Anonymized Reachability of Hybrid Automata Networks

Taylor T. Johnson[1] and Sayan Mitra[2]

[1] University of Texas at Arlington, Arlington, TX 76019, USA
taylor.johnson@uta.edu
[2] University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
mitras@illinois.edu

Abstract. In this paper, we present a method for computing the set of reachable states for networks consisting of the parallel composition of a finite number of the same hybrid automaton template with rectangular dynamics. The method utilizes a symmetric representation of the set of reachable states (modulo the automata indices) that we call anonymized states, which makes it scalable. Rather than explicitly enumerating each automaton index in formulas representing sets of states, the anonymized representation encodes only: (a) the classes of automata, which are the states of automata represented with formulas over symbolic indices, and (b) the number of automata in each of the classes. We present an algorithm for overapproximating the reachable states by computing state transitions in this anonymized representation. Unlike symmetry reduction techniques used in finite state models, the timed transition of a network composed of hybrid automata causes the continuous variables of all the automata to evolve simultaneously. The anonymized representation is amenable to both reducing the discrete and continuous complexity. We evaluate a prototype implementation of the representation and reachability algorithm in our satisfiability modulo theories (SMT)-based tool, Passel. Our experimental results are promising, and generally allow for scaling to networks composed of tens of automata, and in some instances, hundreds (or more) of automata.

Keywords: hybrid automata network, reachability, verification, symmetry

1 Introduction

Networks consisting of automata that communicate via shared variables are useful for modeling distributed algorithms such as mutual exclusion algorithms, media access control (MAC) such as time-division multiple access (TDMA) protocols, and distributed cyber-physical systems (CPS) such as air-traffic control systems [1]. However, as the state-space of the network consisting of parallel compositions of these automata grows exponentially in the number of automata N, automated analysis is challenging. It is particularly challenging for timed and hybrid systems, where the number of continuous variables (dimensions) also grows. Such networks are often specified in a symmetric manner—such as being composed of instantiations of an automaton template $\mathcal{A}(\mathsf{N}, i)$—and are often amenable to methods that exploit symmetries. Formal analysis and state-space construction methods that exploit symmetries have been thoroughly

investigated for many classes of system models, because such methods ameliorate the state-space explosion problem [2–14].

For example, several methods exploiting symmetry have been developed and implemented for the Murφ verification system [15] for discrete systems, such as the *scalarset* data structure [3], and the *repetitive id* data structure [5]. Advances in tools like UPAAAL [16] and PAT [17] that exploit state-space symmetries have enabled scaling to larger models. For instance, the scalarset data structure from Murφ was extended for timed systems and implemented in UPAAAL [8, 18], and a clock-symmetry reduction method has been implemented in the PAT model checker [14]. Quasi-equal clocks and variables for timed [19] and hybrid (multi-rate) [20] automata networks also allow reductions in state-space explosion, but do not require automata in the network to be identical (modulo identifiers), as we do. We focus on safety properties, and to the best of our knowledge, before this paper, such symmetry techniques have not yet been applied to systems with general continuous dynamics like the rectangular differential inclusions we consider (e.g., [20] analyzes multi-rate automata and does not allow differential inclusions). The method described in this paper and implemented in our *Passel* verification tool [21–23] uses the SMT solver Z3 [24]. The method is used as a subroutine in methods for performing uniform verification of parameterized networks of hybrid automata (e.g., verification for all network sizes, $\forall \mathsf{N} \in \mathbb{N}$, $\mathcal{A}(\mathsf{N},1) \| \ldots \| \mathcal{A}(\mathsf{N},\mathsf{N}) \models \zeta(\mathsf{N})$), although we highlight that this paper addresses fixed, constant choices of N only.

2 Hybrid Automata Network Syntax and Semantics

We specify the behavior of each participant in the network using a syntactic structure called a hybrid automaton template, denoted by $\mathcal{A}(\mathsf{N}, i)$.[3] The special symbols N and i are natural numbers that respectively refer to the number of automata, and the i^{th} automaton. For a natural number n, the set $[n]$ is $\{1, \ldots, n\}$. For a set S, the set $S_\perp$ is $S \cup \{\perp\}$. Fixing a particular value of N gives concrete instances of $[\mathsf{N}]$ and $[\mathsf{N}]_\perp$.

Terms and Formulas. We use a class of formulas to: (a) specify the syntactic components of a hybrid automaton template $\mathcal{A}(\mathsf{N}, i)$, and (b) represent sets of states symbolically in the reachability computation. Formulas are built-up from constants, variables, and terms of several types. The grammar for formulas is:

$$
\begin{aligned}
\mathsf{ITerm} &::= \perp \mid 1 \mid \mathsf{N} \mid i \mid p[i] \\
\mathsf{DTerm} &::= l_c \mid q \mid q[\mathsf{ITerm}] \\
\mathsf{RTerm} &::= 0 \mid 1 \mid r_c \mid x \mid x[\mathsf{ITerm}] \\
\mathsf{RPoly} &::= \mathsf{RTerm} \mid \mathsf{RPoly}_1 \ + \ \mathsf{RPoly}_2 \mid \mathsf{RPoly}_1 \ - \ \mathsf{RPoly}_2 \mid (\mathsf{RPoly}_1 \ * \ \mathsf{RPoly}_2) \\
\mathsf{Atom} &::= \mathsf{ITerm}_1 = \mathsf{ITerm}_2 \mid \mathsf{DTerm}_1 = \mathsf{DTerm}_2 \mid \mathsf{RPoly} \ < \ 0 \\
\mathsf{Formula} &::= \mathsf{Atom} \mid \neg \mathsf{Formula} \mid \mathsf{Formula}_1 \wedge \mathsf{Formula}_2 \mid \exists x \ \mathsf{Formula}
\end{aligned}
$$

The grammar is composed of *index terms* (ITerm) with type $[\mathsf{N}]_\perp$, *discrete terms* (DTerm) with type L, and *real terms* (RTerm) with type $\mathbb{R}$. For a discrete term, l_c is constant from L and q is a discrete variable. For a real term, r_c is a real numerical constant and x is a

[3] Readers interested in additional technical details are referred to [23, Chapters 2 and 4].

real variable. Index ($p[i]$), discrete ($q[\mathsf{ITerm}]$), and real ($x[\mathsf{ITerm}]$) *pointer variables* are names for arrays composed of N elements of the corresponding type, respectively referenced at an index variable i, or an evaluation of an index term ITerm. Atoms (Atom) are composed of ordered relations between real polynomials (RPoly), as well as equality relations between index terms and discrete terms. Formulas are composed of Boolean combinations of atoms and shorter formulas. Comparison operators are expressed using negation ($\neg$) and conjunction ($\wedge$) in formulas. Combining the Boolean operators $\wedge$ and $\neg$ with the $<$ operator, other comparison operators like $=$, $\neq$, $\leq$, $>$, and $\geq$, can be expressed. We assume the language contains the standard quantifiers and Boolean operators, even if not explicitly specified in the grammar (e.g., universal quantification $\forall$, implication $\Rightarrow$, disjunction $\vee$, less-than-or-equal $\leq$, etc.).

Variables. A hybrid automaton $\mathcal{A}(\mathsf{N}, i)$ has a set of *variables*, each of which is a name used for referring to state and is a term in the grammar just defined. As specified in the grammar, each variable v is associated with a *type*—denoted $type(v)$—that defines a set of values the variable may take. The type of a variable may be: (a) L: a finite set of locations names, (b) $[\mathsf{N}]_\perp$: a set of automaton indices—called pointers—with the special element $\perp$ that is not equal to any automaton's index, or (c) $\mathbb{R}$: the set of real numbers. A variable may be a *local* variable with a name of the form $variable\_name[i]$, or *global*, in which case its name does not have a symbolic index $[i]$. For example, $q[i] : \mathsf{L}$, $p[i] : [\mathsf{N}]_\perp$, and $x[i] : \mathbb{R}$ respectively define location, pointer, and real typed local variables, while $g : [\mathsf{N}]_\perp$ is a global variable of pointer type. The sets of local and global variables are denoted by $\mathsf{V}_L(i)$ and $\mathsf{V}_G(i)$, respectively. The *valuation* of a variable v is a function that associates the variable name v to a value in its type $type(v)$. For a set of variables V, $val(\mathsf{V})$ is the set of valuations of each $v \in \mathsf{V}$. For a set of variables V, $\mathsf{V}' \triangleq \{v' | v \in \mathsf{V}\}$ and $\dot{\mathsf{V}} \triangleq \{\dot{v} | v \in \mathsf{V} \wedge type(v) = \mathbb{R}\}$. V' is used for specifying resets of discrete transitions and $\dot{\mathsf{V}}$ is used for specifying continuous dynamics. For a formula ϕ, let: (a) $vars(\phi)$ be the set of variables appearing in ϕ, (b) $ivars(\phi)$ be the set of distinct index variables appearing in ϕ.

