbf{v}_{k+1}) \in D_N$.

3 Small Model Theorem

We begin this section with the main small model result, Lemma 1, for LH-assertions with the signature introduced in the previous section. Then we show how inductive invariants for networks of RHAs can be encoded as LH-assertions. Thus, for a specific inductive invariant I, Lemma 1 provides a threshold on size of models, written $n(I)$. If for all $N \leq n(I)$, I is an inductive invariant for A^N, then I is indeed an inductive invariant for all $N \in \mathbb{N}$. This makes it possible to verify inductive invariants for parameterized networks of hybrid automata by verifying I for a finite number of instances of A^N.

Lemma 1. *Let ψ be a LH-assertion of the form $\forall t_1 \in \mathbb{R}\ \forall i_1, \ldots, i_k \in [N]$ $\exists t_2 \in \mathbb{R}\ \exists j_1, \ldots, j_m \in [N] : \varphi$, where φ is a quantifier-free formula involving the index variables $i_1, \ldots, i_k$, $j_1, \ldots, j_m$, real variables t_1 and t_2, and global and array variables in V_N. Then, ψ is valid iff for all $n \leq N_0 = (e+1)(k+2)$, ψ is satisfied by all n-models, where e is the number of index array variables in φ and k is the largest subscript of the universally quantified index variables in ψ.*

Proof. We assume that all models of size n, for $n \leq (e+1)(k+2)$, satisfy ψ. It suffices to show that ψ is valid. Suppose for the sake of contradiction that ψ is not valid. Then there exists a model M of size $n > (e+1)(k+2)$ that satisfies $\neg\psi = \exists\ t_1, i_1, \ldots, i_k : \forall t_2, j_1, \ldots, j_m : \neg\varphi$. We will show that for any model of

size $n > (e+1)(k+2)$, there exists a model of size $n-1$ that contradicts the assumption.

The n-model M assigns a real value to the variable t_1 and values in $\{1,\ldots,n\}$ to the index variables $i_1,\ldots,i_k$ (in addition to providing interpretations for the other bounded variables and arrays). The values assigned to the universally quantified variables $t_2, j_1,\ldots,j_m$ in the model M are not important, because any value of these variables would satisfy $\neg\psi$. The set of values assigned to $i_1,\ldots,i_k$ can contain at most k distinct values. Consider an index term with one of the forms 1, N, or i_m, where i_m is an existentially quantified index variable in $\neg\psi$: any such term can take at most $k+2$ distinct index values. Thus, an index array term $\bar{p}[i_m]$ can take at most $k+2$ distinct values. Since there are at most e index arrays, the set of all possible index arrays and terms can take at most $(e+1)(k+2)$ distinct values. Therefore, there exists a value in $\{1,\ldots,n\}$, say u, that is not assigned to any index variable or to any of the referenced values of the index arrays, in M.

Now, we define an $(n-1)$-model M' by removing u from $\{1,\ldots,n\}$ and shifting values appropriately. The constants n is interpreted as $n-1$ in M'. For each index variable i_j, if $i_j < u$ then we assign $M'(i_j) = M(i_j)$, and otherwise we assign $M'(i_j) = M(i_j)-1$. For each (index, discrete, or real) array $\bar{z}$, for each $i \in \{1,\ldots,n-1\}$, if $i < u$ then we assign $M'(\bar{z}[i]) = M(\bar{z}[i])$, and otherwise we assign $M'(\bar{z}[i]) = M(\bar{z}[i+1])$. Finally, it is routine to check that M' assigns the same binary value to each Atom in φ as M, and therefore M' also satisfies $\neg\psi$.

