Passel: A Verification Tool for Parameterized Networks of Hybrid Automata

Taylor T. Johnson and Sayan Mitra

Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA,
taylor.johnson@acm.org, mitras@illinois.edu,
WWW home page: http://www.taylortjohnson.com

Abstract. The Passel verification tool for parameterized networks of hybrid automata is presented in this paper. Passel automatically proves safety properties of networks of arbitrarily many interacting copies of a template hybrid automaton with rectangular dynamics by using a combination of invariant synthesis and inductive invariant proving. The invariant synthesis method generates quantified inductive invariants by transforming the set of reachable states of finite instantiations of the network. This is an extension to hybrid automata of the project-and-generalize method used in the invisible invariants method for synthesizing inductive invariants for parameterized networks of discrete automata. We use this extended method in a fixed-point iteration that is complete in the sense that it is ensured to generate an inductive invariant of a certain class of assertions for the parameterized network of hybrid automata. We present some of the engineering and design choices made in developing Passel, and present promising experimental results where the invariant synthesis procedure has been useful in automatically proving safety properties of examples like Fischer’s mutual exclusion protocol (with rectangular dynamics instead of clocks), a conceptual air-traffic control protocol, and others.

Keywords: hybrid systems, invariant synthesis, parameterized systems

1 Introduction

Hybrid automata [5,19] combine continuous and discrete evolution and have become a standard formalism for modeling software systems interacting with the physical world. In systems where many nearly identical automata interact, the hybrid network model [18] preserves the symmetry arising from the repeated structure. Systems exhibiting this pattern abound around us—from MAC protocols, air-traffic control systems, real-time distributed algorithms, to control systems for robotic swarms and cell arrays in tissue. In a hybrid network, each automaton is an instance of a hybrid automaton template that interacts with others only through shared discrete transitions (and not through continuous signals). The complete system has both discrete and continuous dynamics, and the communication topology between automata may itself evolve over time.
In this paper, we present Passel, a software tool that embodies an automatic safety verification technique for hybrid networks of arbitrary size. Given a safety property $\zeta(N)$ parameterized by the size $N$ of the network, and a rectangular hybrid automaton template $A(i)$, Passel attempts to verify that for any natural number $N$, the parameterized network of size $N$ obtained by the parallel composition $A(1) \parallel \ldots \parallel A(N)$ satisfies $\zeta(N)$. The core of Passel has procedures for finding and checking inductive invariants for hybrid networks of arbitrary size. For checking quantified inductive invariants, Passel uses quantifier elimination and instantiation, or exploits small model properties of the inductive invariant assertions of hybrid networks [18]. For finding invariants of hybrid networks, Passel builds upon the invisible invariant method used for discrete transition systems [6,26,8,23,21].

The invisible invariant method starts by computing the set of reachable states for a small instantiation of the network. Say for $N = 3$, the reach set (or its approximation) for the network $A(1) \parallel A(2) \parallel A(3)$ is computed. Then this set is projected onto a smaller instance of size $P < N$. Finally, this projected subset is generalized to produce a candidate invariant for a network of arbitrary size. The choice of $P$ determines the shape of the generated invariant. For $P = 1$ the invariant asserts properties about the variables of a single automaton, for $P = 2$ the properties may include linear inequalities involving pairs of automata, and so forth. In our methodology, the user may choose the projection to be made onto a subset of the variables of the automata in the $P$-sized network, such as only the real or discrete variables. This choice proves to be crucial in some of the case studies. If the generated candidate invariant is inductive and sufficient to prove the property $\zeta(N)$, then a completely automatic inductive invariance proof is obtained.

The project-generalize method is incomplete even for discrete systems [26]. The candidate invariants generated by our method may not be inductive nor are they guaranteed to prove $\zeta(N)$. Using the above project-generalize method as a subroutine we present an algorithm which is guaranteed to terminate with either an automated proof or a potential counterexample instance. The algorithm is guaranteed to generate an inductive invariant with a given number of universally quantified automata indices ($P$) for arbitrarily large networks, provided the the reach set computation is performed over a large enough network and the component automata admit exact reach set computations. The algorithm implemented in Passel produced the first fully automatic proof of correctness for several nontrivial hybrid networks. Notable among these are the core of the NASA-developed SATS air-traffic control protocol [1,22,17]. Another example verified automatically is Fischer’s mutual exclusion protocol with drifting

---

1 Rectangular hybrid automata admit continuous dynamics of the form $\dot{x} \in [a,b]$, which can exactly describe continuous variables with constant slope, such as drifting clocks. Rectangular dynamics can also approximate more complex linear and nonlinear differential equations arbitrarily closely up to some bounded time.
The synthesis method found non-trivial invariants that implied collision avoidance in the first example and mutual exclusion in the latter. Extending the invisible invariant method to hybrid automata required overcoming several conceptual and technical challenges. First, unlike the methods for discrete models [6,26,8,23,21] which primarily used BDDs for representing states (as implemented in TLV [27]), modeling real-valued variables and their continuous evolution, we require a symbolic representation for expressing multiple types of variables and state updates involving real arithmetic. In Passel, the states, transitions, and continuous trajectories are represented using satisfiability modulo theories (SMT) formulas. This is made possible by our observation that for rectangular hybrid automata with convex invariants and stopping conditions, the (possibly nondeterministic) trajectories can be encoded by transition rules that involve a finite number of existentially quantified real-valued variables. With this representation, inductive invariant checks become satisfiability queries which are implemented in Passel using the Z3 SMT solver [12]. Further, a small model theorem can be used to discharge the proof obligations with finite instances [18]. Passel uses the hybrid automata model checker PHAVer [13] for computing the reach sets for finite instances of the network. Second, the reach set of a finite instance is encoded in a disjunctive normal form (DNF) formula which grows exponentially with the number of instances in the network, as well as with the number of discrete locations and continuous variables of each automaton component. Naively checking satisfiability of these formulas becomes infeasible beyond the simplest of examples. To overcome this, we exploited logical equivalences—such as existential quantification distributing over disjunction—to decompose the problem into smaller, equivalent representations of the reach set encoding. This made it possible to compute projections and generalizations of different pieces of the invariant separately, which are then combined together in the final step.

## 2 Hybrid Automata Networks

A network of hybrid automata \( \mathcal{A}^N \) is a collection of \( \mathcal{A} \) interacting copies of a template automaton \( \mathcal{A}(N, i) \). Here \( N \) is an arbitrary natural number which defines the size of the network and \( i \) is an element in the set \( [N] \equiv \{1, \ldots, N\} \), which identifies the \( i^{th} \) member of the network. We drop the argument \( N \) from \( \mathcal{A}(N, i) \) and write \( \mathcal{A}(i) \) when it is clear from context.

An individual hybrid automaton, say \( \mathcal{A}(N, 3) \) (or \( \mathcal{A}(3) \)), is a (possibly nondeterministic) state machine with finitely many discrete locations and real-valued variables. The state of \( \mathcal{A}(N, i) \) can change instantaneously through discrete transitions and its real-valued variables can evolve continuously over time according to trajectories specified by ordinary differential equations or inclusions. In a network \( \mathcal{A}^N \) with \( N \) automata, the constituent automata may communicate over

---

2 Previous parameterized verification of Fischer’s protocol assumed clocks evolving at unit rate \( \dot{x} = 1 \) [4,11], while we model rectangular dynamics \( \dot{x} \in [1 - \rho, 1 + \rho] \).
parameter name='A' type='real' value = 5.0 // smaller timing parameter
parameter name='B' type='real' value = 35.0 // larger timing parameter
parameter name='ub' type='real' value = 2.0 // upper clock rate

automaton name='Fischer'
variable name=q[i] type='L' // control location local variable
variable name=x[i] type='real ' // continuous local variable
variable name=g type='index' // global lock variable

location name='rem'
flowrate: x[i] _dot = 0.0
stop: x[i] = A
flowrate: x[i] _dot >= lb and x[i] _dot <= ub

location name='try'
inv: x[i] <= A
flowrate: x[i] _dot >= lb and x[i] _dot <= ub

location name='wait'
flowrate: x[i] _dot = 0.0

location name='cs'
flowrate: x[i] _dot = 0.0

transition from='rem' to='try'
grd: g = ⊥
eff: x[i] = 0.0

transition from='try' to='wait'
eff: g' = i and x[i]' = 0.0

transition from='wait' to='cs'
grd: g = i and x[i] >= B
eff: x[i] = 0.0

transition from='wait' to='rem'
grd: g != i and x[i] >= B
eff: x[i] = 0.0

transition from='cs' to='rem'
eff: g' = ⊥ and x[i]' = 0.0

property: forall i j ((i != j and q[i] = cs) implies (q[j] != cs))
initially: forall i (q[i] = rem and x[i] = 0 and g = ⊥)

