

“ReLIC: Reduced Logic Inference for Composition” for Quantifier Elimination based Compositional Reasoning

Hao Ren¹, Ratnesh Kumar² and Matthew A. Clark³

¹*Honeywell Aerospace Advanced Technology, 12001 Hwy 55, Plymouth, MN, U.S.A.*

²*Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, U.S.A.*

³*Galois Inc., 444 E 2nd St, Dayton, OH, U.S.A.*

Keywords: Quantifier Elimination, Compositional Verification, System Property Composition.

Abstract: We present our work on the role and integration of quantifier elimination (QE) for compositional verification. In our approach, we derive in a single step, the strongest system property from the given component properties. This formalism is first developed for time-independent properties, and later extended to the case of time-dependent property composition. The extension requires additional work of replicating the given properties by shifting those along time so the entire time-horizon of interest is captured. We show that the time-horizon of a system property is bounded by the sum of the time-horizons of the component properties. The system initial condition can also be composed, which, alongside the strongest system property, are used to verify a postulated system property through induction. The above approaches are implemented in our prototype tool called ReLIC (Reduced Logic Inference for Composition) and demonstrated through several examples.

1 INTRODUCTION


Compositional reasoning is employed for scaling the model-based verification approaches for distributed systems. The component-based compositional design paradigm emphasizes the separation of concerns with respect to the system design through a modular and reuse based paradigm for defining, implementing, and composing components into systems.


For example, Unmanned Systems Autonomy Services (UxAS) software system (Rasmussen et al., 2016) consists of a collection of modular services that interact via a common message passing architecture. It provides a framework to construct and deploy software services that are used to enable autonomous capabilities by flexibly implementing autonomy algorithms on-board Unmanned Aerial Vehicles (UAV)s (Kingston et al., 2016). Verification of such systems seeks to utilize component properties to establish the composed properties of a system.


Many properties in systems and controls can be formulated as formulae in the first-order logic of real closed field. Quantifier elimination (QE) (Tarski,

1998) is a powerful technique for gaining insight in system properties, through simplification, by way of projection where a formula with some quantified variables is projected to a lower dimension over only its free variables.

In our previous work (Ren et al., 2016; Ren et al., 2018), we established that QE provides a foundation for compositional reasoning, and also can be used to aid formal verification and model-checking. Initially, during component development phase, each component is annotated with an assume-guarantee style contract (Emmi et al., 2008; Quinton and Graf, 2008). Supposing a system is composed of N components, the contract formula of the i th component can be expressed as $A_i \Rightarrow G_i$ where A_i (the “assumption”), G_i (it’s “guarantee”) are both expressed by formulae over the set of component variables. Then the set of all the system behaviors is constrained by the conjunction of all the components’ contracts $\bigwedge_{i=1}^N (A_i \Rightarrow G_i)$. Under these contracts, we show that “the strongest system property”, that can be claimed that the system satisfies, can be obtained by existentially quantifying the system’s internal variables in the conjunct of $\bigwedge_{i=1}^N (A_i \Rightarrow G_i)$ and the constraints resulting from the connectivity relation among the components. Thus we established that QE serves as a foundation for property/contract composition. Now

^a  <https://orcid.org/0000-0001-7101-6120>

^b  <https://orcid.org/0000-0003-3974-5790>

^c  <https://orcid.org/0000-0003-4396-9041>

to check whether a system satisfies a postulated property, we only need to check if the postulated property is implied by the aforementioned strongest system property. This in itself can be cast as a QE problem. Our QE based framework can derive the above strongest system property automatically as we show it later in the next section as part of review of our previous work.

This paper further extends our QE-based property composition to the case of time-dependent or temporal properties, which can depend on a (finite) history of input/output variables of a system and its components. We show that the composed property may involve a longer history, but no more than the cumulative histories of all its components. Accordingly, we introduce the notion of property order, derivation of system order, and the composition of given properties along with their time-shifted replicas to infer the strongest system property. We have implemented our QE-based compositional verification approach in a prototype tool, ReLIC (Reduced Logic Inference for Composition), based on the integration of Redlog (red, 2015) with AGREE (AGR, 2012; Gacek et al., 2015; Stewart et al., 2017)—the former supports QE, while the latter supports description of a system and its components, their connectivity, and properties in the modeling framework of AADL (Feiler et al., 2006). Our integration uses only the front-end of AGREE for specifying system architecture/connectivity, components, and their properties in AADL and AGREE annex, and also for reporting the result of property composition to the user.