3.1 Applying the Small Model Result to Check Inductive Invariants

For an automaton network A^N, an *invariant assertion* is a logical sentence involving the variables in V_N (and possibly the global variables). In this paper we will consider *regular invariants* in which (a) the indices of all the arrays are index variables (and not constants), and (b) index variables can only be compared with other index variables (not constants). Furthermore, we require the regular invariants to have all the universal quantifiers precede the existential quantifiers. Thus, a regular invariant is of the form $\psi \triangleq \forall\, i_1,\ldots,i_k \in [N] : \exists j_1,\ldots,j_m \in [N] : \varphi$, where φ is a quantifier-free formula involving the index variables $i_1,\ldots,i_k$, $j_1,\ldots,j_m$, and the global and array variables in V_N.

In the case of SATS, the regular invariant specifying a safe separation of aircraft is $\forall i,j \in [N] \;:\; (i \neq j \wedge l[i] = B \wedge l[j] = B \wedge next[j] = i) \Rightarrow x[i]-x[j] \geq L_S$. That is, if there is an aircraft i attempting to land, the aircraft immediately ahead of it is at least L_S physical distance away.

In the remainder of this section, we show how inductive invariant assertions for networks of RHA can be stated as LH-assertions. For the purposes of this presentation we assume that there are no global variables. It can be checked in a straightforward manner that these derivations hold for systems with global variables. An assertion ψ is an invariant assertion for the parameterized network A^N if, for all $N \in \mathbb{N}$,

(A) **initiation**: for each state $\mathbf{v} \in \Theta_N \Rightarrow \mathbf{v} \models \psi$,

(B) **transition consecution**: for each $(\mathbf{v}, \mathbf{v}') \in D_N$, $\mathbf{v} \models \psi \Rightarrow \mathbf{v}' \models \psi$, and
(C) **trajectory consecution**: for each $(\mathbf{v}, \mathbf{v}') \in T_N$, $\mathbf{v} \models \psi \Rightarrow \mathbf{v}' \models \psi$.

We derive an LH-assertion for each of the above conditions.

From the definition of the initial states, $\mathbf{v} \in \Theta_N$ iff $\forall\ i \in [N] : \mathbf{v}[i] \models \mathsf{Init}_i$, where recall that Init_i is a formula involving the variables in Var_i. Thus, condition A is equivalent to checking:

$$(\forall\ i \in [N] : \mathsf{Init}_i) \Rightarrow (\forall\ i_1, \ldots, i_k \in [N] : \exists j_1, \ldots, j_m \in [N] : \varphi)$$

Moving the quantifiers of ψ to the front, we obtain:

$$\begin{aligned} \forall\ i_1, \ldots, i_k \in [N] : \exists\ j_1, \ldots, j_m \in [N] : (\forall\ i \in [N] : \mathsf{Init}_i \Rightarrow \varphi), \\ \forall\ i_1, \ldots, i_k \in [N] : \exists\ i, j_1, \ldots, j_m \in [N] : (\mathsf{Init}_i \Rightarrow \varphi), \end{aligned}$$

which is in the required LH-assertion form.

From the definition of discrete transitions D_N, condition B can be written:

$$\begin{aligned} (\psi \wedge (\exists h \in [N] : \exists \mathsf{a} \in \mathsf{Act} : \mathbf{pre}(\mathsf{a}, h)\ \wedge\ \mathbf{eff}(\mathsf{a}, h)) \\ \wedge \forall g \in [N] :\ g \neq h \Rightarrow \mathbf{id}(g)]) \Rightarrow \psi'. \end{aligned}$$

Here $\mathbf{id}(g)$ is a shorthand for the formula $\bigwedge_{x \in \mathsf{Var}_g} x'[g] = x[g]$ and ψ' is the formula obtained by replacing each variable in ψ with its primed version. Moving the quantifier to the front and rearranging we obtain the LH-assertion:

$$\forall h, \mathsf{a} : \exists g : (\psi\ \wedge\ \mathbf{pre}(\mathsf{a}, h)\ \wedge\ \mathbf{eff}(\mathsf{a}, h) \wedge\ (g \neq h \Rightarrow \mathbf{id}(g)) \Rightarrow \psi').$$