Fig. 1. Passel input file specifying $A(N,i)$ for Fischer’s mutual exclusion algorithm.
The specification of $A(i)$ may use a set of symbolic or numerical parameters (constants). Each parameter is specified by its name, type, and optionally by a set of constraints it must satisfy. For the example Fischer, there are four real-valued parameters, $A, B, lb, ub \in \mathbb{R}$ (lines 1, 2, 3, and 4). A hybrid automaton $A(i)$ is specified by the following components: (a) a finite set of location names $L$, (b) an initial condition assertion $Init$, (c) a finite set of variable names $V_i$ with their associated types, (d) a finite set of guarded transition statements $Trans$, and (e) a finite set of trajectory statements $Flow$. The subscript $i$ emphasizes that components may use the automaton’s index.

Variables, Locations, and Initialization. Each variable $v$ in the set of variables $V_i$ is associated with a type, denoted $type(v)$, that defines the values the variable may take. A variable may be local with a name of the form $(\text{variable}_\text{name})[i]$, or global, in which case the name does not have the index $[i]$. The type may be (a) the set $L$ of locations, (b) the set $[\mathbb{N}] \perp \triangleq [\mathbb{N}] \cup \{\perp\}$ of automaton indices (identifiers) with the special element $\perp$ that is not equal to any index, or (c) the set $\mathbb{R}$ of real numbers. For automaton Fischer with index $i$, the set of variables is specified by the list of variable names. It has two local variables $q[i], x[i]$ with types $L$ (specified below) and $\mathbb{R}$, and a single global variable $g$ of type $[\mathbb{N}] \perp$ (lines 7, 8, and 9, respectively). We denote the set of local variable names by $V_L$, the set of global variables by $V_G$, and the set of parameters by $V_P$.

The set of locations is specified by the list of location names. In this example, the set of locations $L$ is $\{\text{rem, try, wait, cs}\}$ (lines 11, 13, 17, and 19).

The initial condition assertion $Init$ is a predicate involving the variables in all the automata in the network and it defines the set of initial valuations of the variables. In the example, the initial assertion $Init$ (line 37) specifies that for each automaton $i \in [\mathbb{N}]$, $q[i] = \text{rem}$, and $x[i] = 0$ and the global variable $g = \perp$. Assertions with universally quantified index variables may also be specified for desired properties or the initial conditions. For example, we may specify a mutual exclusion property as (line 36):

$$\text{for all } i j (i \neq j \text{ and } q[i] = \text{cs}) \implies (q[j] \neq \text{cs}).$$

Here $i$ and $j$ are implicitly in the set $[\mathbb{N}]$.

Transitions. The set of guarded transitions is specified by the list of transition from-to statements. Each transition $t \in Trans_i$ specifies a guard following the keyword $\text{grd}$ and an effect following the keyword $\text{eff}$. The guard is a predicate involving the variables in $V_i$ and it specifies the enabling condition for the transition. The effect is a predicate involving the variables and primed versions of the variables (for example, $x[i']$). It specifies a relation between the valuation of the variables before and after the transition occurs. In the following sections, we will denote the guard and the effect of transition $t$ of automaton $A(i)$ by $\text{grd}(t, i)$ and $\text{eff}(t, i)$.

The transition from $\text{wait}$ to $\text{cs}$ for Fischer with index $i$ (line 28), where

$$\text{grd} : g = i \text{ and } x[i] > B, \text{ and } \text{eff} : x[i'] = 0.0$$
specifies that this automaton may transition from $q[i] = \text{wait}$ to $q[i]' = \text{cs}$ only if the global index-valued variable $g$ is equal to $i$ and the local real-valued variable $x[i]$ is at least as large as the parameter value $B$. Further, if the automaton does make this transition then $x[i]$ is reset to 0. If a guard condition is omitted for a transition then it is assumed to be just the control location condition. For example, the transition from $\text{try}$ to $\text{wait}$ is enabled when $q[i] = \text{try}$.

Trjectories. The set of trajectory statements are listed following the corresponding location names. Each location $m \in L$ has a trajectory statement in $\text{Flow}_i$ which consists of an invariant following the keyword $\text{inv}$, a stopping condition following the keyword $\text{stop}$, and a sequence of flow rate inequalities following the keyword $\text{flowrate}$. The first two are predicates involving the variables in $V_i$ (as well as real-valued numerical constants). If no invariant is specified, it is assumed to be true. If no stopping condition is specified, it is assumed to be false. The flow rate associates each real-valued variable in $V_i$ with an upper and a lower bound in terms of a numerical constant or a parameter name.

Together, the components of a trajectory statement define how variables of $A(i)$ behave over intervals of time. For example, the statement for $\text{wait}$ is:

\[
\text{flowrate} : x[i] \cdot \dot{} \geq lb \text{ and } x[i] \cdot \dot{} \leq ub,
\]

which specifies that while $q[i] = \text{wait}$ the local variable $x[i]$ evolves with time at a rate that is upper and lower-bounded by $ub$ and $lb$. If $lb = ub$ then $x[i]$ models a continuous variable with a constant slope, and if they are equal to one, then $x[i]$ models a perfect clock. The trajectory statement for $\text{try}$:

\[
\begin{align*}
\text{inv} & : x[i] \leq A \\
\text{stop} & : x[i] = A \\
\text{flowrate} & : x[i] \cdot \dot{} \geq lb \text{ and } x[i] \cdot \dot{} \leq ub
\end{align*}
\]

specifies the same differential constraints on $x[i]$ while $q[i] = \text{try}$. In addition, its invariant requires that the automaton with index $i$ can have $q[i] = \text{try}$ only as long as $x[i] \leq A$. Its stopping condition requires that if $x[i] = A$, then time cannot continue to elapse\(^4\).

In the following sections, the invariant of a location $m$ and automaton $A(i)$ will be denoted by $\text{inv}(m, i)$, the stopping condition will be denoted by $\text{stop}(m, i)$, and the upper and the lower bounds of the flow inequalities on $x[i] \cdot \dot{}$, for some real-valued variable $x[i] \in V_i$ will be denoted by $u(m, x[i])$ and $l(m, x[i])$.

### 2.2 Semantics of Hybrid Automata Networks

Given the specification of one hybrid automaton $A(N, i)$, the semantics of the hybrid automata network $A^N$ is defined in this section. The semantics is defined

\(^4\)In this example, the stopping condition is redundant. However, if the flow equations allow time to elapse while the continuous state remains at the boundary of the invariant condition, the stopping condition allows for modeling urgent transitions. The stopping condition forces time to stop, which can force discrete transitions.
in terms of a transition system with set of states \(Q^N\), initial states \(\Theta^N\), and transitions relation \(T^N\).

**States.** The state space \(Q^N\) is the set of all possible valuations of all the local and global variables of all automata in the network. The state space \(Q_i\) of an individual automaton \(\mathcal{A}(N, i)\) is the Cartesian product of all the types of the local variables \(V_L\) and global variables \(V_G\). The parameters remain constant and are excluded from the state-space. The state space \(Q^N\) of the network \(\mathcal{A}^N\) is the product space of \(N\) copies of \(\mathcal{A}(N, i)\)'s local variable valuations with a single copy of the global variable valuations. For the example, the state-space for the Fischer network with \(N\) automata is \(L^N \times \mathbb{R}^N \times [N]_L\), which corresponds to the valuations of the \(N\) \(q[i]\) variables, the \(N\) \(x[i]\) variables, and the single global variable \(g\).