2 QE-BASED COMPOSITIONAL VERIFICATION

Our QE-based compositional reasoning approach is based upon introducing the notion of the “strongest system property”, derived from the given component-level properties. Consider a system S composed of N components. Let $X := \{x_1, \dots, x_n\}$ be the set of all variables in S , $X_{\text{int}} := \{x_1, \dots, x_m\} \subseteq X (m \leq n)$, be the set of internal variables, $X_{\text{sys}} := X \setminus X_{\text{int}} = \{x_{m+1}, \dots, x_n\}$ be the set of external variables (namely, the inputs and outputs of S), and $C := \{(x_p, x_q) \mid x_p \text{ and } x_q \text{ are variables of connected ports in } S\}$ be the set of connectivity relation among component input/output variables. Suppose the i^{th} component’s property is described by an assume-guarantee style contract (A_i, G_i) in first-order logic, meaning $A_i \Rightarrow G_i$ holds. We define the *strongest system property* and present a result that provides a method to derive it. The proof can be found in (Ren et al., 2016).

Definition 2.1. The strongest system property is the system property that implies any other system properties established upon the given component properties and the connectivity relation.

Theorem 2.1. The strongest system property of a system S established upon its component contracts $\{(A_i, G_i) \mid 1 \leq i \leq N\}$, component connectivity relation $C = \{(x_p, x_q) \mid x_p \text{ and } x_q \text{ are connected ports in } S\}$, and internal variables $\{x_i, \dots, x_m\}$, is given by,

$$\exists x_1 \dots \exists x_m \left(\bigwedge_{i=1}^N (A_i \Rightarrow G_i) \wedge \bigwedge_{(x_p, x_q) \in C} (x_p = x_q) \right). \quad (1)$$

Remark. Through (1) in Theorem 2.1, we have shown that property composition, in a component-based compositional framework, is an instance of a QE problem. Based on this insight, we put forth a two-step QE-based compositional verification procedure. The first step is to generate the strongest system property, through a QE process of (1), applied to component contracts and connectivity relation. The strongest system property upon QE, denoted ϕ_{sys} , contains only the system-level input/output variables (as the internal variables get existentially quantified and eliminated upon QE). The second step is to check if ϕ_{sys} implies any postulated system property ϕ_{post} that also contains only system-level input/output variables. For this we can employ yet another instance of QE, this time over the external variables of S : $\forall x_{m+1} \dots \forall x_n (\phi_{\text{sys}} \Rightarrow \phi_{\text{post}})$ to reduce through QE either to *true* or *false*.

3 RELIC FOR TIME-DEPENDENT PROPERTY COMPOSITION

Complex systems often exhibit time-dependent features through components such as PID controller, delay, counter, or state-machine. In such cases, a component property can be a constraint over its input/internal/output variables at different time-steps. Let $x(k)$ denote the variable at the k^{th} time step with step 0 being the initial step, and for $s, t \in \mathbb{Z}_{\geq 0}, s \leq t$: $X[s, t] := \{x(k) \mid x \in X, k \in [s, t]\}$ be the variables over the time interval $[s, t]$.

Example 3.1. Consider a *cascade* of two identical components with input/output pairs (u, x) and (x, y) , and with properties $x(k) > u(k-1)$ and $y(k) > x(k-1)$ respectively, whereas the common variable x represents the cascade connection. Then one can see that composed system satisfies $y(k) > u(k-2)$. But a standard quantifier elimination as in (1) cannot be employed to obtain the above final formula since the term $u(k-2)$ is not even present

in the component properties. However, if we compose the component properties and their one-time-step shifted replicas with the internal variable $x(k)$, $x(k-1)$, and $x(k-2)$ existentially quantified: $\exists x(k)\exists x(k-1)\exists x(k-2)((x(k) > u(k-1)) \wedge (y(k) > x(k-1)) \wedge (x(k-1) > u(k-2)) \wedge (y(k-1) > x(k-2)))$, then upon quantifier elimination, we do get the desired composed property: $y(k) > u(k-2)$. To account for the number of time-shifts involved in the component properties versus the system properties, we introduce the notion of “order”.