Let N be a symbol representing an arbitrary natural number and i be a symbol representing an arbitrary element of $[\mathsf{N}]$. For the remainder of the paper, we fix N and refer to it implicitly in the remaining definitions. When clear from context, we drop the parameter N, for instance, a hybrid automaton template $\mathcal{A}(\mathsf{N}, i)$ is written $\mathcal{A}(i)$, etc.

Definition 1. *A* hybrid automaton template $\mathcal{A}(\mathsf{N}, i)$ *is specified by the syntactic components: (a)* $\mathsf{V}(i)$*: a finite set of variable names with associated types. (b)* L*: a finite set of location names. (c)* $\mathsf{Init}(i)$*: an initial condition formula over* $\mathsf{V}(i)$*. (d)* $\mathsf{Trans}(i)$*: a finite set of discrete* transition statements*, each of which is a tuple* $\langle \mathbf{from}, \mathbf{to}, \mathbf{grd}, \mathbf{rst} \rangle$*, where* $\mathbf{from}$*,* $\mathbf{to} \in \mathsf{L}$*,* $\mathbf{grd}$ *is a formula over* $\mathsf{V}(i)$ *called a* guard *and* $\mathbf{rst}$ *is a formula over* $\mathsf{V}(i) \cup \mathsf{V}'(i)$ *called a* reset*. The guard is an enabling condition that must be satisfied so that a transition* may *be taken, and the reset models the update of state. (e)* $\mathsf{Traj}(i)$*: for each element in* L*, there is a* trajectory statement*, each of which is a tuple* $\langle \mathbf{loc}, \mathbf{inv}, \mathbf{frate} \rangle$*, where* $\mathbf{loc} \in \mathsf{L}$*,* $\mathbf{inv}$ *is a formula over the real variables* $\mathsf{X}(i)$ *called an* invariant*, and* $\mathbf{frate}$ *is a formula over* $\mathsf{X}(i) \cup \dot{\mathsf{X}}(i)$ *called a* flowrate *that specifies how the real variables evolve over time. The invariant is an assertion that must be satisfied while* $\mathcal{A}(i)$ *is in* $\mathbf{loc}$*, and the flow rate associates each real-valued variable of* $\mathcal{A}(i)$ *with a rectangular differential inclusion.*

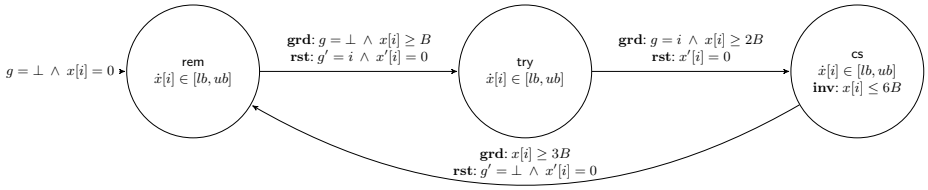

Fig. 1. Hybrid automaton template $\mathcal{A}(i)$ for MUX-INDEX-RECT mutual exclusion algorithm.

Let $\mathsf{X}(i) \triangleq \{v \in \mathsf{V}(\mathsf{N}, i) | type(v) = \mathbb{R}\}$ be the set of variables of $\mathcal{A}(i)$ with real type.

MUX-INDEX-RECT (Figure 1) is a timed mutual exclusion algorithm with an imprecise real clock $x[i]$ that evolves between rates $lb \leq ub$. There is a global variable g with type $[\mathsf{N}]_\perp$. Each automaton i starts in rem with $x[i] = 0$ and $g = \perp$, then after waiting B time i may enter try which also sets the global variable g to the identifier i. Then after waiting at least $2B$ time, it may enter the critical section cs after another $2B$ time and stay there for at least $3B$ and at most $6B$ time before returning to rem and setting $g = \perp$.

2.1 Semantics of Hybrid Automata Networks

For a hybrid automaton template $\mathcal{A}(i)$, we define a transition system to formalize the semantics of the network where N instantiations of $\mathcal{A}(i)$ operate concurrently.

Definition 2. *Let* N *be a symbol representing an arbitrary natural number. A* hybrid automata network *is a tuple* $\mathcal{A}^{\mathsf{N}} \triangleq \langle \mathsf{V}^{\mathsf{N}}, Q^{\mathsf{N}}, \Theta^{\mathsf{N}}, T^{\mathsf{N}} \rangle$*, where: (a)* V^{N} *are the variables of the network,* $\mathsf{V}^{\mathsf{N}} \triangleq \mathsf{V}_G \cup \bigcup_{i=1}^{\mathsf{N}} \mathsf{V}_L(i)$*, (b)* $Q^{\mathsf{N}} \subseteq val(\mathsf{V}^{\mathsf{N}})$ *is the state-space, (c)* $\Theta^{\mathsf{N}} \subseteq Q^{\mathsf{N}}$ *is the set of initial states, and (d)* $T^{\mathsf{N}} \subseteq Q^{\mathsf{N}} \times Q^{\mathsf{N}}$ *is the transition relation, which is partitioned into sets of discrete transitions* $\mathcal{D}^{\mathsf{N}} \subseteq Q^{\mathsf{N}} \times Q^{\mathsf{N}}$ *and continuous trajectories* $\mathcal{T}^{\mathsf{N}} \subseteq Q^{\mathsf{N}} \times Q^{\mathsf{N}}$.

A *state* $\mathbf{x}$ of $\mathcal{A}^{\mathsf{N}}$ is a valuation of *all* the variables in V^{N} and is denoted by boldface $\mathbf{v}$, $\mathbf{v}'$, etc. The set of all states is called the state-space and is denoted Q^{N}. If a state $\mathbf{v} \in Q^{\mathsf{N}}$ satisfies a formula ϕ—that is, the corresponding variable valuations result in ϕ evaluating to $true$—we write $\mathbf{v} \models \phi$. For a formula ϕ with $vars(\phi) \subseteq \mathsf{V}(i)$, the corresponding states $\mathbf{x} \in Q^{\mathsf{N}}$ satisfying ϕ are $[\![\phi]\!] \triangleq \{\mathbf{x} \in Q^{\mathsf{N}} | \mathbf{v} \models \phi\}$. For instance, the initial states $\Theta^{\mathsf{N}} \triangleq [\![\mathsf{Init}(i)]\!]$ are the states satisfying $\mathsf{Init}(i)$. For some state $\mathbf{v}$, the valuation of a particular local variable $x[i] \in \mathsf{V}_L(i)$ for automaton $\mathcal{A}(i)$ is denoted by $\mathbf{v}.x[i]$, and $\mathbf{v}.g$ for some global variable g in $\mathsf{V}_G(i)$. For a set of variables V, the valuations of each $v \in \mathsf{V}$ at state $\mathbf{v}$ is denoted by $\mathbf{v}.\mathsf{V}$. For a formula ϕ and a set of variables $\mathsf{V} \subseteq vars(\phi)$, let $\phi{\downarrow}\,\mathsf{V}$ be the projection of ϕ onto the variables V, such that $vars(\phi{\downarrow}\,\mathsf{V}) = \mathsf{V}$ and $[\![\phi]\!] \subseteq [\![\phi{\downarrow}\,\mathsf{V}]\!]$, which can be computed by eliminating the existential quantifiers from the formula $\exists vars(\phi) \setminus \mathsf{V} : \phi$. The evolution of the states of $\mathcal{A}^{\mathsf{N}}$ are describing by a transition relation $T^{\mathsf{N}} \subseteq Q^{\mathsf{N}} \times Q^{\mathsf{N}}$. For a pair $(\mathbf{v}, \mathbf{v}') \in T^{\mathsf{N}}$, we use the notation $\mathbf{v} \rightarrow \mathbf{v}'$, where $\mathbf{v}$ is called the *pre-state* and $\mathbf{v}'$ is called the *post-state*. There are two ways variables may be updated by T^{N}. Discrete transitions $\mathcal{D}^{\mathsf{N}}$ model instantaneous changes and continuous trajectories $\mathcal{T}^{\mathsf{N}}$ model evolution over a real time interval. When necessary to disambiguate state updates owing to either discrete transitions and continuous trajectories, we write $\mathbf{v} \rightarrow_{\mathcal{D}^{\mathsf{N}}} \mathbf{v}'$ or $\mathbf{v} \rightarrow_{\mathcal{T}^{\mathsf{N}}} \mathbf{v}'$, respectively.