Subsets of \(Q^N\) are often represented by formulas involving the variables. For such a formula \(\phi\), the corresponding states satisfying \(\phi\) is denoted by \([\phi]\). Elements of the state space \(Q^N\) are called *states* and are denoted by boldface \(v\), \(v'\), etc. At a state \(v\), the valuation of a particular local variable \(x[i] \in V_L\) for automaton \(i\) is denoted by \(v.x[i]\), and \(v.g\) for some global variable \(g \in V_G\). If a state satisfies a formula \(\phi\)—that is, the corresponding variable valuations result in \(\phi\) evaluating to true—we write \(v \models \phi\). The valuation of all the local variables for automaton \(\mathcal{A}(i)\) at state \(v\) is denoted by \(v[i]\).

The set of initial states \(\Theta^N \subseteq Q^N\) is defined as \([\text{Init}]\), that is, the set of states satisfying the formula \(\text{Init}\). In the example, the set of initial states specified by line 37 is

\[
\Theta^N \doteq [\text{Init}] = \{v \in Q^N \mid \forall i \in [N]v.q[i] = \text{rem} \land v.x[i] = 0.0 \land g = \perp\}.
\]

**Transitions and Trajectories.** The evolution of the states of \(\mathcal{A}^N\) are describing by a transition relation \(T^N \subseteq Q^N \times Q^N\). For a pair \((v, v') \in T^N\), we use the notation \(v \rightarrow v'\), where \(v\) is called the *pre-state* and \(v'\) is called the *post-state*. There are two kinds of transitions in \(T^N\): discrete transitions describe instantaneous change of state and trajectories describe change of state after a time interval. There is a discrete transition \(v \rightarrow v'\) iff:

\[
\exists i \in [N] \exists t \in \text{Trans} : v[i] \models \text{grd}(t, i) \land v[i]' \models \text{eff}(t, i) \land \\
(\forall j \in [N] : j \neq i \implies v'[j] = v[j]).
\]

From the pre-state \(v\), any automaton in the network, with any transition satisfying the guard *may* update its post-state according to transition effect, while the states of the other automata remain unchanged.

There is a trajectory \(v \rightarrow v'\) iff some amount of time that can elapse from \(v\) after which the states of all the automata in the network is updated to \(v'\) according to their individual flow constraints. To define this formally, we first define the function \(\text{flow}(m, v[i], \delta(m), t)\) that returns the state of \(\mathcal{A}(i)\) with \(q[i] = m\) that can be reached from \(v[i]\) in \(t\) time for particular choices of the flow rates \(\delta(m)\) for each of the \(n\) real-valued variables of \(\mathcal{A}(i)\). Concretely, \(\delta(m)\) is
an n-dimensional vector and its component for variable $x[i]$, $\delta(m, x[i])$, is in the interval $[l(m, x[i]), u(m, x[i])]$ defined by the upper and lower flow rates for $x[i]$ at location $m$. And $v'[i] = \text{flow}(m, v[i], \delta(m), t)$ iff (a) for each real-valued $x[i] \in \mathcal{V}_i$, $v'.x[i] = v.x[i] + \delta(m, x[i]) \times t$, and (b) for each non-real $y[i] \in \mathcal{V}_i$, $v'.y[i] = v.y[i]$.

There is a trajectory $v \rightarrow v'$ iff

$$\exists t_1 \in \mathbb{R}_{\geq 0} \forall i \in [N] \exists m \in L, \delta(m) \in [l(m), u(m)] \forall t_2 \leq t_1 :$$

$$\text{flow}(m, v[i], \delta(m), t_2) \models \text{inv}(m, i) \land$$

$$(\text{flow}(m, v[i], \delta(m), t_2) \models \text{stop}(m, i) \Rightarrow t_2 = t_1) \land$$

$$v'[i] = \text{flow}(m, v[i], \delta(m), \delta t_1).$$

For each $i \in [N]$ and each real variable $x \in \mathcal{V}_i$, $v[i].x$ must evolve to the valuations $v'[i].x$, in exactly $t_1$ time in some location $m \in L$ according to the flow rates allowed for $x$ in that location. All intermediate states along the trajectory must also satisfy the invariant $\text{inv}(m, i)$, and if an intermediate state satisfies $\text{stop}(m, i)$, then that state must be $v'$ (that is, the end of a trajectory).

**Executions, Invariants, and Safety.** An execution of the network $\mathcal{A}^N$ models a particular behavior of the complete system. An execution of $\mathcal{A}^N$ is a sequence of states $\alpha = v_0, v_1, \ldots$ such that $v_0 \in \mathcal{Q}^N$, and for each index $k$ appearing in the sequence $(v_k, v_{k+1}) \in T^N$. A state $x$ is reachable if there is a finite execution ending with $x$. The set of reachable states for $\mathcal{A}^N$ is $\text{Reach}(\mathcal{A}^N)$. The set of reachable states for $\mathcal{A}^N$ starting from an arbitrary subset $v_0 \subseteq Q^N$ (instead of $v_0 \in \mathcal{Q}^N$) is $\text{Reach}(\mathcal{A}^N, v_0)$. An invariant for $\mathcal{A}^N$ is a set of states which contains $\text{Reach}(\mathcal{A}^N)$. In general, any assertion over the variables of the automata in $\mathcal{A}^N$ defines a subset of $Q^N$. The dependence of such assertions on $N$ is made explicit by using names like $\zeta(N)$. A network $\mathcal{A}^N$ is safe with respect to an assertion $\zeta(N)$ if all its reachable states satisfy it, that is, $\text{Reach}(\mathcal{A}^N) \subseteq \llbracket \zeta(N) \rrbracket$. Given $\mathcal{A}(i)$ and a property $\zeta(N)$, Passel attempts to prove that for all $N \in \mathbb{N}$, every network is safe, that is, $\text{Reach}(\mathcal{A}^N) \subseteq \llbracket \zeta(N) \rrbracket$.

### 3 Verification Methodology

Passel attempts to automatically verify safety properties of $\mathcal{A}^N$ that hold for any $N \in \mathbb{N}$ by checking and synthesizing inductive invariants. In Subsection 3.1, we describe a method for checking invariants, and then in Subsection 3.2 present invariant synthesis.

#### 3.1 Proving Inductive Invariants

In the following sections, $N$ and $P$ are constant natural numbers with $P < N$ and $N \geq 2$ (e.g., $P = 2$, $N = 3$), and $N$ is a symbol denoting an arbitrary natural number.

**Definition 1 (Inductive Invariance).** An assertion $\psi(N)$ is an inductive invariant for the parameterized network $\mathcal{A}^N$ if, for all $N \in \mathbb{N}$,
\begin{lstlisting}[language=python]
1  function inductiveInvariance(\(\mathcal{A}(\mathcal{N}, i)\), \(\zeta(\mathcal{N})\), Init, N, P) {
2     // synthesize candidate inductive invariants from finite instances
3     \(\psi(i_1, \ldots, i_P) \leftarrow \text{synthesis}(\mathcal{A}(\mathcal{N}, i), \text{Init}, N, P)\)
4     \(\theta(\mathcal{N}) \leftarrow \forall i_1, \ldots, i_P \in [N]. \psi(i_1, \ldots, i_P)\)
5     // inductive invariance check for any \(\mathcal{N}\)
6     if \((\forall N \in N \text{ Init}(N) \Rightarrow \theta(N) \text{ is valid and})\)
7         \(\forall N \in N \text{ transitionConsecution}(\mathcal{A}(\mathcal{N}, i), N, \theta(N)) \text{ is valid and})\)
8         \(\forall N \in N \text{ trajectoryConsecution}(\mathcal{A}(\mathcal{N}, i), N, \theta(N)) \text{ is valid and})\)
9         \(\forall N \in N \theta(N) \Rightarrow \zeta(N) \text{ is valid})\) {
10            return \(\zeta(\mathcal{N}) \text{ is invariant for all } \mathcal{N}\)
11        }
12    else {
13        return potential counterexample
14    }
15 \}
\end{lstlisting}

Fig. 2. Inductive invariance proof method with auxiliary invariant synthesis. The inputs are an automaton specification \(\mathcal{A}(\mathcal{N}, i)\), a desired safety property \(\zeta(\mathcal{N})\), an initial condition assert \(\text{Init}\), a two constants \(N\) and \(P\). The output is either a proof of the safety property \(\zeta(\mathcal{N})\) for all \(\mathcal{N} \in N\), or a potential counterexample. The latter either indicates \(\mathcal{A}(\mathcal{N}, i)\) has a bug and does not satisfy \(\zeta(\mathcal{N})\) or that the synthesized invariants are not strong enough to prove \(\zeta(\mathcal{N})\).