Definition 3.1. We define component/system order as the difference of maximum and minimum time-shifts present in its governing difference equations/inequations.

As illustrated in above example, the component properties may need to be time-shifted to match the possibly higher order of the governing equations/inequations of the composed system. In order to decide the number of time-shifts needed, the order M_{sys} of the composed system must be estimated. This is presented in the following theorem, which states that an upper bound for the system order is the sum of all its component orders.

Theorem 3.1. Given a system of N multi-input-multi-output (MIMO) components, if the i^{th} component is of order M_i , then $\sum_{i=1}^N M_i$ is an upper bound for the system order M_{sys} .

Proof. Without loss of generality, pick any two components of S with orders M_1 and M_2 respectively. Each component possesses a set of properties specified as nonlinear difference equations/inequations in the general form of $f(\cdot) \sim 0$, where $\sim \in \{>, \geq, =\}$. For an inequation, we can introduce a slack variable $u_f \sim 0$ such that the given difference inequation can equivalently be written as the conjunction of a difference equation $f(\cdot) - u_f = 0$ and an extra linear constraint $u_f \sim 0$. Note this additional constraint is of zero order and hence can not alter the overall order.

For the set of difference equations of component i with N_i inputs (including slack variables) and O_i outputs, $i = 1, 2$, it is known that there exists an equivalent state-space representation (Fadali and Visioli, 2012), and also the number of the state variables equals the order of the component. The general form of such a state-space representation for component i is:

$$\begin{aligned} \mathbf{x}_i(k+1) &= \mathbf{f}_i(\mathbf{x}_i(k), \mathbf{u}_i(k)), \\ \mathbf{y}_i(k) &= \mathbf{g}_i(\mathbf{x}_i(k), \mathbf{u}_i(k)), \end{aligned}$$

where vectors \mathbf{u}_i (size: $N_i \times 1$), \mathbf{x}_i (size: $M_i \times 1$), and \mathbf{y}_i (size: $O_i \times 1$) are respectively the input, state, and

output variable vectors of component i . (The constraints on the added slack variables also exist, but as noted, do not involve time-shifts and so do not alter the component order.) $\mathbf{f}_i(\cdot)$ (size: $M_i \times 1$) and $\mathbf{g}_i(\cdot)$ (size: $O_i \times 1$) are vectors of functions.

One can simply stack the two state space representations into a single one:

$$\begin{aligned} \begin{bmatrix} \mathbf{x}_1(k+1) \\ \mathbf{x}_2(k+1) \end{bmatrix} &= \begin{bmatrix} \mathbf{f}_1(\mathbf{x}_1(k), \mathbf{u}_1(k)) \\ \mathbf{f}_2(\mathbf{x}_2(k), \mathbf{u}_2(k)) \end{bmatrix}, \\ \begin{bmatrix} \mathbf{y}_1(k) \\ \mathbf{y}_2(k) \end{bmatrix} &= \begin{bmatrix} \mathbf{g}_1(\mathbf{x}_1(k), \mathbf{u}_1(k)) \\ \mathbf{g}_2(\mathbf{x}_2(k), \mathbf{u}_2(k)) \end{bmatrix}. \end{aligned}$$

It is then clear that the number of states of these, so far unconnected components, equals the sum of the numbers of states of the two individual components. We claim that when the components are connected, the state size does not grow, from which the desired result stated in the theorem can be obtained. A general connectivity relation between two components can be formulated as $\mathbf{u}_c = \mathbf{y}_c$ where \mathbf{u}_c is a vector of inputs from the union of the two component inputs, and \mathbf{y}_c is a vector of outputs from the union of two component outputs. We note that one input can only connect to one output whereas one output may connect to multiple inputs. Since the overall connected system can be obtained by iteratively adding one connection at a time, it suffices to show that adding a single connection does not introduce an additional state variable. For compactness of notation, let the state-space representation of the combined system be:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)), \\ \mathbf{y}(k) &= \mathbf{g}(\mathbf{x}(k), \mathbf{u}(k)), \end{aligned}$$