Exposing the quantifier in ψ and ψ':

$$\begin{aligned} \forall h, \mathsf{a} : \exists k : ((\forall\ i_1, \ldots, i_k : \exists\ j_1, \ldots, j_m : \varphi) \wedge\ \mathbf{pre}(\mathsf{a}, h)\ \wedge\ \mathbf{eff}(\mathsf{a}, h) \\ \wedge\ (g \neq h \Rightarrow \mathbf{id}(g)) \Rightarrow (\forall\ i'_1, \ldots, i'_k : \exists\ j'_1, \ldots, j'_m : \varphi)). \end{aligned}$$

Moving quantifiers to the front of the formula, we obtain:

$$\begin{aligned} \forall h, \mathsf{a}, j_1, \ldots, j_m, i'_1, \ldots, i'_k : \exists\ g, i_1, \ldots, i_k, j'_1, \ldots, j'_m : \\ ((\varphi\ \wedge\ \mathbf{pre}(\mathsf{a}, h)\ \wedge\ \mathbf{eff}(\mathsf{a}, h) \wedge\ (g \neq h \Rightarrow \mathbf{id}(g))) \Rightarrow \varphi). \end{aligned}$$

As a is universally quantified over the finite set of actions Act_i, it is removed by writing the above as a finite conjunction of LH-assertions.

Finally, by the definition of trajectories T_N, condition C can be written as:

$$\begin{aligned} \psi \wedge (\exists t_1 \in \mathbb{R} : \forall h \in [N], t_2 \in \mathbb{R} : \exists \mathsf{m} \in \mathsf{Loc} : \mathbf{inv}(\mathsf{m}, h) \wedge (\mathbf{stop}(\mathsf{m}, h) \Rightarrow t_2 = t_1) \\ \bigwedge_{x \in \mathsf{Var}_h^c} x[h] + t_1 \mathbf{lflowrate}(\mathsf{m}, h, x) \leq x'[h] \leq x[h] + t_1 \mathbf{uflowrate}(\mathsf{m}, h, x)) \\ \Rightarrow \psi', \end{aligned} \tag{5}$$

where **inv** and **stop** have all continuously updated real array variables replaced with primed versions using a time-elapse of t_2, and Var_h^c are the continuously updated real array variables. The conversion of Equation 5 to an LH-assertion is essentially the same as the discrete case, but more tedious. The easiest way to do this is first to convert to prenex normal form. In summary, these derivations show how we can check inductive invariants as LH-assertions for networks of RHA using the small model result introduced above.

4 Passel: Tool Implementation and Results

We developed a prototype tool called Passel—which is a large group of people or things of indeterminate number—for verifying that properties are inductive invariants. Passel utilizes the satisfiability modulo theories (SMT) [17] solver Z3 [16] to prove validity of an LH-assertion by checking unsatisfiability of the negation of the assertion. Our program is written in C# and we used the managed .NET API to version 3.2 of Z3. The input to Passel comes from the HyXML format for describing hybrid automata developed for Hylink [33]. We configured Z3 to use a variety of options, in particular, our method requires either having model-based quantifier instantiation (MBQI) enabled or quantifier elimination enabled, as otherwise we may receive unknown as a response from Z3 for some satisfiability checks. Within Z3, we model array variables of processes as functions mapping a subset of the integers (i.e., the set of process indices) to the type of the variable. We did not need to use any special encoding to represent our systems for Z3, so the queries we ask are almost exactly the same as the formulas appearing in Subsection 3.1. Given the finite bound N_0 from Lemma 1, we could potentially have composed the system for each instantiation $2 \leq N \leq N_0$ and used existing tools (for instance, HyTech [28] or PHAVer [22]), but the LH-assertions specifications were more natural to state in an environment that allows quantifiers. Additionally, our prior experience in model checking such parameterized systems [31] indicated that the bound allowed in practice due to memory requirements may be less than the bound N_0, and may prevent verification.