Discrete Transitions. Discrete transitions model atomic, instantaneous updates of state due to *one* automaton in the network $\mathcal{A}^{\mathsf{N}}$. Informally, a discrete transition from pre-state $\mathbf{v}$ to post-state $\mathbf{v}'$ models the discrete transition of one particular hybrid automaton $\mathcal{A}(i)$ by some transition $\mathsf{t} \in \mathsf{Trans}(i)$. There is a discrete transition $\mathbf{v} \rightarrow \mathbf{v}' \in \mathcal{D}^{\mathsf{N}}$ iff: $\exists i \in [\mathsf{N}]\ \exists \mathsf{t} \in \mathsf{Trans}(i) : \mathbf{v}.\mathsf{V}(i) \models \mathbf{grd}(\mathsf{t}, i) \wedge \mathbf{v}'.\mathsf{V}(i) \models \mathbf{rst}(\mathsf{t}, i) \wedge (\forall j \in [\mathsf{N}] : j \neq i \Rightarrow \mathbf{v}'.\mathsf{V}(j) = \mathbf{v}.\mathsf{V}(j))$. From the pre-state $\mathbf{v}$, any automaton $\mathcal{A}(i)$ in the network $\mathcal{A}^{\mathsf{N}}$ that has some transition where $\mathbf{v}$ satisfies its guard *may* update its post-state according to the transition's reset, while the variable valuations of all the other automata in $\mathcal{A}^{\mathsf{N}}$ remain unchanged.[4]

Continuous Trajectories. Continuous trajectories model update of state over intervals of real time. Informally, there is a trajectory $\mathbf{v} \rightarrow \mathbf{v}' \in \mathcal{T}^{\mathsf{N}}$ iff some amount of time—t_e—can elapse from $\mathbf{v}$, such that, (a) the states of *all* automata in the network $\mathcal{A}^{\mathsf{N}}$ are updated to $\mathbf{v}'$ according to their individual trajectory statements, and (b) while ensuring the invariants of *all* automata along the entire trajectory. Formally, trajectories are defined as solutions of differential equations or inclusions specified in the trajectory statements of $\mathcal{A}(i)$. For a state $\mathbf{v}$, a location m, a real time t, and a real variable $v \in \mathsf{X}(i)$, let $\mathbf{flow}(\mathbf{v}, \mathsf{m}, v, t) = \mathbf{v}.v + \int_{\tau=0}^{t} \mathbf{frate}(\mathsf{m}, v) d\tau$. Since $\mathbf{frate}$ may specify a differential inclusion, $\mathbf{flow}$ is a set-valued function. There is a trajectory $\mathbf{v} \rightarrow \mathbf{v}' \in \mathcal{T}^{\mathsf{N}}$ iff: $\exists t_e \in \mathbb{R}_{\geq 0}\ \forall t_p \in \mathbb{R}_{\geq 0}\ \forall i \in [\mathsf{N}]\ \exists \mathsf{m} \in \mathsf{L} : t_p \leq t_e \wedge \mathbf{flow}(\mathbf{v}, \mathsf{m}, \mathsf{X}(i), t_p) \models \mathbf{inv}(\mathsf{m}, i) \wedge \mathbf{v}'.\mathsf{X}(i) \in \mathbf{flow}(\mathbf{v}, \mathsf{m}, \mathsf{X}(i), t_e)$. For each $i \in [\mathsf{N}]$ and each real variable $x[i]$, $\mathbf{v}.x[i]$ must evolve to the valuations $\mathbf{v}'.x[i]$, in exactly t_e time in some location $\mathsf{m} \in \mathsf{L}$ according to the flow rates allowed for $x[i]$ in location m. In addition, all intermediate states along the trajectory must satisfy the invariant $\mathbf{inv}(\mathsf{m}, i)$.

Executions and Invariants. An execution of the network $\mathcal{A}^{\mathsf{N}}$ models a particular behavior of all the automata in the network. An *execution* of $\mathcal{A}^{\mathsf{N}}$ is a sequence of states $\alpha = \mathbf{v}_0, \mathbf{v}_1, \ldots$ such that $\mathbf{v}_0 \in \Theta^{\mathsf{N}}$, and for each index k appearing in the sequence, $(\mathbf{v}_k, \mathbf{v}_{k+1}) \in T^{\mathsf{N}}$. A state $\mathbf{x}$ is *reachable* if there is a finite execution ending with $\mathbf{x}$. The set of reachable states for $\mathcal{A}^{\mathsf{N}}$ is $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}})$. The set of reachable states for $\mathcal{A}^{\mathsf{N}}$ starting from an arbitrary subset $\mathbf{V}_0 \subseteq Q^{\mathsf{N}}$ is $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}}, \mathbf{V}_0)$. An *invariant* for $\mathcal{A}^{\mathsf{N}}$ is any set of states that contains $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}})$.

3 Anonymized State-Space Representation

For any fixed $\mathsf{N} \in \mathbb{N}$, let i be a symbol representing an arbitrary element of $[\mathsf{N}]$, and for the hybrid automaton template $\mathcal{A}(\mathsf{N}, i)$, the composed automaton modeling a network of size N is $\mathcal{A}^{\mathsf{N}}$ (Definition 2). We present an algorithm for computing $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}})$ that takes advantage of the symmetries in the template $\mathcal{A}(i)$ instantiated in $\mathcal{A}^{\mathsf{N}}$. The representation of $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}})$ is *anonymized*, so numerical automaton indices—1, 2, …, N—are not explicitly enumerated and are instead modeled using symbolic indices—i_1, i_2, …, i_{N}. Frequently, the number of symbolic indices needed to represent equivalent states is significantly smaller than the number N of numerical indices. For example, in

[4] The guard may depend on the variables of some automaton $j \neq i$, so automata may communicate via global variables and local variables, for details, see [23, Chapter 2].

MUX-INDEX-RECT (Figure 1), a single symbolic index is sufficient independent of N. For a given state $\mathbf{x} \in Q^{\mathsf{N}}$, the set of corresponding states $\mathbf{X} \subseteq Q^{\mathsf{N}}$ that are equivalent modulo indices is obtained by substituting any numerical index i of all local variables $v[i] \in \mathsf{V}_L(i)$ with a symbolic index j with type $[\mathsf{N}]$.

Definition 3. *Two states* $\mathbf{x}$, $\mathbf{x}' \in Q^{\mathsf{N}}$ *of* $\mathcal{A}^{\mathsf{N}}$, *are* equivalent modulo indices *if there exists a bijection* $\pi : [\mathsf{N}] \to [\mathsf{N}]$ *such that for each* $v[i] \in \mathsf{V}(i)$, $\mathbf{x}.v[i] = \mathbf{x}'.v[\pi(i)]$. *For a state* $\mathbf{x} \in Q^{\mathsf{N}}$ *of* $\mathcal{A}^{\mathsf{N}}$, *the set of states* $\varepsilon(\mathbf{x})$ *that is equivalent modulo indices to* $\mathbf{x}$ *is:* $\varepsilon(\mathbf{x}) \triangleq \{\mathbf{x}' \in Q^{\mathsf{N}} \mid \mathbf{x}$ *and* $\mathbf{x}'$ *are equivalent modulo indices*$\}$.