where the state set has $M_1 + M_2$ variables. Suppose a single connection (u_p, y_q) is introduced within the system, where $u_p \in \mathbf{u}$, $y_q \in \mathbf{y}$, and u_p, y_q are from different components. Let \mathbf{u}' (resp. \mathbf{y}') be the vector after removing u_p (resp. y_q) from \mathbf{u} (resp. \mathbf{y}). Then, we have $u_p(k) = y_q(k) = g_q(\mathbf{x}(k), \mathbf{u}(k)) = g_q(\mathbf{x}(k), \mathbf{u}'(k))$ with $g_q \in \mathbf{g}$, and so the state-space representation can be written as:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{f}(\mathbf{x}(k), \mathbf{u}'(k), g_q(\mathbf{x}(k), \mathbf{u}'(k))), \\ \mathbf{y}(k) &= \mathbf{g}(\mathbf{x}(k), \mathbf{u}'(k), g_q(\mathbf{x}(k), \mathbf{u}'(k))). \end{aligned}$$

Then it is easily seen that the updated state-space representation possesses input variables from \mathbf{u}' (u_p becomes internal), output variables from \mathbf{y} (or $\mathbf{y} \setminus \{y_q\}$ if y_q also becomes internal), while the state-space remains \mathbf{x} . (Further simplification and a state-space reduction may be possible.) It follows that the system order of the connected system remains upper bounded by $M_1 + M_2$. The result of the theorem then follows by

iteratively connecting more components to the composition. \square

Note any component may be described by a *set* of properties over its own input, internal, and output variables, and thus can be viewed as a set of sub-components, each associated with one of those properties, and with their connectivity relation implied by their common variables. Hence we have the following corollary:

Corollary 3.1.1. The order of a component is bounded by the sum of all its property orders. The system order of S in Theorem 3.1 is bounded by the sum of all the individual property orders of all its components.

After the composed system order bound M_{sys} is determined, the i^{th} individual property of the components can be replicated $M_{\text{sys}}-M_i$ times, where M_i is the order of the i^{th} individual property (so it is defined over variables $X[k, k+M_i]$), and the j^{th} replica shifted j time-steps ($j = 1, \dots, M_{\text{sys}}-M_i$) to obtain the set of all constraints over $X[k, k+M_i] \cup_{1 \leq j \leq M_{\text{sys}}-M_i} X[k+j, k+j+M_i] = X[k, k+M_{\text{sys}}]$.

Theorem 3.2. The strongest system property of a time-dependent system (time-dependent version of S) can be obtained from the replicated and time-shifted components properties by:

$$\exists X_{\text{int}}[k, k+M_{\text{sys}}] \left(\bigwedge \text{component properties} \right. \\ \left. \& \text{ shifted replicas over } X[k, k+M_{\text{sys}}] \right), \quad (2)$$

where the existential quantification is with respect to the set of internal variables $X_{\text{int}}[k, k+M_{\text{sys}}] \subseteq X[k, k+M_{\text{sys}}]$ of the system.

Remark. Theorem 3.1 provides an upper bound to the system order, while the actual system order M_{sys} may be less than the upper bound $\sum_{i=1}^N M_i$. In the case, the result of the QE in (2) will contain the conjunction of $\sum_{i=1}^N M_i - M_{\text{sys}}$ redundant expressions that are time-shifted replicas of some other expressions within the same conjunct. Such replicas can be dropped, without altering the resultant property, reducing the order to equal M_{sys} .