\((A)\) \textbf{initiation:} for each initial state \(v \in \Theta^\mathcal{N} \Rightarrow v \models \psi(\mathcal{N}).\)

\((B)\) \textbf{transition consecution:} for each transition \((v, v') \in T^\mathcal{N}, v \models \psi(\mathcal{N}) \Rightarrow v' \models \psi(\mathcal{N}), and\)

\((C)\) \textbf{trajectory consecution:} for each trajectory \((v, v') \in T^\mathcal{N}, v \models \psi(\mathcal{N}) \Rightarrow v' \models \psi(\mathcal{N}).\)

This is a verification problem over arbitrary compositions of potentially infinite-state automata, so in general we may need to query a theorem prover to check conditions \(A, B,\) and \(C\). Passel checks these quantified formulas (see Section 4) using quantifier elimination and instantiation procedures. However, if the conditions satisfy the hypothesis of Theorem 1, then Passel discharges the proof obligations for all instantiations up to some fixed sized network \(\mathcal{A}^{N_0}\).

\textbf{Theorem 1 (Small model theorem for LH-assertions [18])}. \textit{Let }\(\theta(\mathcal{N})\text{ be a formula of the form }\forall i_1 \in \mathbb{R} \forall i_1, \ldots, i_k \in [N] \exists t_2 \in \mathbb{R} \exists j_1, \ldots, j_m \in [N]. \psi, \text{ where } \psi \text{ is a quantifier-free formula involving the quantified index variables }i_1, \ldots, i_k, j_1, \ldots, j_m, \text{quantified real variables } t_1 \text{ and } t_2, \text{and variables of } V_i.\text{ Then, }\theta(\mathcal{N})\text{ is valid iff for all } n \leq N_0 = (e + 1)(k + 2), \theta(\mathcal{N})\text{ is satisfied by all }n\text{-models, where } e \text{ is the number of index-valued variables in } \psi \text{ and } k \text{ is the number of universally quantified index variables in } \theta(\mathcal{N}).\)

\subsection{3.2 Synthesizing Inductive Invariants}

If a safety property \(\zeta\) itself is not an inductive invariant for \(\mathcal{A}^\mathcal{N}\), as is often the case, then Passel attempts to find stronger inductive invariants that imply \(\zeta\). In this section, we first present the project-and-generalize subroutine of Passel, and then an algorithm that uses it for synthesizing inductive invariants.
function projectAndGeneralize(AN, θ(N, P), N, P) {
  V ← ∪i∈[N]\[P\]V_i
  R ← Reach(AN, θ(N, P)) // assume in DNF: R = r_1 ∨ r_2 ∨ ... 
  foreach r in R {
    // project onto variables of processes 1, 2, ..., P
    QF[r] ← QuantElim(∃V . r)
    // syntactically substitute 1, 2, ..., P to symbols i_1, ..., i_P
    foreach n in {1, 2, ..., P} {
      QF[r] ← Substitute(QF[r], "n", "i_n")
    }
    // abstract index-valued variable valuations that are > P
    foreach variable v in V_i with type(v) = [N]_⊥ in V_i {
      foreach n in {P + 1, P + 2, ..., N} {
        QF[r] ← Substitute(QF[r], "v = n", "v = i_1 ∧ ... ∧ v ≤ i_P")
      }
    }
    ψ(i_1, ..., i_P) ← V_r∈R QF[r]
  }
  return ψ(i_1, ..., i_P)
}

Fig. 3. Inductive invariant synthesis subroutine. The input arguments are the network AN (previously composed from the specification A(N, i)), a formula θ describing an initial set of states, a constant integer N, and a constant integer P, where N > P. The method computes the set of reachable from θ for a network of N automata, then transforms this reach set into an assertion ψ(i_1, ..., i_P) over the variables of automata with (symbolic) indices i_1, i_2, ..., i_P.

Project-and-Generalize Subroutine. The project-and-generalize subroutine is shown in Figure 3. This subroutine is called with two input parameters N and P. The subroutine first computes the reach set of a network of size N and then through a sequence of transformations generates a candidate invariant for an arbitrary network with P universally quantified index variables.

Reachability Computation (line 3). The reach set Reach(AN) or its over-approximation is computed for the hybrid network AN with N automata. For general hybrid automata, computing the exact reach set is undecidable, however, there are several tools available for computing over-approximations. This step can use any such tool. In the results presented here, Passel uses PHAVer [13] for this step, however, it also supports an SMT-based reach set computation for rectangular hybrid automata. The output of this step is Reach(AN) as a disjunctive normal form (DNF) formula over the variables of A(1), ..., A(N).

Projection of Reach(AN) (loop lines 4 through 17). The loop iterates over each clause r ∈ Reach(AN). Given a clause r in Reach(AN), Passel projects away the variables of any automata with indices greater than P. Recall that P specifies the number of universally quantified index variables in the invariant to be synthesized. Passel computes the projection using quantifier elimination procedures—represented by function QuantElim (line 6)—over the types of the variables V_i.

5 Since existential quantification distributes over disjunction, we consider each clause at a time.
function synthesis(A(N, i), Init(N), N, P) {
  A ← A₁∥A₂∥...∥A₉
  θ(N, P) ← Init(N)
  θold(N, P) ← ⊥ // fixed-point check
  while θ(N, P) ⇒ θold(N, P) is valid {
    ψ(i₁, ..., iₚ) ← projectAndGeneralize(A₉, θ(N, P), N, P)
    θold(N, P) ← θ(N, P)
    θ(N, P) ← ∀i₁, ..., iₚ ∈ [N].ψ(i₁, ..., iₚ)
  }
  return ψ(i₁, ..., iₚ)
}

Fig. 4. Inductive invariant synthesis fixed-point method. The input arguments are an automaton specification \(A(N, i)\), an initial condition assertion \(Init(N)\), a constant integer \(N\), and a constant integer \(P\), where \(P < N\). The fixed-point computation starts with the initial states specified by \(Init\), then iteratively computes and transforms the reachable states of the network \(A^N\) to a fixed-point. The output of the method is a candidate inductive invariant \(ψ(i₁, ..., iₚ)\).

These formulas (predicates over booleans, linear real arithmetic, bounded integers, and their combinations) admit quantifier elimination. Based on the value of \(P\), the quantifier elimination on line 6 is applied to \(QF[r] \triangleq ∃ ∪ i \in [N]\setminus[P] V_i. r\), which projects away the variables of all automata with indices higher than \(P\). In general, Passel projects away some subset of the variables \(V_i\), for example, onto only the variables with discrete types or real types.

For example, in Fischer, one of the clauses in \(Reach(A^N)\) is

\[ r \triangleq (q_1 = \text{wait} \land q_2 = \text{wait} \land -5 \leq -x_1 + x_2 \leq 0 \land g = 2 \land x_2 \geq 0). \]

For \(P = 1\), after executing \(QuantElim\), the variables of automaton 2 are eliminated and we have:

\[ QF[r] \triangleq ∃q_2 \in \mathbb{L}, ∃x_2 \in \mathbb{R}. r \equiv (q_1 = \text{wait} \land 5 \geq x_1 \geq 0 \land g = 2). \]

Generalization of Projected Clauses (lines 8 through 16). Next, the Substitute function syntactically substitutes expressions in \(QF[r]\). The generalization syntactically replaces all valuations of index variables equal to a value in \([P]\) with fresh index symbols \(i_1, i_2, ..., i_P\) (lines 8 through 10). Continuing the example, \(r, 1\) is replaced with \(i_1\) yielding

\[ QF[r] \triangleq (q_{i_1} = \text{wait} \land 5 \geq x_{i_1} \geq 0 \land g = 2). \]

The valuations of index-valued variables in \(Reach(A^N)\) which exceed \(P\) are transformed to symbols equal (or not equal) to \(i_1, ..., i_P\) (lines 12 through 16). In the example, \(QF[r]\) has index 2 for valuations of the index-valued global variable \(g\) after projection and replacing 1 with \(i_1\). Passel abstracts such valuations, by looking at each index-valued variable \(v\), if the valuation \(v = k\) where \(k \leq P\),
then set \( v = i_k \), and otherwise for \( k > P \) or \( k = \perp \), set \( v \neq i_1 \land \ldots \land v \neq i_P \).

