Finite Bisimulation of Reactive Untimed Infinite State Systems Modeled as Automata with Variables

Changyan Zhou, and Ratnesh Kumar, Fellow, IEEE

Abstract—Some discrete event systems such as software are typically infinite state systems, and a commonly used technique for performing formal analysis such as automated verification is based on their finite abstractions. In this paper, we consider a model for reactive untimed infinite state systems called input-output extended finite automaton (I/O-EFA), which is an automaton extended with discrete variables such as inputs, outputs, and data. Using I/O-EFA as a model many value-passing processes can be represented by finite graphs. We study the problem of finding a finite abstraction that is bisimilar to a given I/O-EFA. We present a sufficient condition under which the underlying transition system of an I/O-EFA admits a finite bisimilar quotient. We then identify a class of I/O-EFAs for which a partition satisfying our sufficient condition can be constructed by inspecting the structure of the given I/O-EFA. We also identify a lower bound abstraction (that is coarser than any finite bisimilar abstraction), and present an iterative refinement algorithm whose termination guarantees the existence of a finite bisimilar abstraction. The results are illustrated through examples that model reactive software.

Note to Practitioners: The decidability of any analysis of a reactive software (such as the correctness analysis) is guaranteed when the software possesses a finite-state equivalent model. A contribution of the paper is that it identifies a class of reactive software possessing such a property. Also, for a reactive software outside the identified class, the paper presents an abstraction algorithm that terminates with a finite-state equivalent model if and only if one exists. Examples are presented to illustrate the results reported in the paper.

Keywords: Extended automata, symbolic transition systems, formal verification, bisimulation equivalence, software modeling, software abstraction, software verification.

I. INTRODUCTION

In recent years there has been extensive research on symbolic modeling and automated verification of infinite state systems. Examples of symbolic models for untimed discrete event systems include symbolic transition graph (STG) [12], extended finite state machines (EFSMs) [10], and unbounded Petri Nets [24]. These models are extensions of automata through incorporation of variables and possess an underlying infinite transition system such as those categorized in [14]. Untimed infinite state systems deserve a separate attention since a considerable class of software systems can be viewed as consisting of a finite control component and infinite (integer-valued) data component. Such systems evolve data values from a potentially infinite domain and as a result possess an infinite number of different states.

Automated verification methods such as model-checking have been invented for the analysis of finite state systems. An important technique for verifying an infinite state system is its reduction to an equivalent finite state system through abstraction [19], [15], [2]. Another approach is to directly analyze an infinite state system by symbolically encoding the states and the transitions as formulas of a suitable logic, see for example [9], [4], [11], [6], [19], [5] employed predicate abstraction technique for extracting finite state models from infinite state systems. Given a concrete infinite state system and a set of abstraction predicates, a conservative finite state abstraction is generated. An abstraction is exact when the abstracted system is bisimulation equivalent to the original system. Approaches to obtain exact finite abstractions have been pursued for timed systems [2], [8], for linear and nonlinear systems [23], [22], and for hybrid systems [3], [16], [7].

In this paper, we consider a model for reactive untimed infinite state systems, called input-output extended finite automaton (I/O-EFA), which is an automaton extended with discrete variables such as inputs, outputs, and data (see for example [10] for automata extended with data variables and [20] for automata extended with input/output variables), and study the problem of finding their exact finite abstractions. In order to be able to apply existing finite state system verification methods to a system modeled as an I/O-EFA, it is desirable that its underlying transition system possesses a finite bisimilar abstraction (also called quotient). Although I/O-EFA is a special type of a hybrid automaton with no continuous dynamics (i.e., flow rate of each variable is zero), it requires a separate attention since (i) it models a large class of systems such as embedded software (the modeling of a class of reactive software as I/O-EFAs is presented in Appendix A), and (ii) owing to the non-existence of continuous dynamics in I/O-EFAs, restrictions required to ensure bisimilarity with a finite state system can be more relaxed when compared to those required for a hybrid automaton.

For I/O-EFAs, besides the usual notion of bisimilarity, one can define a stronger notion, namely that of “late”-bisimilarity [12], [18]. (In [12], [18], the term “early-bisimilarity” is used for what one would define to be bisimilarity for I/O-EFAs; we avoid using “early-bisimilarity” as that can cause confusion.) According to the usual notion of bisimilarity, a system can use its knowledge about the current input in choosing a transition...
to bisimulate a transition of another system, whereas in the "late" setting the input is read only after choosing a transition for bisimulating a transition of another system.