Example 3.2. Consider three properties $z = y + x$, $y = \mathbf{pre}(u)$, and $x = \mathbf{pre}(w)$, where x and y are internal variables, and the operation \mathbf{pre} denotes the value at the previous step. Then, from Theorem 3.1, the system order is upper bounded by the sum of the three subsystem orders: $0 + 1 + 1 = 2$. On the other hand, it is easy to see that the composed property over the external variables z , u , and w is $z = y + x = \mathbf{pre}(u) + \mathbf{pre}(w)$, which is only of order 1. The computation of the composed property using our approach

requires shifting each components by its order difference with the upper bound (so, 2, 1, 1 resp.), and then combining those using (2):

$$\exists x(k) \exists x(k+1) \exists x(k+2) \exists y(k) \exists y(k+1) \exists y(k+2) \\ \left((z(k) = y(k) + x(k)) \wedge (z(k+1) = y(k+1) + x(k+1)) \right. \\ \left. \wedge (z(k+2) = y(k+2) + x(k+2)) \right) \wedge (y(k+1) = u(k)) \\ \wedge (y(k+2) = u(k+1)) \wedge (x(k+1) = w(k)) \wedge (x(k+2) = w(k+1)).$$

Upon the elimination of the quantified variables, we can obtain:

$$(z(k+1) = u(k) + w(k)) \wedge (z(k+2) = u(k+1) + w(k+1)),$$

in which $z(k+2) = u(k+1) + w(k+1)$ is a one-step shifted replica of $z(k+1) = u(k) + w(k)$, and hence redundant, and so it can be dropped without altering the derived system property. By expressing the QE result in conjunctive normal form (CNF), the redundant expressions can be readily identified as the time-shifted replicas of some other expressions, and eliminated.

Similarly, we can also derive the system initial condition by composing the ones of the components. Then a postulated system property can be verified using an induction-based proof (Sheeran et al., 2000; Kahsai and Tinelli, 2011). For a system of order M_{sys} , initial conditions over the first M_{sys} steps ($0, \dots, M_{\text{sys}}-1$) suffice. So the system-level initial condition can be obtained using:

$$\exists X_{\text{int}}([0, M_{\text{sys}}-1]) \left(\bigwedge \text{component properties} \right. \\ \left. \& \text{ shifted replicas over } X([0, M_{\text{sys}}-1]) \& \text{ ICs} \right), \quad (3)$$

where ICs denotes the initial conditions.

Example 3.3. Consider the same example given at the beginning of this section with the component properties and initial conditions: $y > \mathbf{pre}(x)$, $y(0) > 0$ and $x > \mathbf{pre}(u)$, $u(0) > 1$. Then in this case, (3) is encoded as: $\exists y([0, 1]) ((z(0) > 0) \wedge (z(1) > y(0)) \wedge (y(0) > 1) \wedge (y(1) > x(0)))$, which is equivalent to $(z(0) > 0) \wedge (z(1) > 1)$ by eliminating the existentially quantified internal variable y at the time-steps 0 and 1, namely, $y(0)$ and $y(1)$.

4 IMPLEMENTATION AND EXPERIMENTAL RESULT

We implemented the QE-based the time-independent and time-dependent property composition in a tool, which we refer as ReLIC (Reduced Logic Inferencing for Composition). The architecture of our prototype

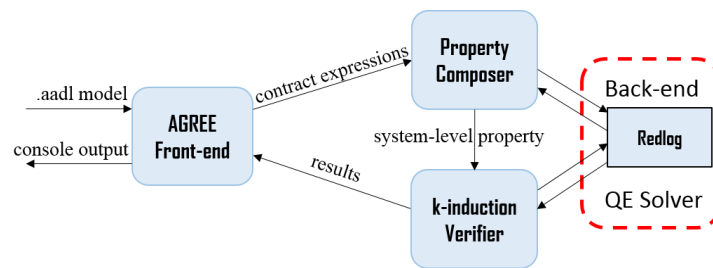
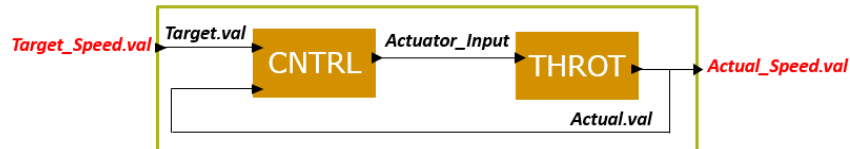


Figure 1: Architecture of ReLIC.