Continuing the example \( r \), we have

\[
QF[r] \triangleq (q_{i_1} = \text{wait} \land 5 \geq x_{i_1} \geq 0 \land g \neq i_1),
\]

which does not contain any indices other than the symbols \( i_1, \ldots, i_P \).

Combining Clauses. Following these transformations of all \( r \)'s, Passel takes the disjunction of \( QF[r] \) for all \( r \in R \) (line 18). This is the formula \( \psi(i_1, \ldots, i_P) \). A quantified formula is then created as (Figure 4, line 9):

\[
\theta(N, P) = \forall i_1, i_2, \ldots, i_P \in [N]. \psi(i_1, i_2, \ldots, i_P),
\]

where \( \forall \) indicates that all the quantified indices are distinct.

In summary, each iteration of the loop Figure 4, lines 6 through 10, which calls the function \textit{projectAndGeneralize}, computes the reach set \( \text{Reach}(A^N) \)—a subset of the state space \( Q^N \)—then projects this onto a smaller state space \( Q^P \), and then lifts this back to \( Q^N \). Although our description above is in terms of the syntactic objects and logical formulas, these operations can be described in terms of mappings between the subsets of the state spaces. We reason about monotonicity of this procedure in terms of sets of states.

**Proposition 1.** Let \( f : \text{Pow}(Q^N) \to \text{Pow}(Q^N) \) be a mapping corresponding to the operations of \textit{projectAndGeneralize}. Under the set inclusion partial order, \( f \) is monotonic.

**Inductive Invariant Synthesis Fixed-Point Procedure.** In this section, we present an algorithm that uses \textit{projectAndGeneralize} for synthesizing inductive invariants (see Figure 4). First, Passel computes the composition \( A^N \) of a fixed number \( N \) of automaton \( A(N, i) \), by first instantiating each \( A_1, \ldots, A_N \) and then taking their composition. This composed automaton is an input argument for \textit{projectAndGeneralize} (line 2). The synthesis procedure of Passel repeatedly calls \textit{projectAndGeneralize} to generate assertions \( \psi(i_1, \ldots, i_P) \). We fix \( N \), \( P \), and \( A^N \), and denote the mathematical function computed by \textit{projectAndGeneralize}(\( A^N \), \( \theta(N, P) \), \( N \), \( P \)) by \( f(\theta(N, P)) \) and suppress the other arguments. The first iteration of \( f \) uses the set of initial states \( \Theta^N \) as the initial state argument of \textit{projectAndGeneralize}, then subsequent iterations use the set of states corresponding to the generated assertion \( \theta(N, P) \) from line 9 as a new set of initial states. Since \( N \) is a fixed integer, the formula \( \theta(N, P) \triangleq \forall i_1, \ldots, i_P \in [N]. \psi(i_1, \ldots, i_P) \) from line 9 is equivalent to a conjunction. Once synthesis reaches a fixed-point (shown below), the final assertion is used as a candidate inductive invariant for the network composed of an arbitrary number \( N \) of automata \( A(N, i) \).

**Proposition 2.** The synthesis function terminates and its output \( \psi \) is a fixpoint of the function \( f \).

**Proposition 3.** The least fixed point \( f^* \) is an assertion of the form \( \theta(N, P) \triangleq \forall i_1, \ldots, i_P \in [N]. \psi(i_1, \ldots, i_P) \) and is an inductive invariant for the \( N \) automata network \( A^N \).
The next result allows us to conclude whether $A^N$ has any inductive invariants of the shape generated using $P$ quantified indices that is sufficient for proving $\zeta(N)$, by checking if the least fixed point $f^*$ is an inductive invariant and implies $\zeta(N)$.

**Proposition 4.** Given $N, P \in \mathbb{N}$, there exists an inductive invariant $\theta(N, P) \Rightarrow \zeta(N)$ if and only if the least fixed point $f^*$ is an inductive invariant and implies $\zeta(N)$.

Up to this point, everything has been for a fixed sized hybrid network $A^N$. We now combine these results with the small model theorem (Theorem 1) to state that synthesis generates an inductive invariant for an arbitrary number $N$ of automata in the network. The next proposition allows us to conclude whether or not the hybrid automaton network $A^N$, with any number of automata $N$ in the network $A^N$, has any inductive invariant of the form $\forall i_1, \ldots, i_P \in [N].\psi(i_1, \ldots, i_P)$ that is sufficient to prove $\zeta(N)$.

**Proposition 5.** Suppose for any $N \in \mathbb{N}$, the conditions for inductiveness of $A^N$ for properties of the form $\forall i_1, \ldots, i_P \in [N].\psi(i_1, \ldots, i_P)$ are assertions with a small model property with bound $N_0$. Let $f^* = \theta(N, P) = \forall i_1, \ldots, i_P \in [N].\psi(i_1, \ldots, i_P)$ be the fix point for every $N$ satisfying $1 \leq N \leq N_0$. Then, $\forall N \in \mathbb{N} \forall i_1, \ldots, i_P \in [N].\psi(i_1, \ldots, i_P)$ is an inductive invariant for $A^N$.

4 Passel and Experimental Evaluation

Passel implements an automatic method for checking the conditions for inductive invariance (Definition 1) for parameterized networks of hybrid automata, as shown in the function `inductiveInvariance` in Figure 2. The inductive invariance conditions (Definition 1) are encoded using formulas appearing essentially as they do in the definitions of the discrete and continuous transition relations in Subsection 2.2.

Passel implements the invariant synthesis procedures presented in Subsection 3.2. Passel uses the SMT solver Z3 [12] for proving validity, checking satisfiability, and performing quantifier elimination. Passel proves validity by checking that the negation of assertions is unsatisfiable, and performs this check relying on the quantifier handling procedures within Z3, particularly model-based quantifier instantiation (MBQI) and quantifier elimination. The synthesis methods of Figures 3 and 4 rely on projection and syntactic manipulations, which are implemented respectively using quantifier elimination procedures in Z3 and syntactic operations (expression replacement and quantifier introduction). In practice, it is useful to project away all variables—Figure 3, line 6—except the discrete ones (variables with types $L$ and $[N]_\bot$), only the control location variables and real variables, and combinations of these with and without projecting any global variables away, so Passel does this. Passel models the local variables of $A(i)$ as unary functions, mapping indices to the variables type—for each local variable $x \in V_i$, $x : [N] \rightarrow \text{type}(x)$, where $N$ is not fixed a priori, but has some assumption specified, such as $N \geq 2$ or $N \geq 2 \wedge N \leq 73$. Global variables are modeled as constants of their types.
Table 1. Experimental results. All time units are seconds and memory units are megabytes. Checks (✓) in columns ∀i.ψ(i) and ∀i,j.ψ(i,j) mean that the single and doubly quantified synthesized invariants θ(N) are inductive, and X means not. Checks (✓) in column ζ(N) means Passel succeeded in generating a quantified strengthening that implied the desired safety property for all N ∈ N (and X, not). Column synth time reports the runtime to synthesize candidate invariants ∀i.ψ(i) and ∀i,j.ψ(i,j), and column inv time reports the runtime to prove the candidate assertions are inductive invariants for any N, and that ∀i.ψ(i) ∧ ∀i,j.ψ(i,j) ⇒ ζ(N).