We note this is the same type of definition as the existence of an automorphism used in [2–4]. A state is equivalent modulo indices to itself by picking the bijection π to be the identity mapping. For a formula ϕ, we will overload π and write $\pi(\phi)$, which modifies ϕ by applying π to each index variable $i \in \mathit{ivars}(\phi)$. The anonymized representation takes this idea a step further by utilizing symbolic names for process indices along with counters, and a formula representing the valuations of any global variables. We use the (.) notation to refer to particular elements of tuples. For example, `C.Count` refers to the count of anonymized class `C`, `C.Form` refers to `C`'s formula, etc. If `C` is clear from context, we refer to `C.Count` as `Count`, etc.

Definition 4. *An* anonymized state $\mathtt{S}$ *of the automaton network* $\mathcal{A}^{\mathsf{N}}$ *(Definition 2) is a tuple* $\langle \mathtt{Classes}, \mathtt{G} \rangle$, *where:*

(a) Each anonymized class $\mathtt{C} \in \mathtt{Classes}$ *is a tuple* $\mathtt{C} \triangleq \langle \mathtt{Count}, \mathtt{I}, \mathtt{Form} \rangle$, *where:*
  (i) $\mathtt{Form}$ *is a quantifier-free formula over the variables* $\mathsf{V}_L(i_1) \cup \ldots \cup \mathsf{V}_L(i_{\mathtt{I}})$, *where* $i_1, \ldots, i_{\mathtt{I}}$ *are* $\mathtt{I}$ *distinct symbolic index variables.*
  (ii) $\mathtt{I} \geq 1$ *is a natural number called the* class's rank, *which is equal to the number of distinct symbolic index variables appearing in* $\mathtt{Form}$*:* $\mathtt{I} \triangleq |\mathit{ivars}(\mathtt{Form})|$.
  (iii) $\mathtt{Count}$ *is a natural number called the* class's count, *and satisfies* $\mathsf{N} \geq \mathtt{Count} \geq |\mathtt{I}|$. *The count is the number of automata of class* $\mathtt{C}$. *Additionally, the sum of all the class counts in* $\mathtt{S}$ *equals* N*:* $\mathsf{N} = \sum_{\mathtt{C} \in \mathtt{S.Classes}} \mathtt{C.Count}$, *where* $\mathtt{C.Count}$ *is the count of class* $\mathtt{C}$.
(b) $\mathtt{G}$ *is a quantifier-free formula over the global variables* V_G.

For an anonymized class `C`, requirement (iii) of Definition 4 that $\mathtt{Count} \geq |\mathtt{I}|$ means the number of automata satisfying `Form` is at least the rank (the number of distinct index variables appearing in `Form`). When the rank `I` is clear from context, we drop it from the `C` tuple and write $\langle \mathtt{Count}, \mathtt{Form} \rangle$. We say two anonymized classes $\mathtt{C}_1$ and $\mathtt{C}_2$ over the same symbolic indices ($\mathit{ivars}(\mathtt{C}_1.\mathtt{Form}) = \mathit{ivars}(\mathtt{C}_2.\mathtt{Form})$) are equivalent and write $\mathtt{C}_1 \equiv \mathtt{C}_2$ iff they have equivalent class formulas and equal class counts:[5]

[5] It is possible for classes with different ranks to represent the same states. For example, consider states arising from MUX-INDEX-REST, $\mathtt{S}_1 = \langle \{\langle 2, 2, q[i_1] = \mathsf{rem} \wedge q[i_2] = \mathsf{rem} \rangle\}, g = \bot \rangle$ and $\mathtt{S}_2 = \langle \{\langle 2, 1, q[i_1] = \mathsf{rem} \rangle\}, g = \bot \rangle$, which both represent there are two automata with location rem and g is $\bot$, i.e., $[\![\mathtt{S}_1]\!] = [\![\mathtt{S}_2]\!]$. However, a class of a particular rank may not be expressible as a different rank. For example, there is no way to express the following using rank 1 classes: $\langle \{\langle 2, 2, q[i_1] = \mathsf{rem} \wedge q[i_2] = \mathsf{rem} \wedge x[i_1] \geq x[i_2] \rangle\}, g = \bot \rangle$, which expresses that there are two automata in rem with one's clock at least as large as the other's.

Definition 5. *Two classes* $\mathtt{C}_1$ *and* $\mathtt{C}_2$ *are* equivalent*, written* $\mathtt{C}_1 \equiv \mathtt{C}_2$ *iff* $\mathtt{C}_1.\mathtt{Count} = \mathtt{C}_2.\mathtt{Count} \wedge \mathtt{C}_1.\mathtt{Form} \equiv \mathtt{C}_2.\mathtt{Form}$.

Here, equivalence between the class formulas is a semantic and not syntactic notion, and means the formula $\mathtt{C}_1.\mathtt{Form} \equiv \mathtt{C}_2.\mathtt{Form}$ is valid.We say two anonymized states $\mathtt{S}_1$ and $\mathtt{S}_2$ are equivalent and write $\mathtt{S}_1 \equiv \mathtt{S}_2$ iff they have the same state counts, the classes in their sets of classes are equivalent, and their global formulas are equivalent. See Footnote 5 for an example from MUX-INDEX-RECT.

Definition 6. *Two anonymized states* $\mathtt{S}_1$ *and* $\mathtt{S}_2$ *are* equivalent*, written* $\mathtt{S}_1 \equiv \mathtt{S}_2$*, iff* $\forall\, \mathtt{C}_1 \in \mathtt{S}_1.\mathtt{Classes}\ \exists\, \mathtt{C}_2 \in \mathtt{S}_2.\mathtt{Classes}\ \mathtt{C}_1 \equiv \mathtt{C}_2 \wedge \mathtt{G}_1 \equiv \mathtt{G}_2$.

We make the following assumption about the format of class formulas.

Assumption 1. *For an anonymized state* $\mathtt{S}$*, for each class* $\mathtt{C} \in \mathtt{Classes}$*, the class formula* $\mathtt{C}.\mathtt{Form}$ *is in conjunctive normal form (CNF). For each index* $i \in \{i_1, \ldots, i_{\mathtt{C}.\mathtt{I}}\}$*,* $\mathtt{C}.\mathtt{Form}$ *contains an equality* $q[i] = \mathbf{loc}$ *for some location* $\mathbf{loc} \in \mathsf{L}$.

For example, Equation 1 (arsing from MUX-INDEX-RECT) satisfies this assumption. This assumption ensures that each class corresponds to a concrete state, and has a control location specified to determine the transitions and trajectories that may be possible (recall Definition 1). Under Assumption 1, the interpretation of an anonymized state $\mathtt{S}$ corresponds to a set of states of Q^{N}, which we write as $[\![\mathtt{S}]\!]$ and define formally next. Since the class formulas of $\mathtt{S}$ are over the variables of automata with symbolic indices, the interpretation instantiates the symbolic indices with specific elements of $[\mathsf{N}]$, which yields the set of states that are equivalent modulo indices.

Definition 7. *For an anonymized state*

$$\mathtt{S} = \langle \{ \underbrace{\langle \mathtt{Count}_1, \mathtt{I}_1, \mathtt{Form}_1 \rangle}_{\mathtt{C}_1}, \ldots, \underbrace{\langle \mathtt{Count}_k, \mathtt{I}_k, \mathtt{Form}_k \rangle}_{\mathtt{C}_k} \}, \mathtt{G} \rangle,$$

we instantiate the set of symbolic indices $\{i_1, \ldots, i_{\mathtt{I}_k}\}$ *with all possible values in* $[\mathsf{N}]$ *as follows. A* consistent partition *of* $[\mathsf{N}]$*,*

$$P = \{ \underbrace{\{P_1^1, \ldots, P_1^{\mathtt{I}_1}\}}_{P_1}, \ldots, \underbrace{\{P_k^1, \ldots, P_k^{\mathtt{I}_k}\}}_{P_k} \},$$

is a partition of $[\mathsf{N}]$*, such that, for any* $P_j \in P$*, (a)* $|P_j| = \mathtt{Count}_j$ *and (b)* P_j *is partitioned into* $\mathtt{I}_j$ *sets* $P_j^1, \ldots, P_j^{\mathtt{I}_j}$ *(and we recall that* $\mathtt{I}_j$ *is the rank of* $\mathtt{C}_j$*).*

For a consistent partition P, we note that (a) $\sum_{P_j \in P} |P_j| = \mathsf{N}$, since P partitions $[\mathsf{N}]$, and (b) $\mathtt{Count}_j \geq \mathtt{I}_j$ (by Definition 4, (iii)). For example, consider the anonymized state (arsing from MUX-INDEX-RECT, Figure 1) with count three and rank two:

$$\langle \{ \langle 3, 2, q[i_1] = \mathsf{try} \wedge q[i_2] = \mathsf{rem} \wedge x[i_1] \geq x[i_2] + B \rangle \}, g = i_1 \rangle. \tag{1}$$

One consistent partition is: $P = \{P_1, P_2\}$ where $P_1 = \{1\}$ and $P_2 = \{2, 3\}$. The set $\{\{1, 2, 3\}\}$ is *not* a consistent partition since it is partitioned into one set, but the rank $\mathtt{I} = 2$, and Definition 7 requires each $P_j \in P$ be partitioned into $\mathtt{I}_j$ partitions.