In general, the problem of finding a finite quotient that is bisimilar to an I/O-EFA is undecidable. [14] studies a class of systems called, symbolic transition systems (STSs), and reports a semi-algorithm to compute a finite-index bisimulation relation by recursively performing a partition refinement. The semi-algorithm terminates if and only if the STS possesses a finite bisimilar quotient. Note [14] does not identify any subclass of STSs for which the semi-algorithm is guaranteed to terminate (so that an STS in the subclass is guaranteed to be bisimilar to a finite-state feature). In contrast we identify a subclass of I/O-EFAs with the property that any I/O-EFA in the subclass is bisimilar to a finite-state system. Another difference between our work and that reported in [14] is that in [14] there is no notion of initial states, and any state is treated as an initial state and so all states are reachable. We do have a notion of initial states, and so not all states may be reachable. Then the unreachable states do not have to satisfy any condition for the existence of a finite bisimilar quotient. This situation does not arise in [14] where the set of unreachable states is empty.

We present a sufficient condition under which an I/O-EFA admits a finite bisimilar quotient. The sufficient condition we present is for the existence of a finite late-bisimilar quotient, and by virtue of late-bisimilarity being stronger than bisimilarity, it also serves as a sufficient condition for the existence of a finite bisimilar quotient. The condition we present is existential as it relies on the existence of a suitable partition of the state space. Next we identify a class of I/O-EFAs for which a partition satisfying our sufficient condition can be constructed by inspecting the structure of the given I/O-EFA. The identified class of I/O-EFAs has the property that any I/O-EFA in this class is bisimilar to a finite-state system and hence decidable (this result was first presented at a conference [17]). We also identify a lower bound abstraction (that is coarser than any finite bisimilar abstraction), and present an iterative refinement algorithm whose termination guarantees the existence of a finite bisimilar abstraction (this result was first presented at a conference [25]). The results are illustrated through examples that model reactive software.

There exist other works related to the topic of the paper. [1] defines a class of infinite state systems, called well-structured systems, for which a finite simulation quotient exists. The problem of bisimulation-checking between infinite-state systems modeled as STGAs has been studied in [12], [18], [18] computes a set of predicate equations whose largest solution produces the condition under which the two STGAs are bisimilar. However, such largest solutions are not automatically computable in general. [21], [13] propose proof systems for model-checking value passing processes, which again are not decidable in general. The main goal of these works is to develop semi-algorithmic approaches for bisimulation and/or model checking for infinite-state systems and not to identify a restricted subclass for which their algorithms are guaranteed to terminate. In contrast we present a verifiable sufficient condition under which an I/O-EFA with an infinite state space possesses a finite bisimilar quotient.

II. INPUT/OUTPUT EXTENDED FINITE AUTOMATA

An input/output extended finite automaton or state machine (I/O-EFA or I/O-EFSM) is a symbolic description of reactive untimed infinite state systems in form of an automaton extended with discrete variables such as inputs, outputs, and data. Using I/O-EFA as a model many value-passing processes can be represented by finite graphs. An I/O-EFA consists of locations (L), data (D), inputs (U), outputs (Y), transition labels (Σ∪{ε}), transitions (E), initial locations (L₀) and data values (D₀). Locations together with data form the state-space of I/O-EFAs. Locations are finite and form the vertices of the automaton graph. Edges of the graph represent transitions between locations and are guarded by constraints over data and inputs. Occurrence of a transition triggers a data update and an output assignment. An I/O-EFA is formally defined as follows.

Definition 1: An input/output extended finite automaton (I/O-EFA) is an eight-tuple \( P = (L, D, U, Y, \Sigma, E, L₀, D₀) \), where

- \( L \) is the set of locations,
- \( D = D₁ \times \ldots \times Dₚ \) is the set of p-dimensional data,
- \( U = U₁ \times \ldots \times Uₚ \) is the set of q-dimensional input,
- \( Y = Y₁ \times \ldots \times Yᵦ \) is the set of r-dimensional output,
- \( \Sigma \cup \{ε\} \) is the set of transition labels,
- \( E \) is the set of edges, and each \( e ∈ E \) is a 6-tuple, \( e = (oₑ, tₑ, \sigmaₑ, Gₑ, fₑ, hₑ) \), where
  - \( oₑ ∈ L \) is the origin location,
  - \( tₑ ∈ L \) is the terminal location,
  - \( \sigmaₑ ∈ Σ \cup \{ε\} \) is the transition label,
  - \( Gₑ ⊆ D × U \) is the enabling guard,
  - \( fₑ : D × U → D \) is the data update function,
  - \( hₑ : D × U → Y \) is the output assignment function,
- \( L₀ \) is the set of initial location, and
- \( D₀ = D₁₀ \times \ldots \times Dₚ₀ \) is the set of initial data values.