<table>
<thead>
<tr>
<th>Name</th>
<th>PHAVer time</th>
<th>mem</th>
<th>∀i.ψ(i)?</th>
<th>∀i,j.ψ(i,j)?</th>
<th>ζ(N)?</th>
<th>synth time</th>
<th>inv time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFischer</td>
<td>1.81</td>
<td>9.94</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>9.69</td>
<td>138.98</td>
</tr>
<tr>
<td>RFischer (Bug)</td>
<td>9.88</td>
<td>12.62</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>34.30</td>
<td>6578.32</td>
</tr>
<tr>
<td>TFischer</td>
<td>1.31</td>
<td>9.43</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>8.52</td>
<td>64.95</td>
</tr>
<tr>
<td>TFischer (Bug)</td>
<td>2.12</td>
<td>12.52</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>19.59</td>
<td>4169.07</td>
</tr>
<tr>
<td>Simple SATS</td>
<td>1.72</td>
<td>10.61</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>4.54</td>
<td>14.70</td>
</tr>
<tr>
<td>Sided SATS</td>
<td>7.47</td>
<td>22.98</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>13.95</td>
<td>81.10</td>
</tr>
<tr>
<td>TMux</td>
<td>0.71</td>
<td>6.74</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3.19</td>
<td>20.01</td>
</tr>
<tr>
<td>MUX-SEM</td>
<td>0.2</td>
<td>5.61</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.68</td>
<td>1.39</td>
</tr>
</tbody>
</table>

Experimental Setup. All experiments were conducted on a modern laptop with a 2.2GHz quad-core Intel Core i7-2670QM processor and 16GB RAM, running 64-bit Windows 8. Passel is written in C#. Passel used the Z3 SMT solver version 4.1 [12] through the C# API. Passel takes a syntactic specification for a single hybrid automaton A(i) in a variant of HyXML [20], as described in detail in Subsection 2.1. The specification of one A(i) is an additional contribution of Passel. While UPPAAL [9] allows for specifying automata in a parameterized manner, all the hybrid systems model checkers require the user to compute each A₁, ..., A₇, and the model checker may then compose these.

Passel has several modes of operation: (a) checking if a list of given assertions are inductive invariants (for arbitrary N or fixed N), (b) bounded model checking, (c) generating an N-instance input file for the hybrid systems reachability tool PHAVer [13], or (d) attempting to automatically find an inductive invariant proof of a safety property ζ using the synthesis method (Subsection 3.2). In the experiments, Passel used PHAVer (version 0.38) for the reachability computation Reach(A⁰) of a finite network A⁰ of N automata. PHAVer was run under an Ubuntu virtual machine using VMWare Player on the same laptop, with access to two cores and 4GB RAM. Passel called PHAVer via the VMWare VIX API, although we can also execute Passel on Linux using Mono. Input files for PHAVer were generated by Passel from the data structure inside Passel encoding the syntactic structure of A(i). We took great care in the development and integration of Passel and PHAVer to ensure consistency of semantics. We measured time and memory usage of PHAVer with memtime.

Experimental Results. We evaluated the invariant synthesis method on several timed and hybrid examples, with our results presented in Table 1⁶. We also

⁶ Passel, the specification files for the examples evaluated, and output logs showing synthesized inductive invariants are available for download from http://www.taylortjohnson.com/research/spin2013/.
evaluated a variety of correct and buggy versions of sanity-check protocols (like the purely discrete MUX-SEM examples from [6], Fischer's mutual exclusion protocol, and other protocols we have previously verified). We used a choice of \( N = 3 \) for the reach set computations \( \text{Reach}(A^N) \), but did not see any benefit (in terms of being able to prove inductive properties for more examples) to using greater values in our experiments. We reiterate that when Passel proves the inductive invariance conditions, the only assumption on \( N \) is \( N \geq 2 \), unless we have a small model theorem with bound \( N_0 \), and then Passel uses \( 1 \leq N \leq N_0 \).

We analyzed correct and buggy versions of Fischer’s mutual exclusion algorithm Benchmarks 1, 2, 3, and 4, as presented earlier in Figure 1. Benchmarks 1 and 2 had rectangular dynamics, so \( lb < ub \) and \( \dot{x}[i] \in [lb,ub] \), while Benchmarks 3 and 4 had timed dynamics, so \( lb = ub \) and \( \dot{x}[i] = lb = ub \), which results in fewer candidate invariants being synthesized (since there are fewer possible states in the reach set). The timed buggy version had \( A \geq B \), while the correct version requires \( A < B \). We used concrete values \( A = 5 \) and \( B = 7 \) for the correct timed version, and \( A = 5 \) and \( B = 4 \) for the buggy version, and \( lb = ub = 1 \) for the dynamics of both timed cases. While the invariant synthesis procedure requires concrete values (since PHAVer uses numerical libraries for computing the reach set), Passel can use symbolic values (e.g., just an assertion \( A < B \) is required for the correct version). The rectangular buggy version had \( A = 5 \) and \( B = 6 \), while the correct version used \( A = 5 \) and \( B = 50 \), and \( lb = 3 \) and \( ub = 7 \) for the dynamics of both cases. With timed dynamics, Fischer maintains mutual exclusion for \( A < B \), but with rectangular dynamics, mutual exclusion requires \( B > A \cdot ub \). Passel automatically proved mutual exclusion of the correct timed and rectangular versions for any \( N \in \mathbb{N} \) as shown in Benchmarks 1 and 3. The key invariant synthesized for Fischer is related to timing, and for the timed version is:

\[
\forall i,j \in [N], (q[i] = \text{wait} \land q[j] = \text{try} \land g = i) \Rightarrow (B - A) > (x[i] - x[j]).
\]

The buggy versions (Benchmarks 2 and 4) were included as sanity checks, and Passel could not prove mutual exclusion of these since it does not hold, although Passel did synthesize many inductive invariants. The runtimes of buggy versions are large, because Passel attempted to prove many candidate invariants. For example in the timed buggy version, 460 candidate invariants were synthesized and Passel proved 98 of them, although these failed to imply mutual exclusion since it is not an invariant.

Passel also proved safety of (Benchmarks 5 to 6), which model two simplified versions of the Small Aircraft Transportation System (SATS) [1], where the safety property \( \zeta \) is that no two aircraft collide. We previously performed a manual deductive strengthening proof for this protocol in [18], and automatically checked it using Passel, and we also analyzed the protocol using a different technique in [17]. Collision avoidance is, for any two aircraft approaching a runway, there is at least a positive real distance \( L_S \) between their positions along a
one-dimensional line (the path to the runway):

$$\forall i, j \in [N]. (q[i] \in \{BR, BL, F\} \land q[j] \in \{BR, BL, F\} \land x[i] > x[j]) \Rightarrow x[i] - x[j] \geq L_S.$$  

The locations $BR$, $BL$, etc. specify the aircraft are attempting to land, and $x[i] > x[j]$ ensures aircraft $i$ is ahead of $j$. The aircraft travel along the line with velocities $\dot{x}[i] \in [v_{min}, v_{max}]$, and the key invariant synthesized used to establish collision avoidance is

$$\forall i, j \in [N]. (q[i] \in \{BR, BL, F\} \land q[j] \in \{BR, BL, F\}) \Rightarrow x[i] \geq \frac{(L_B + L_F - x[j])}{(v_{min}(v_{max} - v_{min}))},$$

where $L_B$ and $L_F$ are the lengths of different paths on the way to the runway. Synthesizing this invariant, and others, allowed Passel to prove the collision avoidance property fully automatically.

5 Related Work

Parameterized systems and verification have been extensively studied in the context of discrete automata, typically modeling processes or threads. Parametrization of real-valued parameters (such as the $A$ and $B$ in Fischer) has been studied for hybrid automata [15], but not the parametrization on the number of automata we consider. The invisible invariants method [26,6,7,23,21] is a heuristic approach for synthesizing inductive invariants for parameterized networks of discrete automata. It is used as a subroutine of a fixed-point procedure in [23], which is a complete method for synthesizing inductive invariants for networks of discrete automata. When viewed in abstract interpretation [21], these methods are subroutines of a more general fixed-point procedure. Our paper presents the first use of invisible invariants for analyzing timed or hybrid networks.

Uniform verification of parameterized timed systems has been studied in [4,3,2]. An abstraction approach for verification of timed systems using networks invariants is developed in [14]. Bounded reachability methods are developed for timed parameterized systems in [11,10], and we applied these reachability techniques to a simplified model of an air traffic control protocol in [17]. Quantified differential dynamic logic can be used to model systems similar, and often more general, than those that we consider, although at the expensive of sometimes requiring manual intervention [24]. In addition, the quantified differential invariants have the same shape as the class of assertions we are attempting to automatically generate for hybrid automata networks [25]. We studied parameterized verification for hybrid systems in [18], where we developed a small model theorem enabling verification of safety properties for arbitrary numbers of rectangular hybrid automata by verifying all instances up to some finite bound.
6 Conclusion and Future Work

We presented the Passel verification tool, which has been successful at automatically proving safety properties of parameterized networks of hybrid automata with rectangular dynamics. To accomplish this, Passel uses an extension of the invisible invariants method in a fixed-point procedure for automatically synthesizing inductive invariants. With this synthesis method and algorithmic checking of the inductive invariance conditions, Passel has performed the first fully automatic verification of several parameterized networks of hybrid automata, such as Fischer’s mutual exclusion with rectangular dynamics, part of a conceptual air-traffic control protocol, and several others. There are many future directions, for instance, we plan to extend Passel and its theoretical basis to systems with more general continuous dynamics, such as linear or nonlinear differential equations.