For an anonymized state $\mathtt{S}$, the set of consistent partitions $ConsPart(\mathtt{S})$ are all consistent partitions of $[\mathsf{N}]$. Continuing MUX-INDEX-RECT for Equation 1, $ConsPart(\mathtt{S})$ is $\{\{\{1\},\{2,3\}\}, \{\{2\},\{1,3\}\}, \{\{3\},\{1,2\}\}, \{\{1,2\},\{3\}\}, \{\{1,3\},\{2\}\}, \{\{2,3\},\{1\}\}\}$. All these partitions define the set of states $[\![\mathtt{S}]\!]$ the anonymized state $\mathtt{S}$ represents. This is the same as all the states equivalent modulo indices to the states $[\![\mathtt{S}_P]\!]$ for a particular consistent partition P.

Definition 8. *For an anonymized state* $\mathtt{S}$ *and a consistent partition* $P \in ConsPart(\mathtt{S})$*, the set of states of network* $\mathcal{A}^{\mathsf{N}}$ *represented by* $\mathtt{S}$ *corresponding to* P *are:*

$$[\![\mathtt{S}_P]\!] \triangleq \{\mathbf{x} \in Q^{\mathsf{N}} \mid \mathbf{x} \models \mathtt{G} \wedge \mathtt{Form}_1(P_1) \wedge \ldots \wedge \mathtt{Form}_k(P_k)\}, \tag{2}$$

where each $\mathtt{Form}_j(P_j) \triangleq \forall i_j^1 \in P_j^1, \ldots, i_j^{\mathtt{I}_j} \in P_j^{\mathtt{I}_j} : \mathtt{Form}_j(i_j^1, \ldots, i_j^{\mathtt{I}_j})$*. The set of states of network* $\mathcal{A}^{\mathsf{N}}$ *represented by* $\mathtt{S}$ *with all consistent partitions is:*

$$[\![S]\!] \triangleq \bigcup_{P \in ConsPart(S)} [\![\mathtt{S}_P]\!]. \tag{3}$$

We have written $\mathtt{Form}_j(i_j^1, \ldots, i_j^{\mathtt{I}_j})$ to highlight that $\mathtt{Form}_j$ is over $\mathtt{I}_j$ symbolic index variables. Note that $\mathtt{Form}_j(P_j)$ is equivalent to a finite-length conjunction since each $P_j^{\mathtt{I}_j}$ is a finite set. The next lemma states that this definition of interpretations of anonymized states yields the same set of states as equivalence modulo identifiers.

Lemma 1. *For an anonymized state* $\mathtt{S}$*, for any* $\mathbf{x} \in [\![\mathtt{S}]\!]$*, for any* $\mathbf{x}' \in \varepsilon(\mathbf{x})$*,* $\mathbf{x}' \in [\![\mathtt{S}]\!]$*.*

Continuing the MUX-INDEX-RECT Equation 1 example with the consistent partition $P = \{\{1\},\{2,3\}\}$, the states represented by $\mathtt{S}_P$ are:

$$\begin{aligned}
[\![\mathtt{S}_P]\!] = \; & \{\mathbf{x} \in Q^3 \mid \mathbf{x} \models \forall i_1^1 \in P_1^1, i_1^2 \in P_1^2 \; : \; q[i_1] = \mathsf{try} \; \wedge q[i_2] = \mathsf{rem} \wedge \\
& \qquad x[i_1] \geq x[i_2] + B \; \wedge \; g = i_1\} \\
= \; & \{\mathbf{x} \in Q^3 \mid \mathbf{x} \models (q[1] = \mathsf{try} \wedge q[2] = \mathsf{rem} \wedge q[3] = \mathsf{rem} \wedge \\
& \qquad x[1] \geq x[2] + B \wedge x[1] \geq x[3] + B \; \wedge \; g = 1) \; \}.
\end{aligned}$$

Applying Lemma 1, $[\![\mathtt{S}]\!] = \varepsilon([\![\mathtt{S}_P]\!])$.

4 Anonymized Reachability of Hybrid Automata Networks

Next we describe an on-the-fly algorithm for overapproximating the reachable states of a network $\mathcal{A}^{\mathsf{N}}$ using anonymized states. We note that the CNF requirement (Assumption 1) is not restrictive: if a new class is created during the execution of the algorithm that contains disjunctions, it is split into multiple classes with CNF formulas. Recall from Section 2.1, that $\phi{\downarrow}\,\mathsf{V}$ is the projection of ϕ onto the variables V.

Pseudocode for the reachability algorithm, areach appears in Figure 2. The algorithm operates on frontiers of reachable states represented by the set Frontier, which is initialized (line 3) to a singleton set with one class that has count equal to N and class formula equal to $\mathsf{Init}(i){\downarrow}\,\mathsf{V}_L(i)$, which is $\mathsf{Init}(i)$ projected onto the local variables. The global formula is initialized with $\mathsf{Init}(i){\downarrow}\,\mathsf{V}_G(i)$, which is $\mathsf{Init}(i)$ projected

```
1   function areach(𝒜(i), Init(i), N)
      AnonReach ← ∅
3     Frontier ← {⟨{⟨N, Init(i)↓ V_L(i)⟩}, Init(i)↓ V_G(i)⟩} // initial anonymized state
      while Frontier ≠ ∅ // repeat until no new states are added to the frontier
5       Frontier_New ← ∅ // initialize next frontier
        AnonReach ← AnonReach ∪ Frontier // add frontier to reachable states
7       // compute successors of each anonymized state in the frontier
        foreach anonymized state S in Frontier
9         Frontier_New ← Frontier_New ∪ discPost(S) // Figure 4
          Frontier_New ← Frontier_New ∪ contPost(S) // Figure 5
11      Frontier_New ← mergeAndDrop(Frontier_New, AnonReach) // Figure 3
        Frontier ← Frontier_New
13    return AnonReach
```

Fig. 2. On-the-fly anonymized reachability algorithm. The inputs are an automaton template $\mathcal{A}(i)$, an initial condition $\mathsf{Init}(i)$, and a constant natural number N. The anonymized reachable states AnonReach are computed as a fixed-point starting from the anonymized initial states.

onto the global variables. The set of reachable anonymized states computed so far is the set AnonReach. Next (line 4), we remove an anonymized state S from Frontier, compute anonymized post-states from S, and continue until no new anonymized states are added to Frontier. Anonymized post-states are added to the frontier using the set $\mathsf{Frontier}_{\mathtt{New}}$ (line 5). Computing successors (post-states)—the states reachable from S in one step—is composed of two parts: (a) computing the discrete successors corresponding to transitions (line 9), and (b) computing the continuous successors corresponding to trajectories (line 10).

Equivalent Class Merging Subroutine. We first describe the mergeAndDrop subroutine (Figure 3). It takes a set of anonymized states $\mathsf{Frontier}_{\mathtt{New}}$ and returns a set of anonymized states that is guaranteed to both (a) not have any equivalent classes (lines 7 through 8) and (b) be new (not already represented in AnonReach) (line 3). Invariant 1 states that no two class formulas in any reachable anonymized state are equivalent, and Invariant 2 states that no two anonymized states in AnonReach are equivalent (Definition 6).