All variables range over countable sets and can be taken to be the set of integers or real numbers with a finite precision. We use \( ⃗u \), \( ⃗y \), and \( ⃗d \) to denote an input, an output, and a data respectively. We use \( d(i) \) to denote the ith component of \( ⃗d \), i.e., \( d(i) = \{d(1), ..., d(p)\} \), where \( p \) is the number of data variables. \( \hat{d}_i = \hat{d}_j \) means component-wise equality, i.e., \((d_i(i)) = d_j(i), \forall i ∈ \{1, ..., p\}\). We use \( fₑ(i)(⃗d, ⃗u) \) (resp., \( hₑ(i)(⃗d, ⃗u) \)) to represent the update function of the ith data (resp., assignment function of the jth output). The edge enabling guard \( Gₑ(D, U) \) is a predicate over data and inputs. Sometimes we write \("(⃗d, ⃗u) ∈ Gₑ(D, U)"\) also as \"\( Gₑ(⃗d, ⃗u) \"\). We define \( Gₑ(D, \{⃗u\}) := \{d ∈ D | (⃗d, ⃗u) ∈ Gₑ(D, D)\} \) to be the predicate over data such that for any data satisfying this predicate the edge \( e \) is enabled on input \( ⃗u \in U \). Functions \( fₑ \) and \( hₑ \) can naturally be extended to be defined over sets:

\[
\forall \hat{D} ⊆ D, \hat{U} ⊆ U : fₑ(\hat{D}, \hat{U}) := \cup_{d, u ∈ \hat{D}, \hat{U}} \{fₑ(d, u)\};
\]

\[
hₑ(\hat{D}, \hat{U}) := \cup_{d, u ∈ \hat{D}, \hat{U}} \{hₑ(d, u)\}.
\]

The semantics of an input-output automaton \( P \) can be understood as follows. Initially, \( P \) starts from the initial location \( l₀ \) and an initial data value \( d₀ \in D₀ \). While at a certain state \((ℓ, ⃗d) ∈ L × D\), a transition \( e ∈ E \) such that \( oₑ = ℓ \) is enabled if the input \( σₑ \) arrives, and the data \( ⃗d \) and input \( ⃗u \) are such that the guard \( Gₑ(⃗d, ⃗u) \) holds. Note when \( σₑ = ε \), the
transition is enabled when only the guard \( G_e(d, u) \) holds; on the other hand when \( G_e(D, U) = \text{True} \), then the transition is enabled when only \( \sigma_e \) arrives. An enabled transition can be executed. The execution of an enabled transition \( e \) at the state \((o_e, d)\) causes \( P \) to transit to the location \( t_e \), the data value is updated to \( f_e(d, u) \), the output variable is assigned the value \( h_e(d, u) \), and a discrete output \( \delta_e \) is emitted.

Note that there is no requirement that data update and output assignment occur in the same transition (i.e., one or both functions can be identity), and so the model is powerful enough to capture both synchronous and asynchronous systems. The following example illustrates the concept of an I/O-EFA.

**Example 1:** Consider the following program:

```c
int main() {
    int d1 = 0, d2 = 0;
    while(1) {
        A: if (d1 < 0) {d1 = d1 + 1; d2 = d1 + 2; printf("A\n");} else break;
        B: if (2*d1 + d2 > 2) {d2 = 0; printf("B\n");} else break;
        C: int temp; scanf("%d", &temp); if (d1 >= temp) {d1 = d1 + 1; d2 = d1 + 1; printf("C\n");} else break;
    }
    return 0;
}
```

An I/O-EFA model \( P \) of the above program (translation of a class of reactive software to I/O-EFA models is presented in Appendix A.) is shown in Figure 1, where \( \xi_1 : A \rightarrow B, \xi_2 : B \rightarrow C, \text{and } \xi_3 : C \rightarrow A \). Let \( L = \{A, B, C\}, U = \{u\}, Y = \{y\}, L_0 = \{A\}, \text{and } d = (d(1), d(2)). \) Let \( D_0 = \{(0, 0)\}. \) At the initial location \( A \), since the guard \( d(1) \leq 0 \) holds, the system can transit to location \( B \). Such a transition updates the data to \((d(1) + 1, d(1) + 2) = (1, 2)\), and assigns the output \( y := \xi_1 \). Next since \( 2d(1) + d(2) = 4 \geq 2 \), the transition from location \( B \) to \( C \) is enabled, and when executed updates the data to \((d(1) + 1, d(1) + 1) = (2, 2)\) and assigns the output \( y := \xi_2 \). Once upon the reception of the input \( u \) := temp, since \( 1 = d(1) \geq \left\lceil \frac{\text{temp}}{2} \right\rceil = 1 \), the transition between \( C \) and \( A \) is enabled, and upon execution updates the data to \((d(1) + 1, d(1) + 1) = (2, 2)\) and assigns the output \( y := \xi_3 \). Figure 1 shows a part of the reachability graph of \( P \), which can be seen to be infinite.