The operation of checking an inductive invariant is as follows. The user specifies a set $\mathcal{I}$ of candidate inductive invariants. Based on the protocol specification, we receive from Lemma 1 a bound N_0 on the number of processes N for which we must check each property. Many of the invariant properties we are interested in are not inductive (e.g., mutual exclusion in Fischer's protocol is not inductive, nor even k-inductive [20]), so having a set of candidate invariants allows us to discharge each until we have a set of proven lemmas that imply the desired invariant. For each candidate invariant $I \in \mathcal{I}$, we check if I is an inductive invariant by attempting to prove I' after each transition, where I' is I with all variables replaced with their primed counterparts (i.e., post-states). If I is successfully proved, we assert I as an assumption lemma and check some other $J \in \mathcal{I}$ for $J \neq I$ until we have proved—or failed to prove—each property in $\mathcal{I}$. We emphasize that if we do not prove a property, it does not necessarily mean that the property does not hold, only that it may not be inductive.

The main difficulty is specifying a rich enough set of properties $\mathcal{I}$. We perform satisfiability checks with model construction enable, thus if a transition or time elapse violates the candidate property, we record it and display it to the user so she/he may use this information to refine the candidate manually. For Fischer, a detailed refinement is performed in [20], and we used several of these refined properties in the set $\mathcal{I}$. In addition to SATS and Fischer, we considered several

Example	Property	Correct	Time (s)	QI	Buggy	Time (s)	QI
SATS	A	✓	0.47	166	✓	24.429	63
	B	✓	0.591	373	✓	0.595	197
	C	✓	0.586	703	✗	1.041	113485
	D	✓	0.757	8298	✗	1.256	1659
Timed SATS	A	✓	0.349	66	✓	0.34	61
	B	✓	0.304	673	✓	0.317	460
	C	✓	0.244	373	✗	1.763	140512
	E	✓	0.467	3032	✗	2.204	26958
Fischer	a	✓	0.498	305	✓	0.491	305
	b	✓	0.33	204	✓	0.325	204
	c	✓	0.376	544	✓	0.33	544
	d	✓	0.396	618	✗	0.533	2548
	e	✓	0.435	1306	✗	0.532	1202
	f	✓	0.414	1036	✗	0.437	3162

Table 1. Example properties and results for $2 \leq N \leq 100$, which exceeds the threshold from Lemma 1 for each property and example. SATS properties are shown in Subsection 2.3 and Fischer properties are shown in the last paragraph of Section 4. A check in the "Correct" column means that a property was shown to be an inductive invariant, while an "X" indicates not, and similarly for buggy versions of the protocols as indicated in the "Buggy" column. Time is the runtime in seconds. QI is the number of quantifier instantiations. The results in this table had both MBQI and quantifier elimination enabled.

other examples[2] We performed the verification on a laptop with a quad-core Intel I7 2.0GHz processor and 8GB RAM.

The buggy version of SATS replaces the precondition $l[next[i]]$ with $l[last]$ and $x[next[i]]$ with $x[last]$ in Figure 1, line 24, which ensures the spacing between i and the last aircraft is large enough. However, this may not be the aircraft immediately ahead of i, for instance, if two aircraft have moved to the base, so the safe separation properties do not hold. The correct version of Fischer's mutual exclusion protocol has a constraint $A < B$, and the buggy version has $A \geq B$ [20]. We checked the following properties for Fischer (see also Table 1):

(a) $\forall i, j \in [N] : x[i] = x[j]$,
(b) $\forall i \in [N] : q[i] = set \Rightarrow last[i] \leq x[i] + A$,
(c) $\forall i \in [N] : q[i] = set \Rightarrow x[i] \leq last[i]$,
(d) $\forall i, j \in [N] : (q[i] = check \wedge g = i \wedge q[j] = set) \Rightarrow first[i] > last[j]$,
(e) $\forall i, j \in [N] : q[i] = crit \Rightarrow (g = i \wedge q[j] = set)$, and
(f) $\forall i, j \in [N] : (i = j) \Rightarrow (q[i] = crit \vee q[j] = crit)$,

where f specifies mutual exclusion.