Invariant 1. *For any* $\mathtt{S} \in \mathsf{AnonReach}$, $\mathtt{C}_1, \mathtt{C}_2 \in \mathtt{S.Classes}$, $\mathtt{C}_1.\mathtt{Form} \not\equiv \mathtt{C}_2.\mathtt{Form}$.

Invariant 2. *For any distinct* $\mathtt{S}_1, \mathtt{S}_2 \in \mathsf{AnonReach}$, $\mathtt{S}_1 \not\equiv \mathtt{S}_2$.

Discrete Successors. The function discPost (Figure 4) computes the discrete successors from an anonymized state S in the frontier (Figure 2, line 9). The post-states $\mathsf{States}_{\mathtt{New}}$ are added to the frontier $\mathsf{Frontier}_{\mathtt{New}}$. First, we iterate over each class C in S.Classes (line 3), and then we iterate over each index variable i in the set of index variables in the class formula, $\{i_1, \ldots, i_{\mathtt{C.I}}\}$ (line 5). Next, we iterate over the (syntactic) transitions $\mathsf{Trans}(i)$ of $\mathcal{A}(i)$ (line 6). For a transition $\mathtt{t} \in \mathsf{Trans}(i)$ and an anonymized class C, line 7 computes the subsequent class from C by transition t, made

```
1   function mergeAndDrop(Frontier_New, AnonReach)
      foreach S in Frontier_New
3       if S ∈ AnonReach then Frontier_New = Frontier_New \ {S}
        else
5         foreach distinct pair of anonymized classes ⟨C1, C2⟩ in S.Classes
            if ¬(C1.Form ≡ C2.Form) is UNSAT then
7             C1.Count ← C1.Count + C2.Count // if equivalent, sum counts
              S.Classes ← S.Classes \ {C2} // if equivalent, drop equivalent class
9     return Frontier_New
```

Fig. 3. mergeAndDrop combines classes with equivalent class formulas and sums their counts.

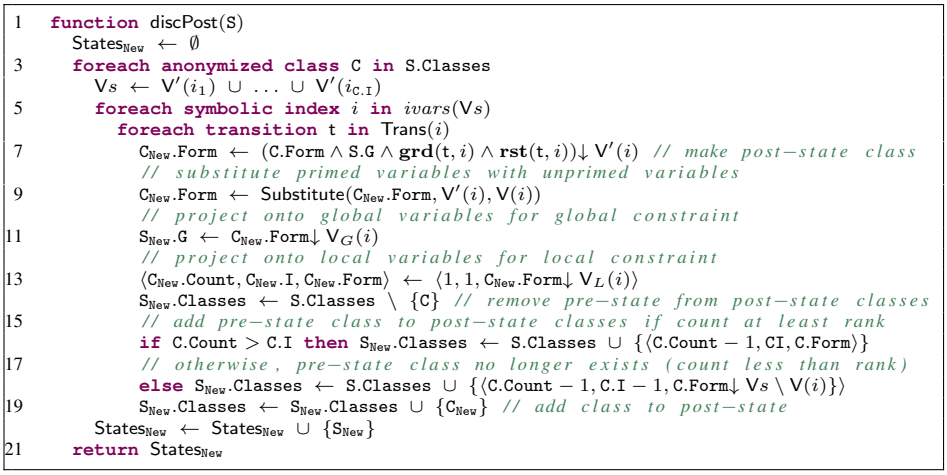

Fig. 4. discPost computes the post-states of an anonymized state S due to discrete transitions for an automaton with index i and states satisfying C's formulas.

by the automaton with index i. This computation can be carried out using quantifier elimination procedures over the types of the variables appearing in the guard and reset of the transition t, and then syntactically unpriming all primed variables (representing successors) following quantifier elimination using Substitute (line 9). This step is an overapproximation, since it computes the successors of each class regardless of the number of automata with states satisfying the anonymized class formula Form, and just presumes *there is some* automaton with variable valuations satisfying Form.

The anonymized post-state $\mathtt{S}_{\mathtt{New}}$ is constructed using the classes of the anonymized pre-state S along with the new anonymized class, $\mathtt{C}_{\mathtt{New}}$ (lines 14 through 19). First, the classes for $\mathtt{S}_{\mathtt{New}}$ are set to be the anonymized classes of S, without the anonymized class of the current iteration, C (line 14). Next, if the class count of C is larger than its rank, then it is added to the classes of the post-state, with its count reduced by one to indicate some automaton has left the set of states satisfying the corresponding class formula (line 16). On the other hand, if the class count is equal or less than its rank, then the pre-state's anonymized class C would no longer satisfy the requirements of Definition 4, (iii), so its class formula is projected onto the variables of all automata except those of automaton i, the one making a transition (line 18). If a class has count or rank equal to 0, then it is removed. This process may result in two classes with equivalent formulas, since the algorithm has not yet detected if any other classes had the same formula and presumed the post-state class $\mathtt{C}_{\mathtt{New}}$ had a count of one, which is why we use mergeAndDrop (Figure 2, line 11).

Lemma 2. *(Discrete Successor Soundness) For an anonymized state* S*, for any corresponding concretized state* $\mathbf{x} \in [\![\mathtt{S}]\!]$*, if* $\mathbf{x} \rightarrow_{\mathcal{D}^{\mathtt{N}}} \mathbf{x}'$*, then* $\mathbf{x}' \in [\![\mathsf{discPost}(\mathtt{S})]\!]$.

Continuous Successors. An overapproximation of continuous successors are computed using contPost—shown in Figure 5—called from symreach (Figure 2, line 10). For an anonymized state S in the frontier, contPost computes an overapproximation of the post-states from S owing to the individual trajectories of all automata in the network for up to the most amount of time that can elapse before any invariant is vi-

```
1   function contPost(S)
      Vs ← V'_G
3     // formula to encode trajectories for all automata in the network
      pf ← (t_e > 0 ∧ S.G)
5     foreach anonymized class C in S.Classes // iterate over each class in pre−state
        Vs ← Vs ∪ V'(i_1) ∪ ... ∪ V'(i_C.I)
7       pf ← pf ∧ C.Form // encode pre−state class formula
        // determine locations any automaton may be in (recall Assumption 1)
9       foreach location loc in L
          foreach i in {i_1, ..., i_C.I} // iterate over all indices (ranks)
11          if C.Form ⇏ (q[i] = loc) is UNSAT then // use loc if automaton i is in loc
              // add the trajectory semantics overapproximating the post−states
13            pf ← pf ∧ inv(loc, i) ∧ X(i) ∈ flow(pf, loc, X(i), t_e)
      pf ← pf↓ Vs
15    pf ← Substitute(pf, V'(i), V(i))
      S_New ← RemapClasses(S, pf) // Figure 6
17    return S_New
```

Fig. 5. contPost function that computes the continuous successors from an anonymized state S.

```
1   function RemapClasses(S, pf)
      S_New.Classes = ∅
3     foreach anonymized class C in S.Classes
        // project pf onto variables of indices in each pre−state class
5       Vs ← V(i_1) ∪ ... ∪ V(i_C.I)
        // create new class with post−state formula and copy pre−state count
7       ⟨C_New.Count, C_New.I, C_New.Form⟩ ← ⟨C.Count, C_New.I, pf↓ Vs⟩
        S_New.Classes ← S_New.Classes ∪ C_New // add post−state class to classes
9     S_New.N ← S.N
      return S_New
```

Fig. 6. RemapClasses recreates variables in pf using their pre-state indices, class counts, and ranks to create the anonymized post-state $\mathsf{S}_{\mathtt{New}}$. It first projects onto the variables with indices of each class in the pre-state and then uses the pre-state count to ensure class counts remain constant over trajectories.