In this paper, sets and predicates are used interchangeably. The same notation is used to denote a set as well as the corresponding predicate. Whether a notation denotes a set or predicate is clear from the context in which it is used.

### III. BISIMULATION AND TRANSITION-/QUOTIENT-SYSTEMS

The section defines bisimilarity and late-bisimilarity for I/O-EFAs, the underlying transition system of an I/O-EFA, and a quotient system of a transition system. We assume, without loss of generality, that none of the transitions are labeled by \( \epsilon \). In case transitions with \( \epsilon \)-labels exist, we can obtain a (“weakly”) bisimulation equivalent I/O-EFA possessing no \( \epsilon \)-labeled transitions by replacing a sequence of transitions in which all but one transition is labeled by \( \sigma \neq \epsilon \) by a single “summary” transition labeled by \( \sigma \). A summary transition is defined as follows. For a set \( S \), the set of sequences of length-\( n \) (\( n \geq 1 \)) of elements of \( S \) is denoted by \( S^n \).

**Definition 2:** Given a sequence of edges \( \xi \in E^n \) of length \( n \geq 1 \), its summary edge \( [\xi] = (o_\xi, t_\xi, s_\xi, G_\xi, f_\xi, h_\xi) \in L \times L \times S^n \times 2^D \times U^n \times D \times Y \times U^n \) is inductively defined as follows. For all \( e \in E, \xi \in E^n, d \in D, u \in U^n, \tilde{u} \in U \):

\[
[e] := e \\
[\xi e] := \begin{cases} 
(o_\xi, t_\xi, s_\xi \sigma_e, [G_\xi(d, u) \land G_\xi(f_\xi(d, u), \tilde{u})], \\
f_\xi(f_\xi(d, u), \tilde{u}), h_\xi(f_\xi(d, u), \tilde{u})) & \text{if } t_\xi = o_\xi; \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

In order to define bisimilarity over states of an I/O-EFA \( P \), we introduce the following notations.

\[
\forall l, l' \in L, \tilde{d}, \tilde{d}' \in D, e \in E, \sigma \in \Sigma, \tilde{u} \in U, \tilde{y} \in Y: \\
[l \xrightarrow{e} l'] \iff [o_e = l] \land [t_e = l'],
\]

\[
[(l, \tilde{d}) \xrightarrow{\sigma, \tilde{y}} (l', \tilde{d}')] \iff \exists e = (l, l', \sigma, G, f, h) \in E \\
\iff G(d, \tilde{u}), \tilde{d} = f(d, \tilde{u}), \tilde{y} = h(d, \tilde{u}).
\]

**Definition 3:** Given an I/O-EFA \( P \), a simulation relation over its states is a binary relation \( \Phi \subseteq (L \times D) \times (L \times D) \) such that \( ((l_1, d_1), (l_2, d_2)) \in \Phi \) implies \( \forall e_1, \forall \tilde{u}_1, \exists e_2 : \\
\sigma_{e_2} = \sigma_{e_1}, := \sigma, \text{ and } [(l_1, d_1) \xrightarrow{\sigma, \tilde{y}_1} (l_1', d_1'), l_1 \xrightarrow{e_1} l_1'] \Rightarrow \\
\exists (l_2, d_2) \xrightarrow{\sigma, \tilde{y}_2}, (l_2', d_2'), l_2 \xrightarrow{e_2} l_2' \text{ s.t. } ((l_1', d_1'), (l_2', d_2')) \in \Phi. 
\]

On the other hand, a late-simulation relation over states of \( P \) is a binary relation \( \Phi \subseteq (L \times D) \times (L \times D) \) such that \( ((l_1, d_1), (l_2, d_2)) \in \Phi \) implies \( \forall e_1, \exists e_2 : \\
\sigma_{e_2} = \sigma_{e_1}, := \sigma, \text{ and } \forall \tilde{u}_1, [(l_1, d_1) \xrightarrow{\sigma, \tilde{y}_1} (l_1', d_1'), l_1 \xrightarrow{e_1} l_1'] \Rightarrow \\
\exists (l_2, d_2) \xrightarrow{\sigma, \tilde{y}_2}, (l_2', d_2'), l_2 \xrightarrow{e_2} l_2' \text{ s.t. } ((l_1', d_1'), (l_2', d_2')) \in \Phi. 
\]