(a) A vehicle model and its components dynamics.

```

system Speed_Control
  features
    Target: in data port Types::speed.speed_impl;
    Actuator_Input: out data port Base_Types::Float;
    Actual: in data port Types::speed.speed_impl;

  annex agree {**
    const P : real = 0.2;
    const D : real = 0.1;
    const I : real = 0.1;

    eq e : real = Target.val - Actual.val;
    eq e_int : real = prev(e, 0.0) + e;
    eq e_dot : real = prev(e, 0.0) - e;

    eq u : real = P*e + D*e_dot + I*e_int;
    guarantee "Acuator_Behavior" : Actuator_Input = u;
  **};
end Speed_Control;
  
```

(b) Specification of CNTRL component.

```

system Throttle
  features
    --Actuator_Input: in data port Types::actuator.actuator_impl;
    Actuator_Input: in data port Base_Types::Float;
    Actual: out data port Types::speed.speed_impl;

  annex agree {**
    guarantee "Throttle_Behavior" : Actual.val = prev(Actual.val, 0.0)
      + 0.1*Actuator_Input;
  **};
end Throttle;
  
```

(c) Specification of THROT component.

Figure 2: A vehicle model, modified from (Gacek et al., 2017).

tool ReLIC is shown in Fig. 1. It utilizes the front-end infrastructure of the AGREE (Stewart et al., 2017) tool for capturing and parsing the input AADL models that describe the components, their connectivity, and their properties, and also its output environment for presenting the property composition and verification results to the console. The “Property Composer” module computes the upper bound for the system-level or-

der based on Theorem 3.1 and computes the strongest system-level property based on Theorem 3.2. This is then used by the “Induction Verifier” module to perform induction-based system-level verification of a postulated system property. Both these modules utilize Redlog (red, 2015) as a back-end solver that supports quantifier elimination.

To demonstrate our tool, we tested it on a vehi-


```

property const_tar_speed =
  Target_Speed.val = prev(Target_Speed.val, 1.0) and prev(const_tar_speed, true);
assume "constant target speed" : const_tar_speed;
guarantee "actual speed is less than constant target speed" :
  (Actual_Speed.val < 1.0);

```

Property	Result
✓ System Contract	1 Valid
✓ actual speed is less than constant target speed	Valid (1s)

```

Strongest System Property
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// The infix operator precedence is from strongest to weakest: and, or, impl, rep
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

The initial constraint:
51*actual_speed - target_speed = 0

The general time-dependent property:
51*actual_speed - 49*pre__actual_speed - pre__target_speed - target_speed = 0

```

Figure 3: ReLIC verification output on the vehicle model.

cle model as shown in Fig. 2a. The vehicle consists of two components, namely, a PID control component “Speed_Control” (CNTRL) and a vehicle throttle “Throttle” (THROT), within a feedback configuration. The dynamics of “Speed_Control” is specified by a set of difference equations involving also certain state variables (Fig. 2b), whereas the dynamics of “Throttle” is specified by a difference equation over only its input and output variables (Fig. 2c).

There are three first-order expressions in total, namely,

$$e_{int} = \mathbf{prev}(e, 0.0) + e,$$

$$e_{dot} = \mathbf{prev}(e, 0.0) - e, \text{ and}$$

$$Actual.val = \mathbf{prev}(Actual.val, 0.0) + 0.1 * Actuator_Input,$$

where $\mathbf{prev}(\cdot)$ is the extended delay operator $\mathbf{pre}(\cdot)$ with the second argument denoting the initial value. The postulated system property is

$$const_tar_speed \Rightarrow (Actual_Speed.val < 1.0),$$

where $const_tar_speed$ is a Boolean variable whose truth indicates system target speed is set to a constant value 1.0 (its defining expression is given in Fig. 3). The ReLIC inferred strongest system property (based on (2) and (3)) contains the system-level difference equation as well as the initial condition over system input $Target_Speed.val$ and system output $Actual_Speed.val$. While the upper bound for the composed system order is 3, it turns out that the system input-output property order is only 1 due to the inherent parallelism among the components, as also computed by our approach (see the encoded expressions below as well as the console output in Fig. 3):

$$51 * actual_speed - 49 * pre_actual_speed - pre_target_speed - target_speed = 0,$$

with the initial condition:

$$51 * actual_speed - target_speed = 0.$$

These are easily shown to imply the postulated system property using induction, where the base step is:

$$(51 * actual_speed - target_speed = 0) \Rightarrow ((target_speed = 1) \Rightarrow (actual_speed < 1)),$$

and the inductive step is

$$((51 * actual_speed - 49 * pre_actual_speed - pre_target_speed - target_speed = 0) \wedge (pre_target_speed = 1) \wedge (pre_actual_speed < 1)) \Rightarrow ((target_speed = pre_target_speed) \Rightarrow (actual_speed < 1)).$$