olated. The anonymized state specifies a location $\mathbf{loc} \in \mathsf{L}$ for each automaton in the network (recall Assumption 1). Each location $\mathbf{loc}$ specifies a trajectory statement, so trajectories are defined for each automaton in the network. Each new anonymized state $\mathsf{S}_{\mathtt{New}} \in \mathsf{States}_{\mathtt{New}}$ computed corresponds to the trajectory semantics updating the real variables of *all* automata in the network $\mathcal{A}^{\mathsf{N}}$. The variable pf encodes the trajectory semantics of all automata in the network $\mathcal{A}^{\mathsf{N}}$ (line 4), which is initially the constraint $t_e > 0$, indicating that some positive real amount of time t_e will elapse. However, for an anonymized state S, for distinct anonymized classes $\mathtt{C}_1$, $\mathtt{C}_2$ in S.Classes, the symbolic indices appearing in the formulas may be equal, i.e., $\exists i \in \mathit{ivars}(\mathtt{C}_1)$ and $\exists j \in \mathit{ivars}(\mathtt{C}_2)$ such that $i = j$. Since pf encodes the states of all automata in the network, the symbolic index variables appearing in any class formula of any anonymized class must be distinct. Rather than performing these tedious syntactic manipulations, we assume that for an anonymized state S, for distinct classes $\mathtt{C}_1$, $\mathtt{C}_2$ in S.Classes, $\forall i \in \mathit{ivars}(\mathtt{C}_1)$, $\forall j \in \mathit{ivars}(\mathtt{C}_2)$, we have $i \neq j$.[6]

Each anonymized class formula C.Form of an anonymized state S specifies the location(s) of the automata, so the first step is to determine the dynamics that will modify each class formula. This is accomplished by first determining the appropriate flow-rate

[6] This is a tedious, but trivial invariant that we maintain in our implementation in *Passel*, so we make this assumption for clarity of presentation only.

conditions to use for each class in S.Classes, which can be detected by finding which Form implies the location variable $q[i]$ is in some location $\mathbf{loc} \in \mathsf{L}$. If the control location of automaton i is found to be equal to location $\mathbf{loc}$, then the trajectory statement of location $\mathbf{loc}$ is used to define the semantics of the time-evolution of i's real variables (line 13). The semantics of trajectories result in *all* the automata's real variables evolving over time t_e, so the formula encoding the trajectory statements of all automata is conjuncted (line 13). The post-states are computed by projecting onto the primed variables of all classes, and then renaming primed variables with their unprimed counterparts (line 15).[7] We call RemapClasses with the pre-state S and pf, which encodes the post-state constraints, to recreate classes from sub-formulas of pf (Figure 6 called at line 16). This is done to ensure the class counts are constant when computing post-states due to trajectories.

Lemma 3. *(Continuous Successor Soundness) For an anonymized state* S*, for any corresponding concretized state* $\mathbf{x} \in [\![\mathsf{S}]\!]$*, if* $\mathbf{x} \rightarrow_{\mathcal{T}^{\mathsf{N}}} \mathbf{x}'$*, then* $\mathbf{x}' \in [\![\mathsf{contPost}(\mathsf{S})]\!]$.

The next invariant states the sum of all class counts equals N. It follows from the definitions of discPost and contPost, since discPost always decreases class counts by the same amount it increases them—so the sum remains invariant—and contPost does not change class counts (only formulas). Additionally, mergeAndDrop changes class counts, but their sum remains the same since it removes any duplicate classes after adding their counts (Figure 3, lines 7 through 8).

Invariant 3. *For any* $\mathsf{S} \in \mathsf{AnonReach}$, $\mathsf{N} = \sum_{\mathtt{C} \in \mathtt{S.Classes}} \mathtt{C.Count}$.

Theorem 1 states soundness of the algorithm: the concretization of the anonymized reachable states AnonReach contains the reachable states for network $\mathcal{A}^{\mathsf{N}}$. It follows from Lemmas 2 and 3. The approximation comes from: (a) transitions are allowed as long as *some* automaton satisfies a guard, (b) index-typed variables are abstracted to be equal or not equal only, and (c) rectangular dynamics are overapproximated.

Theorem 1. *(Soundness) For a fixed* $\mathsf{N} \in \mathbb{N}$*, for the network* $\mathcal{A}^{\mathsf{N}}$ *composed of* N *instantiations of the template* $\mathcal{A}(\mathsf{N}, i)$*, the anonymized reachable states* AnonReach *computed by* areach *overapproximate the reachable states of* $\mathcal{A}^{\mathsf{N}}$*:* $\mathsf{Reach}(\mathcal{A}^{\mathsf{N}}) \subseteq [\![\mathsf{AnonReach}]\!]$.

5 Experimental Results

The anonymized reachability algorithm is implemented in *Passel* [21–23]. The current implementation of *Passel* uses the SMT solver Z3 [24] for proving validity, checking satisfiability, and performing quantifier elimination. *Passel* is written in C# and uses the managed .NET API to Z3, with experimental results reported using version 4.1. *Passel* proves validity of a formula ϕ by checking unsatisfiability of $\neg\phi$. The variables $\mathsf{V}(i)$ used in defining $\mathcal{A}(i)$ are specified to the SMT solver. Each local variable $v[i] \in \mathsf{V}_L(i)$ is modeled as an uninterpreted function $v : [\mathsf{N}] \rightarrow \mathit{type}(v)$. *Passel* automatically generates and asserts trivial data-type lemmas that the SMT solver requires. The experiments were conducted in an Ubuntu 12.04 VMWare virtual machine with 4 GB RAM

[7] This may result in a DNF formula, and if so, each conjunctive clause is added as a new anonymized state by iterating over the conjunctive clauses so all class formulas are CNF.

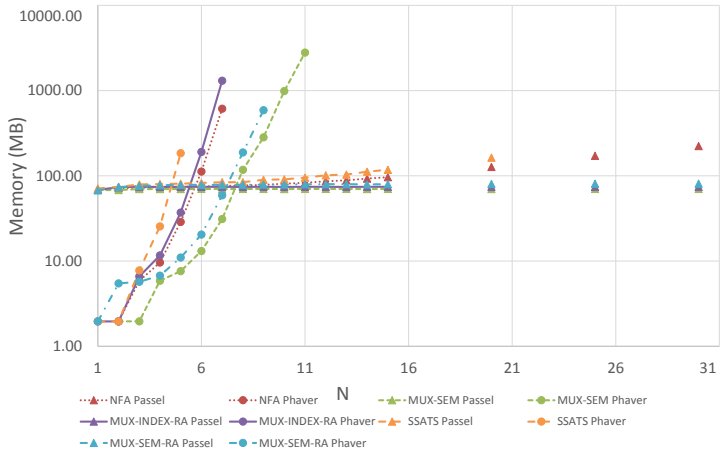

Fig. 7. Memory usage comparison of PHAVer and *Passel*'s anonymized reachability. Vertical axis scale is logarithmic and has units of megabytes, and horizontal axis is number of automata, N.

allocated running *Passel* through Mono, executed on a modern laptop with a quad-core Intel i7 processor running Windows 8 with 16 GB RAM physically available. For comparison purposes, we evaluated *Passel*, PHAVer (version 0.38), and SpaceEx (version 0.9.8b). We do not present results for SpaceEx, as the only scenario—out of the PHAVer, LeGuernic-Girard (LGG), and STC scenarios—that can compute the reachable states of systems with rectangular differential inclusion dynamics ($\dot{x} \in [a, b]$ for real constants $a \leq b$) adequately is the PHAVer scenario, so the results are equivalent.

Figures 7 and 8 show, respectively, a runtime and memory usage comparison between PHAVer and *Passel* for several examples as a function of N, the number of automata.[8] The examples include several timed mutual exclusion algorithms (such as MUX-INDEX-RECT from Figure 1), a simplified SATS model [21–23], and several purely discrete examples. All properties were safety properties (invariants), such as mutual exclusion, separation (collision avoidance) in SATS, etc. Comparing all the examples, the anonymized reachability method implemented in *Passel* allows us to compute the reachable states of networks composed of many more automata than PHAVer, which runs out of memory on all examples for $\mathsf{N} \geq 11$. The experimental results indicate that the primary advantage is reduced memory growth. Even for networks of tens of automata, in all examples, *Passel* never uses more than a few hundred megabytes of memory as shown in Figure 7.[9] For protocols that are highly asymmetric, the worst-case asymptotic memory growth may be exponential. The runtime required by *Passel* could be reduced by performing some operations more efficiently in the implementation—particularly the checks to determine if a new anonymized state representation is actually new or not—which we plan to implement for future work.