References


A Appendix

Fig. 5. Graphical depiction of \( A(\mathcal{N}, i) \) from Figure 1 specifying Fischer’s mutual exclusion algorithm.

B Synthesis Explanation

Suppose we want to synthesize inductive invariants with \( P = 1 \) universally quantified index variables of the form \( \psi(\mathcal{N}) = \forall i_1 \in [\mathcal{N}], \phi(i_1) \), where \( \phi(i_1) \) is a quantifier-free formula over the variables \( V_{i_1} \). We accomplish this by computing reachability of finite instantiations \( \mathcal{N} \), and transforming this set into an inductive invariant for any \( \mathcal{N} \in \mathbb{N} \). First, the reach set \( \text{Reach}(A^N) \) of a network with a finite fixed number \( N > P \in \mathbb{N} \) of automata is computed (line 3)\(^7\). For the example, suppose \( N = 2 \) so we have a formula \( \phi(1, 2) \) corresponding to the reach

\( ^7 \) For hybrid automata with dynamics specified by general (e.g., nonlinear or even linear) ordinary differential equations, this may be undecidable or yield an overapproximation of the reach set.
Fig. 6. Passel input file specifying $A(N, i)$ for simple mutual exclusion algorithm.

set:

$\text{Reach}(A^N) \triangleq (q_1 = \text{rem} \land q_2 = \text{rem} \land g = \bot \land x_1 \geq 0 \land x_2 \geq 0) \lor$

$(q_1 = \text{try} \land q_2 = \text{rem} \land g = 1 \land 4x_2 \geq x_1 + 40 \land x_1 \geq 0) \lor$ (1)

$(q_1 = \text{rem} \land q_2 = \text{try} \land g = 2 \land x_2 \geq 0 \land 4x_1 \geq x_2 + 40) \lor$ (2)

$(q_1 = \text{rem} \land q_2 = \text{cs} \land g = 2 \land x_2 \geq 0 \land 6x_1 \geq x_2 + 75) \lor$

$(q_1 = \text{cs} \land q_2 = \text{rem} \land g = 1 \land 6x_2 \geq x_1 + 75 \land x_1 \geq 0)$. 

Note that $\text{Reach}(A^N)$ is in disjunctive normal form (DNF).

Next, for each conjunct $r$ in $\text{Reach}(A^N)$, we project away the variables of any automata with indices greater than $P = 1$, since we want to synthesize an inductive invariant with $P$ (one) universally quantified index variables (line 6)\(^8\). This projection can be computed using quantifier elimination procedures over the types of the variables $V_i$, and is how we accomplish this in Passel. For example, suppose $r$ is from Equation 1, that is,

$r \triangleq (q_1 = \text{try} \land q_2 = \text{rem} \land g = 1 \land 4x_2 \geq x_1 + 40 \land x_1 \geq 0)$.

\(^8\) Since existential quantification distributes over disjunction, we consider each conjunct at a time. This is one difference in our method from the original invisible invariants methods, and since that approach was implemented using BDDs, all of these operations were done on the whole reach set, whereas our representation of state is through formulas over booleans, (bounded) integers, and reals.
then projecting away the variables of automaton 2 at line 6, we have:

\[
QF[r] = \exists V_2. r \\
= \exists q_2 \in \mathbb{L}, \exists x_2 \in \mathbb{R}. r \\
= (q_1 = \text{try} \land g = 1 \land x_1 \geq 0).
\]

Based on the value of \( P \), the projection on line 6 is

\[
QF[r] \triangleq \exists V_N \setminus \left( \bigcup_{i \in [P]} V_i \right). r,
\]

which projects away the variables of all automata with indices higher than \( P \). In general, we project away some subset of the variables \( V_N \), for example, onto all the real or discrete variables. We have found in practice that it is useful to project away all but variables except the discrete ones (variables with types \( L \) and \( \perp \)), only the control location variables and real variables, and combinations of these with and without projecting any global variables away.

Next, we syntactically manipulate \( QF[r] \) in order to determine a quantified assertion for any choice of the number of automata \( \mathcal{N} \). We define the generalization by syntactically replacing all valuations of index variables equal to a value in \( [P] \) with fresh index variables (symbols) \( i_1, i_2, \ldots, i_P \) (line 8). For the \texttt{TMux} example, we replace 1 with \( i_1 \) yielding a formula

\[
QF[r] \triangleq (q_{i_1} = \text{try} \land g = i_1 \land x_{i_1} \geq 0).
\]

For this \( QF[r] \in \text{Reach}(\mathcal{A}^N) \), we are finished. However, \( QF[r] \) may still contain index 2 for valuations of the index-valued global variable \( g \) after replacing 1 with \( i_1 \) for instance in Equation 2:

\[
QF[r] \triangleq (q_{i_1} = \text{rem} \land q_2 = \text{try} \land g = 2 \land x_2 \geq 0 \land 4x_1 \geq x_2 + 40),
\]

Thus, we must take additional care in generalizing any index-valued variables (line 12). Since some of the valuations of index-valued variables in \( \text{Reach}(\mathcal{A}^N) \) will exceed \( P \) (since \( P < N \)), we must transform these valuations to symbols equal (or not equal) to \( i_1, \ldots, i_P \). The process described by line 12 is: for any index-valued variable \( v \), if the valuation \( v = k \) where \( k \leq P \), then set \( v = i_k \), and otherwise for \( k > P \) or \( k = \perp \), set \( v \neq i_1 \land \ldots \land v \neq i_P \). For the \texttt{TMux} example for \( r \) from Equation 2, we have

\[
QF[r] \triangleq (q_{i_1} = \text{rem} \land g \neq i_1 \land x_{i_1} \geq 10.0),
\]

which is now ensured not to contain any indices or index-valued variable valuations other than the symbols \( i_1, \ldots, i_P \).
Finally, we take the disjunction of the \( QF[r] \)'s for all \( r \in R \) (line 18). For the TMux example, this yields the assertion

\[
\phi(i_1) \overset{\text{def}}{=} (q_{i_1} = \text{rem} \land g \neq i_1 \land x_{i_1} \geq 0) \lor \\
(q_{i_1} = \text{rem} \land g \neq i_1 \land x_{i_1} \geq 10.0) \lor \\
(q_{i_1} = \text{rem} \land g \neq i_1 \land x_{i_1} \geq 12.5) \lor \\
(q_{i_1} = \text{try} \land g = i_1 \land x_{i_1} \geq 0) \lor \\
(q_{i_1} = \text{cs} \land g = i_1 \land x_{i_1} \geq 0).
\]

Finally, we create a quantified formula \( \theta(N, P) \) (Figure 4, line 9) as:

\[
\theta(N, P) = \bigvee_{i_1, i_2, \ldots, i_P \in [N]} \psi(i_1, i_2, \ldots, i_P),
\]

where \( \bigvee \) indicates that all the quantified index variables are distinct (i.e., \( i_1 \neq i_2 \ldots \neq i_P \)). At this point, since \( N \) is a fixed, finite number (e.g., 3), we could convert \( \theta(N, P) \) to a conjunction, but for an arbitrary \( N \), we would need the quantifiers. To summarize, \textit{projectAndGeneralize} computes the reach set—a subset of the state space \( Q^N \)—then projects this onto a smaller state space \( Q^P \), and then lift this back to \( Q^N \).

C Proofs and Additional Lemmas

\textit{Proof (of Proposition 1).} Since \( N \) is fixed for Figure 3, \textit{projectAndGeneralize} defines a mapping \( f : \text{Pow}(Q^N) \rightarrow \text{Pow}(Q^P) \), and we show for any \( x, y \in \text{Pow}(Q^N) \), if \( x \subseteq y \) then \( f(x) \subseteq f(y) \). We prove this for each operation of \textit{projectAndGeneralize}.