A symmetrical simulation (resp., late-simulation) relation is called bisimulation (resp., late-bisimulation) relation. Two states \( (l_1, d_1), (l_2, d_2) \in L \times D \) are bisimilar (resp., late-bisimilar), denoted \( (l_1, d_1) \simeq (l_2, d_2) \) (resp., \( (l_1, d_1) \simeq_l (l_2, d_2) \)), if there exists a bisimulation (resp., late-bisimulation) relation \( \Phi \) such that for each \( (l_1, d_1, \tilde{d}_{10}) \in L_{10} \times D_{10} \) there exists \( (l_2, d_2, \tilde{d}_{20}) \in L_{20} \times D_{20} \) such that \( ((l_1, d_1, \tilde{d}_{10}), (l_2, d_2, \tilde{d}_{20})) \in \Phi \).

**Remark 1:** The notion of late-bisimulation is strictly stronger than bisimulation [12], [18]. It follows that if a system possesses a finite late-bisimilar quotient (refer to Definition 5 for the definition of a quotient system), it also possesses a finite bisimilar quotient. For this reason we concentrate mainly on the late-bisimulation relation.

The definition above defines the bisimilarity of a pair of states of the same system. This is general enough to define the bisimilarity of a pair of states of a pair of systems (we just consider the “union” system).

The next two definitions define the underlying transition and quotient systems.

**Definition 4:** Given an I/O-EFA \( P = (L, D, U, Y, \Sigma, E, L_0, D_0) \), its underlying transition system \( P \) is a 6-tuple \( P = (S, U, Y, \Sigma, E, S_0) \), where \( S := L \times D \) is the set of its states,
The reachability set of the underlying transition system is symmetric, and so it suffices to show that it is a simulation property, it should hold that for any input the data update (resp., output assignment) along any enabled edge maps an unbounded region to either another unbounded region or to some fixed data value (resp., be identical over an entire unbounded region). This is stated and proved in the following theorem. (With an abuse of notation, in the following we use π for an element of both data and state partitions and that which one it represents should be clear from context.)

**Theorem 1:** Given an I/O-EFA P, it admits a finite late-bisimilar quotient if the data space D can be partitioned into a finite number of regions, π₁, ..., πₙ ⊆ D (where π₁, ..., πₙ are unbounded, and πₙ₊₁, ..., πₙ are bounded) such that ∀i ≤ m, ∀e, ∀u : 

\[ G_e(D, \{u\}) \cap \pi_i \neq \emptyset \]

1. \( \pi_i \subseteq G_e(D, \{u\}) \), and 
2. \( (\exists \bar{d} \leq m : f_e(\pi_i, \bar{d}) \subseteq \pi_j) \lor (\forall \bar{d} \in \pi_i, f_e(\pi_i, \bar{d}) = \{f_e(\bar{d}, u)\}) \), and 
3. \( \forall \bar{d} \in \pi_i, h_e(\pi_i, u) = h_e(\bar{d}, u) \).

**Proof:** We define a finite partition of the state space \( L \times D \) of P based on the given partition of the data space, and show that each member of the partition is a late-bisimulation equivalence class. Consider, \( \Pi := \Pi_1 \cup \Pi_2 \); \( \Pi_1 := \{\{l\} \times \pi_i \mid l \in L, i \leq m\} \); \( \Pi_2 := \{(l, \bar{d}) \mid l \in L, \bar{d} \in \pi_i, m < i \leq n\} \). We will show that each \( \pi \in \Pi \) is a late-bisimulation equivalence class. For this we define a relation \( \Phi \subseteq (L \times D) \times (L \times D) \) as follows, \( \Phi := \{((l_1, \bar{d}_1), (l_2, \bar{d}_2)) \mid \pi \in \pi \times \pi \} \).