Both the base and induction steps can be verified to be true, again using quantifier elimination, this based on universal quantification over the external variables.

5 CONCLUSION

We introduced for a first-time a property composition framework to support the composition of time-dependent properties. This is a foundational step towards compositional reasoning of systems possessing bounded “memory”. The extension developed a new procedure to determine the system order, given the component orders, and used it to form the required time-shifted replicas of the component properties and their QE-based composition. We implemented our compositional approach in our prototype tool ReLIC that uses AADL for modeling system architecture and component properties using the tool AGREE, and the QE solver Redlog at the back-end for performing the QE. The proof of a postulated system-level property involves induction, which is also supported in our ReLIC tool.

ACKNOWLEDGEMENTS

This work was partially supported by the National Science Foundation under the grants NSF-CCF-1331390, NSF-ECCS-1509420, NSF-IIP-1602089, and a sabbatical support from the Air Force Research Lab, Dayton, OH.

REFERENCES

- (2012). Agree. Available from: <http://loonwerks.com/tools/agree.html>.
- (2015). redlog. Available from: <http://www.redlog.eu/>.
- Emmi, M., Giannakopoulou, D., and Păsăreanu, C. S. (2008). Assume-guarantee verification for interface automata. In *International Symposium on Formal Methods*, pages 116–131. Springer.
- Fadali, M. S. and Visioli, A. (2012). *Digital control engineering: analysis and design*, chapter 7. State-Space Representation. Academic Press.
- Feiler, P. H., Gluch, D. P., and Hudak, J. J. (2006). The architecture analysis & design language (aadl): An introduction. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.
- Gacek, A., Backes, J., Mike, W., Darren, C., and Jing, L. (2017). Agree users guide, v0.9. Available from: <https://github.com/smaccm/smaccm/tree/master/documentation/agree>.
- Gacek, A., Katis, A., Whalen, M. W., Backes, J., and Cofer, D. (2015). Towards realizability checking of contracts using theories. In *NASA Formal Methods Symposium*, pages 173–187. Springer.
- Kahsai, T. and Tinelli, C. (2011). Pkind: A parallel k-induction based model checker. *arXiv preprint arXiv:1111.0372*.
- Kingston, D., Rasmussen, S., and Humphrey, L. (2016). Automated uav tasks for search and surveillance. In *Control Applications (CCA), 2016 IEEE Conference on*, pages 1–8. IEEE.
- Quinton, S. and Graf, S. (2008). Contract-based verification of hierarchical systems of components. In *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 377–381. IEEE.
- Rasmussen, S., Kalyanam, K., and Kingston, D. (2016). Field experiment of a fully autonomous multiple uav/ugs intruder detection and monitoring system. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 1293–1302. IEEE.
- Ren, H., Clark, M., and Kumar, R. (2018). Integration of quantifier eliminator with model checker and compositional reasoner. In *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, pages 1070–1075. IEEE.
- Ren, H., Clark, M. A., and Kumar, R. (2016). Integration of quantifier eliminator with model checker and compositional reasoner. In *Air Force Research Laboratory's Safe & Secure Systems and Software Symposium*.

- Sheeran, M., Singh, S., and Stålmarck, G. (2000). Checking safety properties using induction and a sat-solver. In *International conference on formal methods in computer-aided design*, pages 127–144. Springer.
- Stewart, D., Whalen, M. W., Cofer, D., and Heimdahl, M. P. (2017). Architectural modeling and analysis for safety engineering. In *International Symposium on Model-Based Safety and Assessment*, pages 97–111. Springer.
- Tarski, A. (1998). A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24–84. Springer.