[2] Passel and case study specification files are available from: http://www.taylortjohnson.com/research/forte2012/.

5 Conclusion and Future Work

In this paper, we developed a small model theorem for networks of rectangular hybrid automata (RHA), and used the theorem to establish inductive invariant properties for several case studies. To the best of our knowledge, this is the first positive result on automatic parameterized verification of hybrid automata, beyond previous results for timed automata [4,3,25,14,21,13]. The modeling framework and process of inductive invariant checking are amenable to automation, so we implemented a prototype called Passel using the SMT solver Z3. We plan to continue development of Passel, and investigate other methods of verifying such RHA networks like automated abstraction and invariant generation. One weakness of the current method is that the user is required to specify all the inductive invariants that will imply the desired invariant property (for instance, mutual exclusion in Fischer is not an inductive invariant, so one must find a stronger property that implies mutual exclusion and is an inductive invariant; a similar process was necessary for the SATS example). While we aid the user in this task by providing counterexample models, she/he must still manually perform the strengthening, so methods to automatically generate or strengthen invariants—such as invisible invariants [37,8,9]—or k-induction [12,19] would be interesting to investigate. It will also be interesting to investigate parameterized verification for hybrid automata with linear or nonlinear dynamics.

Acknowledgments. This paper is based upon work supported by the National Science Foundation under CAREER Grant No. 1054247. The authors thank the anonymous reviewers for their suggestions that helped improve the presentation of this paper.

References

1. Abbott, T.S., Jones, K.M., Consiglio, M.C., Williams, D.M., Adams, C.A.: Small aircraft transportation system, higher volume operations concept: Normal operations. Tech. Rep. NASA/TM-2004-213022, NASA (Aug 2004)
2. Abdulla, P., Delzanno, G., Rezine, A.: Parameterized verification of infinite-state processes with global conditions. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification, LNCS, vol. 4590, pp. 145–157. Springer (2007)
3. Abdulla, P.A., Deneux, J., Mahata, P.: Multi-clock timed networks. In: Proc. of 19th Annual IEEE Symposium Logic in Computer Science. pp. 345–354 (Jul 2004)
4. Abdulla, P.A., Jonsson, B.: Model checking of systems with many identical timed processes. Theoretical Computer Science 290(1), 241–264 (2003)
5. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
6. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) Hybrid Systems. pp. 209–229. Springer-Verlag, London, UK (1993)