We claim that \( \Phi \) is a late-bisimulation relation. Clearly \( \Phi \) is symmetric, and so it suffices to show that it is a simulation relation. Pick \((l_1, \bar{d}_1), (l_2, \bar{d}_2)\) ∈ \( \Phi \), then by definition of \( \Pi \), \( l_1 = l_2 := l \). Further if \((l_1, \bar{d}_1), (l_2, \bar{d}_2) \in \pi \in \Pi_2\), then \( \bar{d}_1 = \bar{d}_2 \), and so obviously \((l, \bar{d}_1)\) simulates \((l, \bar{d}_2)\). Consider next the case when \((l_1, \bar{d}_1), (l_2, \bar{d}_2) \in \pi \in \Pi_1\). Pick any \( e_1 \in E \). We claim that we can choose \( e_2 = e_1 := e \) to satisfy the definition of a simulation relation. Obviously \( \sigma_{e_2} = \sigma_{e_1} =: \sigma_e \). We need to further show that for all \( \bar{u} \in U \), \[ (l_1, \bar{d}_1) \xrightarrow{e, \bar{u}} (l'_1, \bar{d}'_1) \Rightarrow (l_2, \bar{d}_2) \xrightarrow{e, \bar{u}} (l'_2, \bar{d}'_2) \]

First note that \( l'_1 = l_1 = l_2 \), i.e., \( l'_1 = l'_2 \). Next note that \( (l_1, \bar{d}_1) \xrightarrow{e, \bar{u}} (l'_1, \bar{d}'_1) \Rightarrow l = o_e, \bar{d}_1 \in G_e(D, \{\bar{u}\}), \bar{d}'_1 = \bar{d}_1 = \bar{d}_2 = \bar{d}'_2 \)
reason. Suppose this may look restrictive, since the edge that in both cases, $\pi \in G_e(D, \{\vec{u}\})$, so we have, $d_2 \in \pi \subseteq G_e(D, \{\vec{u}\})$, i.e., e is enabled at $(d_2, \vec{u})$. We claim that if we let $d'_2 := f_e(d_2, \vec{u})$, then $[(l, d'_2) \in \sigma_{\vec{u}}(\vec{y}')] \subseteq \Phi$, from the third part of our sufficient condition $h_e(\pi, \vec{u}) = \{\vec{y}\}$. Since $d'_2 \in \pi$, this further implies that $h_e(d'_2, \vec{u}) = \vec{y}$. Thus indeed it holds that, $[(l, d'_2) \in \sigma_{\vec{u}}(\vec{y}')] \subseteq \Phi$.

It remains to be shown that $((l', d'_2), (l', d'_2)) \in \Phi$. By the second part of our suffcient condition, either $d'_2 = f_e(d_1, \vec{u})$ and some $d'_1 = f_e(d_2, \vec{u}) \in \pi_j$ for $j < m$ or $d'_1 = f_e(d_2, \vec{u}) \in \pi_j$ for $j < n$. It follows that in both cases, $((l', d'_2), (l', d'_2)) \in \Phi$.

Remark 2: The sufficient condition given in Theorem 1 demands that for state $(l, d_1)$ to be simulated by $(l, d_2)$, each edge $e_1$ with $o_{e_1} = l$ be “simulated” by the same edge $e_2 = e_1$ for all inputs $\vec{u} \in U$ such that $G_{e_i}(d_i, \vec{u})$ holds. This may look restrictive, since the edge $e_2$ does not have to be the same as the edge $e_1$, but it is not so for the following reasons. Suppose $e_2$ can be chosen differently from $e_1$, then we can refine the partition of the data space so that $d_1$ and $d_2$ are in two different partitions. Since there are only a finite number of edges, this refinement strategy will preserve the finiteness of the partition.

The following example illustrates Theorem 1.

Example 3: We revisit the I/O-EFA $P$ of Example 1 shown in Figure 1. As mentioned earlier $P$ has an infinite reachability set. Figure 2 shows a partition of the data space $D$ satisfying the condition of Theorem 1, where the members of the partition are as follows:

$$
\pi_1 = [2d(1) + d(2) < 2] \land [d(1) > 1], \quad \pi_2 = [2d(1) + d(2) < 2] \land [d(1) > 1], \\
\pi_3 = [2d(1) + d(2) < 2] \land [d(1) = 0], \\
\pi_5 = [2d(1) + d(2) < 2] \land [d(1) < 0]. \\
$$

Let $c_1$, $e_2$ and $e_3$ denote the edges from $A$ to $B$, $B$ to $C$ and $C$ to $A$, respectively. It can be seen from Table 1 that the sufficient condition of Theorem 1 holds (the entry “T” in the column for $h_{e_1}$ indicates that the third part of the sufficient condition holds True). It follows that $P$ admits a finite bisimilar quotient; it is shown in Figure 2, with the unreachable states omitted.