First of all, \( \text{Reach}(A^N, x) \subseteq \text{Reach}(A^N, y) \). Next, we consider some clause \( r \in R \) where \( R \) is the DNF representation of some \( \text{Reach}(A^N, \ldots) \). That is, \([r] \subseteq \text{Pow}(Q^N) \). The projection is a mapping \( \rho : \text{Pow}(Q^N) \rightarrow \text{Pow}(Q^P) \), where \( P < N \), is also monotonic. That is for any \( r' \Rightarrow r \), \([r'] \subseteq [r] \in \text{Pow}(Q^N) \) and \([\rho(r')] \subseteq [\rho(r)] \in \text{Pow}(Q^P) \). Next, the abstraction of index-valued variables in line 12 is a mapping \( \alpha : \text{Pow}(Q^P) \rightarrow \text{Pow}(Q^P) \). Since \( \alpha \) substitutes explicit values of index-valued variables, for instance, \( v = n \) for some \( n > P \) with symbolic \( v \neq i_1 \land \ldots \land v \neq i_P \), it maps to an equal or larger set of states by considering every possible valuation of \( i_1 \) through \( i_P \), so \( \alpha \) is monotonic. Since each transformation in the loop line 4 is monotonic for each clause \( r \in R \), the transformation of \( R \) is also monotonic. Finally, consider the generalization in line 9 of Figure 4, which is a mapping \( \gamma : \text{Pow}(Q^P) \rightarrow \text{Pow}(Q^P) \). For any \( x, y \in \text{Pow}(Q^P) \), if \( x \subseteq y \), then \( \gamma(x) \subseteq \gamma(y) \) is satisfied since \( \gamma \) is the identity mapping (modulo renaming). Since all operations are monotone under the set inclusion partial order and the compositions of monotonic functions are monotonic, the composition, \( f : \text{Pow}(Q^N) \rightarrow \text{Pow}(Q^N) = \gamma \circ \alpha \circ \rho \) is monotonic.

\textit{Proof (of Proposition 4).} For the only if proof, pick \( \theta(N, P) = f^*(\text{Init}) \), which implies \( \theta(N, P) \Rightarrow \zeta(N) \). For the if proof, there is some \( \theta(N, P) \) such that \( \theta(N, P) \Rightarrow \zeta(N) \). Since \( f^* \) is the least fixed point, we have \( f^* \Rightarrow \theta(N, P) \), so we also have \( f^* \Rightarrow \zeta(N) \).
Proof (of Proposition 2). It is straightforward to see that projectAndGeneralize terminates, and by Proposition 1, all the operations of the fixed-point loop lines 6 through 10 can be described by a mapping \( f : \text{Pow}(Q^N) \rightarrow \text{Pow}(Q^N) \), which is monotonic under set inclusion \( \subseteq \). Since the partial order \( L = \langle \text{Pow}(Q^N), \subseteq \rangle \) is complete and \( f \) is monotonic, by Kleene’s fixpoint theorem, the loop terminates and the computed \( \psi \) is a fixpoint of \( f \).

Proof (of Proposition 5). We show that the \( \psi(i_1, \ldots, i_p) \) computed as the least fixed-point \( f^* \) is an inductive invariant \( \forall \mathcal{N} \geq N_0 \). Suppose not, so \( \exists N_0 \) such that \( \forall i_1, \ldots, i_p \in [N_0], \psi(i_1, \ldots, i_p) \) violates the conditions for inductive invariance. Rewriting, this means that \( \exists N_0 \in \mathbb{N}, \forall i_1, \ldots, i_p \in [N_0], \neg \psi(i_1, \ldots, i_p) \) is satisfied. Since \( \mathcal{N} \geq N_0 \) and since \( f^* = \theta(N,P) = \forall i_1, \ldots, i_p. \psi(i_1, \ldots, i_p) \) is an inductive invariant for every finite instance of \( N \) automata for \( 1 \leq N \leq N_0 \), by applying Proposition 3 and Theorem 1, we reach a contradiction, since \( \theta(N,P) \) must be an inductive invariant for each \( N \) by Theorem 1. Thus, \( \forall \mathcal{N} \in \mathbb{N}, \forall i_1, \ldots, i_p. \psi(i_1, \ldots, i_p) \) is an inductive invariant of \( \mathcal{A}^{\mathcal{N}} \).

Semantics of Continuous Trajectories We model rectangular dynamics using an additional existential quantifier over reals over the time transition. The discussion that follows is how we are able to convert the relation flow used to define the set of continuous trajectories with a function \( \text{flow}_f \) defined below. An alternative would be to track upper and lower bounds of rectangular variables using two clocks, and convert to a timed automata as done in [16]. To define \( T_N \) we first define the function \( \text{flow}(\nu[i], m, t) \) which returns a valuation \( \nu'[i] \), such that for each \( v \in V_i \), if \( v \)'s type is not real, then \( \nu'[i].v = \nu[i].v \), but otherwise, \( \nu'[i].v = \nu[i].v + \text{flowrate}(m, v)t \).

Proposition 6. Consider the flow function defined by \( \text{flow}_f(\nu[i], m, t) \), which returns a valuation \( \nu'[i] \), such that for each \( v \in V_i \), if \( v \)'s type is not real or its update type is not continuous, then \( \nu'[i].v = \nu[i].v \), but otherwise, \( \nu'[i].v = \nu[i].v + \text{flowrate}(m, v)t \), where \( \text{flowrate}(m, v)t = \delta t \), for any \( \delta \in [a, b] \). Alternatively, consider the flow relation defined by \( \text{flow}_r(\nu[i], m, t) \), which returns a set of valuations \( \nu[i] \), where for each \( \nu'[i] \in \nu[i] \), such that for each \( v \in V_i \), if \( v \)'s type is not real or its update type is not continuous, then \( \nu'[i].v = \nu[i].v \); but otherwise, \( \nu[i].v + at \leq \nu'[i].v \leq \nu[i].v + bt \).

Recall a pair \((\nu, \nu') \in T_N \) iff:

\[
\exists t_1 \in \mathbb{R}_{\geq 0} : \forall i : [N] : \exists m \in \text{Mode}_i : \\
\land \forall t_2 \leq t_1 : \text{flow}(\nu[i], m, t_2) \models \text{inv}(m, i) \\
\land \forall t_2 \leq t_1 : \text{flow}(\nu[i], m, t_2) \models \text{stop}(m, i) \implies t_2 = t_1 \\
\land \nu'[i] \in \text{flow}_r(\nu[i], m, t_1).
\]
Consider the alternative definition of $T^f_N$, where a pair $(v, v') \in T^f_N$ iff:

\[
\exists t_1 \in \mathbb{R}_{\geq 0} : \forall i : [N] : \exists \delta \in \mathbb{R}_{\geq 0} \exists m \in \text{Mode}_i : \\
\land \forall t_2 \leq t_1 : \text{flow}(v[i], m, t_2) = \text{inv}(m, i) \\
\land \forall t_2 \leq t_1 : \text{flow}(v[i], m, t_2) = \text{stop}(m, i) \implies t_2 = t_1 \\
\land v'[i] = \text{flow}_f(v[i], m, t_1).
\]

Then, the sets of trajectories under these definitions are the same, $T_N = T^f_N$.

Proof. We show $T_N \subseteq T^f_N$ and $T^f_N \subseteq T_N$. It is clear that $T^f_N \subseteq T_N$. For $T_N \subseteq T^f_N$, take any trajectory $\tau \in T_N$. The valuation of any variable $v$ at state $v'$ along $\tau$ satisfies $v[i].v + at \leq v'[i].v \leq v[i].v + b[t]$. Consider a trajectory under the other semantics, where the first state along this trajectory $x$ satisfies $x[i].v = v[i].v$ for each $i \in [N]$ and each variable $v$. Suppose $\delta = \frac{1}{t} \int_0^t v(t)dt$, where $v(t)$ is the actual choice of $\text{flowrate}(m, v[i].v)$ over the length of the trajectory. This integral must exist, and thus we have picked $\delta$ as the average flow rate over the trajectory of length $t$. Since $\text{flowrate}(m, v[i].v) \in [a, b]$ for $a = \text{lflowrate}$ and $b = \text{uflowrate}$, which is a convex set, and $\delta \in [a, b]$ is also a convex set, we have that for this choice of $\delta$, $x[i].v + \delta t \in \tau$. Thus, $\tau \in T^f_N$. 