7. Apt, K.R., Kozen, D.C.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. 22(6), 307–309 (1986)
8. Arons, T., Pnueli, A., Ruah, S., Xu, Y., Zuck, L.: Parameterized verification with automatically computed inductive assertions? In: Berry, G., Comon, H., Finkel, A. (eds.) Computer Aided Verification, LNCS, vol. 2102, pp. 221–234. Springer (2001)
9. Balaban, I., Fang, Y., Pnueli, A., Zuck, L.: Iiv: An invisible invariant verifier. In: Computer Aided Verification, LNCS, vol. 3576, pp. 293–299. Springer (2005)
10. Bjørner, N.: Linear quantifier elimination as an abstract decision procedure. In: Automated Reasoning, LNCS, vol. 6173, pp. 316–330. Springer (2010)
11. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Springer (2001)
12. Brown, G., Pike, L.: Easy parameterized verification of biphase mark and 8n1 protocols. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 3920, pp. 58–72. Springer (2006)
13. Bruttomesso, R., Carioni, A., Ghilardi, S., Ranise, S.: Automated analysis of parametric timing based mutual exclusion protocols. In: Proc. of 4th NASA Formal Methods Symposium (NFM). Springer (2012)
14. Carioni, A., Ghilardi, S., Ranise, S.: MCMT in the land of parameterized timed automata. In: In Proc. of VERIFY 2010 (Jul 2010)
15. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
16. De Moura, L., Bjørner, N.: Z3: an efficient smt solver. In: Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag (2008)
17. De Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Commun. ACM 54, 69–77 (Sep 2011)
18. Delzanno, G.: Automatic verification of parameterized cache coherence protocols. In: Emerson, E., Sistla, A. (eds.) Computer Aided Verification, LNCS, vol. 1855, pp. 53–68. Springer Berlin / Heidelberg (2000)
19. Donaldson, A., Haller, L., Kroening, D., Rmmer, P.: Software verification using k-induction. In: Yahav, E. (ed.) Static Analysis, LNCS, vol. 6887, pp. 351–368. Springer Berlin / Heidelberg (2011)
20. Dutertre, B., Sorea, M.: Timed systems in sal. Tech. Rep. SRI-SDL-04-03, SRI International (Oct 2004)
21. Faber, J., Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V.: Automatic verification of parametric specifications with complex topologies. In: Integrated Formal Methods, LNCS, vol. 6396, pp. 152–167. Springer (2010)
22. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: HSCC. pp. 258–273 (2005)
23. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Computer Aided Verification (CAV). LNCS, Springer (2011)
24. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 5643, pp. 306–320. Springer Berlin / Heidelberg (2009)
25. Grinchtein, O., Leucker, M.: Network invariants for real-time systems. Formal Aspects of Computing 20, 619–635 (2008)
26. Hanna, Y., Samuelson, D., Basu, S., Rajan, H.: Automating cut-off for multi-parameterized systems. In: Dong, J., Zhu, H. (eds.) Formal Methods and Software Engineering, LNCS, vol. 6447, pp. 338–354. Springer Berlin / Heidelberg (2010)

27. Henzinger, T.A.: The theory of hybrid automata. In: IEEE Symposium on Logic in Computer Science (LICS). p. 278. IEEE Computer Society, Washington, DC, USA (1996)
28. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: a model checker for hybrid systems. Journal on Software Tools for Technology Transfer 1, 110–122 (1997)
29. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? Journal of Computer and System Sciences 57, 94–124 (1998)
30. Johnson, T.T., Mitra, S.: Safe flocking in spite of actuator faults using directional failure detectors. Journal of Nonlinear Systems and Applications 2(1-2), 73–95 (Apr 2011)
31. Johnson, T.T., Mitra, S.: Parameterized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In: ACM/IEEE 3rd International Conference on Cyber-Physical Systems (Apr 2012)
32. Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) Formal Methods. LNCS, Springer (2011)
33. Manamcheri, K., Mitra, S., Bak, S., Caccamo, M.: A step towards verification and synthesis from simulink/stateflow models. In: Proc. of the 14th Intl. Conf. on Hybrid Systems: Computation and Control. pp. 317–318. ACM (2011)
34. Muñoz, C., Carreño, V., Dowek, G.: Formal analysis of the operational concept for the small aircraft transportation system. In: Butler, M., Jones, C., Romanovsky, A., Troubitsyna, E. (eds.) Rigorous Development of Complex Fault-Tolerant Systems, LNCS, vol. 4157, pp. 306–325. Springer Berlin / Heidelberg (2006)
35. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Computer Science Logic. LNCS, vol. 6247, pp. 469–483 (2010)
36. Platzer, A.: Quantified differential invariants. In: Proc. of the 14th ACM Intl. Conf. on Hybrid Systems: Computation and Control. pp. 63–72. ACM (2011)
37. Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 2031, pp. 82–97. Springer (2001)
38. Umeno, S., Lynch, N.: Safety verification of an aircraft landing protocol: A refinement approach. In: Hybrid Systems: Computation and Control, LNCS, vol. 4416, pp. 557–572. Springer (2007)
39. Viken, S., Brooks, F.: 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)
40. Wolper, P., Lovinfosse, V.: Verifying properties of large sets of processes with network invariants. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems, LNCS, vol. 407, pp. 68–80. Springer Berlin / Heidelberg (1990)