Remark 3: If there are no inputs and outputs (i.e., the case of closed systems), then the condition of Theorem 1 can be simplified to as follows. Given an EFA (I/O-EFA without inputs and outputs) $P = (L, D, \Sigma, E, L_0, D_0)$, it admits a finite late-bisimilar quotient if the data space of $P$ can be partitioned into a finite number of regions, $\pi_1, \ldots, \pi_n \subseteq D$ (where $\pi_1, \ldots, \pi_n$ are unbounded, and $\pi_{m+1}, \ldots, \pi_n$ are bounded) such that $\forall i \leq m$, $\forall e : \pi_i \cap G_e(D) \neq \emptyset \Rightarrow$

1. $\pi_i \subseteq G_e(D), \quad$ and

2. $(\exists j \leq m : f_e(\pi_i) \subseteq \pi_j) \lor (\forall d \in \pi_i, f_e(\pi_i) = \{f_e(d)\}).$

In the following proposition we present a weaker condition, than the one appearing in Theorem 1, that guarantees the existence of a finite bisimilar quotient. The proof is similar to Theorem 1 and omitted.

Proposition 1: For $l \in L, \sigma \in \Sigma, \vec{d} \in D, \vec{u} \in U$, define:

$$
G_{e_1}(D, U) := \{\vec{d}, \vec{u} \in D \times U \mid \exists e \in E \text{ s.t. } o_e = l, \sigma_e = \sigma, G_e(d, \vec{u})\}, \\
f_{i_1}(\vec{d}, \vec{u}) := \{\vec{d} \in D \mid \exists e \in E \text{ s.t. } o_e = l, \sigma_e = \sigma, \vec{d} = f_e(d, \vec{u})\}, \\
G_{e_1}(D) := \{\vec{d} \in D \mid \exists e \in E \text{ s.t. } o_e = l, \sigma_e = \sigma, \vec{y} = h_e(d, \vec{u})\}.
$$

Given an I/O-EFA $P$, it admits a finite bisimilar quotient if the data space $D$ can be partitioned into a finite number of regions, $\pi_1, \ldots, \pi_n \subseteq D$ (where $\pi_1, \ldots, \pi_m$ are unbounded, and $\pi_{m+1}, \ldots, \pi_n \subseteq D$ are bounded) such that $\forall i \leq m, \forall l' \forall \sigma, \forall \vec{u} : G_{e_1}(D, \{\vec{u}\}) \cap \pi_i \neq \emptyset \Rightarrow$

1. $\pi_i \subseteq G_{e_1}(D, \{\vec{u}\})$,

2. $\forall j \leq m : f_{i_1}(\pi_i, \vec{u}) \subseteq \pi_j \Rightarrow f_{e_1}(\pi_i, \vec{u}) \subseteq \pi_i \times \{\vec{u}\}$,

3. $\forall j < m \leq n, \forall \vec{d} \in \pi_j : \vec{d} \in f_{e_1}(\pi_i, \vec{u}) \Rightarrow f_{e_1}(\vec{d}) \supseteq \pi_i \times \{\vec{u}\}$, and

4. $\forall \vec{y} \in h_{e_1}(\pi_i, \vec{u}) : h_{e_1}(\vec{y}) \supseteq \pi_i \times \{\vec{u}\}$.

V. I/O-EFA SUBCLASS WITH FINITE BISIMILAR QUOTIENT

The condition of Theorem 1 for the existence of a finite late-bisimilar quotient is existential as it relies on the existence of a certain partition. In this section, we identify a subclass of I/O-EFAs for which a partition satisfying the sufficient condition of Theorem 1 can be constructed by inspecting the structure of a given I/O-EFA of the subclass. The subclass is obtained by imposing two conditions on the class of the I/O-EFAs. The first condition restricts the manner in which the transition guards are formed, and is specified by the following grammar.

Condition 1: $G(D, U) \rightarrow G(U) \mid d(i) \leq c \mid d(i) \geq c \mid \neg G(D, U) \mid G_1(D, U) \cap G_2(D, U)$,

where $G(U)$ is a predicate over inputs $U$ and $c$ is an integer constant (it can also be a rational constant but there is no loss of generality in restricting it to the domain of integers). According to the above grammar, an “atomic” guard is either a predicate over only the inputs, or it is a predicate over only the data and in which case it can be written as a simple inequality constraint over one of the data components. A generic guard is a boolean combination of the atomic guards.