For MUX-INDEX-RECT, PHAVer runs out of memory for $\mathsf{N} \geq 8$. As shown in Figures 7 and 8, for $\mathsf{N} = 7$, PHAVer uses over 1.3 GB memory and completes in over 3

[8] *Passel* and the examples may be downloaded from: https://publish.illinois.edu/passel-tool/.

[9] For small N, PHAVer uses less memory than *Passel* because *Passel* must load runtime components (e.g., the .NET framework via Mono) and libraries (e.g., Z3).

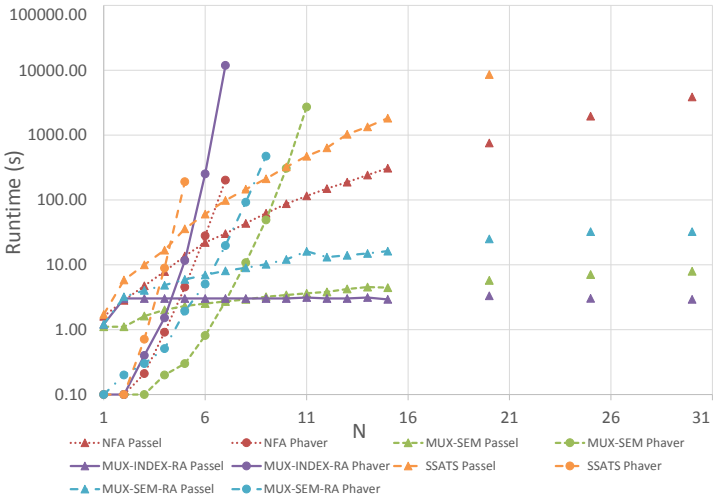

Fig. 8. Runtime comparison of PHAVer and *Passel*'s anonymized reachability. Vertical axis is logarithmic and has units of seconds, and horizontal axis is number of automata, N.

hours, while *Passel* uses over an order of magnitude less memory at about 70 MB and nearly four orders of magnitude less runtime at about three seconds. Because of the anonymized representation, *Passel* is able to compute the reachable states of $\mathsf{N} = 30$ in a few seconds using about 70 MB memory. For this example, the number of anonymized classes in `ReachForms` does not increase as a function of N. Additionally, for any $\mathtt{S} \in \mathsf{AnonReach}$, the sum of the class counts of S is 1, N, or $\mathsf{N} - 1$.[10]. Due to this state-space size independence from N, our experiments have been successful for computing the reachable states for compositions of hundreds of automata.

6 Summary

In this paper, we present an on-the-fly forward reachability algorithm that computes an anonymized representation of the reachable states for hybrid automata networks consisting of N instantiations of a template $\mathcal{A}(\mathsf{N}, i)$. The anonymized representation uses symbolic automato indices instead of explicit ones to avoid generating all permutations of automata indices and states. We showed it to be effective at computing the reachable states of networks with tens of automata for several examples, with significantly lower memory usage than PHAVer. The restriction to rectangular inclusion dynamics is due in part to *Passel*'s implementation, but a future direction is to evaluate the anonymized reachability method on examples with linear and nonlinear dynamics.

References

1. T. T. Johnson and S. Mitra, "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.
2. E. M. Clarke, R. Enders, T. Filkorn, and S. Jha, "Exploiting symmetry in temporal logic model checking," *Formal Methods in System Design*, vol. 9, pp. 77–104, 1996.

[10] Of course, these counts are a function of N, but the number of distinct counts is independent of N, which is not the case in general or the other examples.

3. C. N. Ip and D. L. Dill, "Better verification through symmetry," *Formal Methods in System Design*, vol. 9, pp. 41–75, 1996.
4. E. A. Emerson and A. P. Sistla, "Symmetry and model checking," *Formal Methods in System Design*, vol. 9, no. 1-2, pp. 105–131, 1996.
5. C. N. Ip and D. L. Dill, "Verifying systems with replicated components in Murφ," *Formal Methods in System Design*, vol. 14, no. 3, May 1999.
6. V. Braberman, D. Garbervetsky, and A. Olivero, "Improving the verification of timed systems using influence information," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, J.-P. Katoen and P. Stevens, Eds. Springer, 2002, vol. 2280, pp. 21–36.
7. G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen, "Static guard analysis in timed automata verification," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, H. Garavel and J. Hatcliff, Eds. Springer, 2003, vol. 2619, pp. 254–270.
8. M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager, "Adding symmetry reduction to UPPAAL," in *Formal Modeling and Analysis of Timed Systems (FORMATS '03)*, ser. LNCS, K. G. Larsen and P. Niebert, Eds., no. 2791. Springer–Verlag, 2004, pp. 46–59.
9. E. Emerson and T. Wahl, "Dynamic symmetry reduction," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, N. Halbwachs and L. Zuck, Eds. Springer, 2005, vol. 3440, pp. 382–396.
10. W. D. Obal, M. McQuinn, and W. Sanders, "Detecting and exploiting symmetry in discrete-state Markov models," *Reliability, IEEE Transactions on*, vol. 56, no. 4, pp. 643–654, Dec. 2007.
11. T. Wahl, N. Blanc, and E. Emerson, "SVISS: Symbolic verification of symmetric systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, C. Ramakrishnan and J. Rehof, Eds. Springer, 2008, vol. 4963, pp. 459–462.
12. G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening, "Symbolic counter abstraction for concurrent software," in *Computer Aided Verification*, ser. LNCS, A. Bouajjani and O. Maler, Eds. Springer, 2009, vol. 5643, pp. 64–78.
13. J. Sun, Y. Liu, J. S. Dong, Y. Liu, L. Shi, and E. André, "Modeling and verifying hierarchical real-time systems using stateful timed csp," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 1–29, Mar. 2013.
14. Y. Si, J. Sun, Y. Liu, and T. Wang, "Improving model checking stateful timed csp with non-zenoness through clock-symmetry reduction," in *Formal Methods and Software Engineering*, ser. LNCS, L. Groves and J. Sun, Eds. Springer, 2013, vol. 8144, pp. 182–198.
15. D. L. Dill, "The murφ verification system," in *Proceedings of the 8th International Conference on Computer Aided Verification*, ser. CAV '96. London, UK, UK: Springer-Verlag, 1996, pp. 390–393.
16. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL: A tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. LNCS, R. Alur, T. Henzinger, and E. Sontag, Eds. Springer, 1996, vol. 1066, pp. 232–243.
17. J. Sun, Y. Liu, J. Dong, and J. Pang, "PAT: Towards flexible verification under fairness," in *Computer Aided Verification*, ser. LNCS, A. Bouajjani and O. Maler, Eds. Springer, 2009, vol. 5643, pp. 709–714.
18. M. Hendriks, "Model checking timed automata: Techniques and applications," Ph.D. dissertation, University of Nijmegen, The Netherlands, 2006.
19. C. Herrera, B. Westphal, S. Feo-Arenis, M. Muñiz, and A. Podelski, "Reducing quasi-equal clocks in networks of timed automata," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS, M. Jurdzinski and D. Nickovic, Eds. Springer, 2012, vol. 7595, pp. 155–170.
20. S. Bogomolov, C. Herrera, M. Muñiz, B. Westphal, and A. Podelski, "Quasi-dependent variables in hybrid automata," in *17th International Conference on Hybrid Systems: Computation and Control*, 2014.
21. T. T. Johnson and S. Mitra, "A small model theorem for rectangular hybrid automata networks," in *Proceedings of the IFIP International Conference on Formal Techniques for Distributed Systems, Joint 14th Formal Methods for Open Object-Based Distributed Systems and 32nd Formal Techniques for Networked and Distributed Systems (FMOODS-FORTE)*, ser. LNCS. Springer, June 2012, vol. 7273.
22. ——, "Invariant synthesis for verification of parameterized cyber-physical systems with applications to aerospace systems," in *Proceedings of the AIAA Infotech at Aerospace Conference (AIAA Infotech 2013)*, Boston, MA, Aug. 2013.
23. T. T. Johnson, "Uniform verification of safety for parameterized networks of hybrid automata," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana, IL 61801, 2013.
24. L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS '08/ETAPS '08. Springer-Verlag, 2008, pp. 337–340.