When guards are formed using the above grammar, there exists a natural partition of the data space into a finite number of regions. In order to define such a partition, we let $d(i)_{\text{max}}$ and $d(i)_{\text{min}}$ denote the largest and smallest integer against which the $i$th data variable is compared over all the guards. Then the domain of $i$th data component is naturally partitioned into up to three regions: One where $d(i)$ is below $d(i)_{\text{min}}$, another where it is in between $d(i)_{\text{min}}$ and $d(i)_{\text{max}}$, and the last one where $d(i)$ exceeds $d(i)_{\text{max}}$. This (3-way partition
of the domain of the \( i \)-th data component naturally yields a \((\leq 3^p)\)-way partition of the entire data space (recall \( p \) is the dimension of data space), only at most one of which is bounded.

Since each element within a bounded region is to be its own late-bisimulation equivalence class, for each \( \bar{d} \in D \) we define its equivalence class, \( [\bar{d}] \subseteq D \) as in the following definition. We first define index sets \( I_{\max}(\bar{d}) \) and \( I_{\min}(\bar{d}) \) containing indices \( i \leq p \) for which \( d(i) \) is above \( d(i)_{\max} \) and below \( d(i)_{\min} \) respectively. We let \( I := \{1, \ldots, p\} \) denote the set of data components.

**Definition 6:** Given an I/O-EFA satisfying Condition 1, for \( \bar{d} \in D \) define
\[
I_{\max}(\bar{d}), I_{\min}(\bar{d}) \subseteq I = \{1, \ldots, p\} \text{ as:}
\[
\{ i \in I_{\max}(\bar{d}) \} \Leftrightarrow [d(i) > d(i)_{\max}], \quad \{ i \in I_{\min}(\bar{d}) \} \Leftrightarrow [d(i) < d(i)_{\min}].
\]

Define an equivalence class of \( \bar{d} \in D \), denoted \( [\bar{d}] \subseteq D \), as:
\[
[\bar{d}] := \{ \bar{d} \in D \ \mid \ I_{\max}(\bar{d}) = I_{\max}(\bar{d}) := I_{\max} \}.
\]
\[
\land [I_{\min}(\bar{d}) = I_{\min}(\bar{d}) := I_{\min} \} \land [d(i) = d(i), \forall i \in I - I_{\max} - I_{\min}].
\]

Note that for any \( \bar{d} \in D \), \( I_{\max}(\bar{d}) \cap I_{\min}(\bar{d}) = \emptyset \), or equivalently, \( I_{\max}(\bar{d}) \subseteq I - I_{\min}(\bar{d}) \), or equivalently, \( I_{\min}(\bar{d}) \subseteq I - I_{\max}(\bar{d}) \).

The following lemma states that the equivalence class of \( \bar{d} \) is well defined.

**Lemma 1:** Consider the definition of equivalence class \([\bar{d}]\) for \( \bar{d} \in D \) given in Definition 6. Then for any \( \bar{d}, \bar{d}' \in D \),
\[
(\bar{d} \in [\bar{d}]) \iff \left( I_{\max}([\bar{d}]) = \max(\bar{d}) := I_{\max} \right) \land \left( I_{\min}([\bar{d}]) = \min(\bar{d}) := I_{\min} \right) \land \left( [\bar{d}'(i) = d(i), \forall i \in I - I_{\max} - I_{\min}] \right).
\]

**Proof:** It follows from Definition 6 that
\[
(\bar{d} \in [\bar{d}]) \iff \left( I_{\max}([\bar{d}]) = \max(\bar{d}) := I_{\max} \right) \land \left( I_{\min}([\bar{d}]) = \min(\bar{d}) := I_{\min} \right) \land \left( [\bar{d}'(i) = d(i), \forall i \in I - I_{\max} - I_{\min}] \right).
\]

On the other hand,
\[
(\bar{d}' \in [\bar{d}]) \iff \left( I_{\max}([\bar{d}]) = \max(\bar{d}) := I_{\max} \right) \land \left( I_{\min}([\bar{d}]) = \min(\bar{d}) := I_{\min} \right) \land \left( [\bar{d}'(i) = d(i), \forall i \in I - I_{\max} - I_{\min}] \right).
\]

It follows that if (i) \( \bar{d}' \in [\bar{d}] \), which is equivalent to the right hand side of the second equivalence, and (ii) \( \bar{d} \in [\bar{d}] \), which is equivalent to the right hand side of the first equivalence, then by combining the two right hand sides and applying a transitivity argument we obtain a condition that is equivalent to \( \bar{d}' \in [\bar{d}] \). So it holds that if \( \bar{d}' \in [\bar{d}] \) and \( \bar{d} \in [\bar{d}] \), then \( \bar{d}' \in [\bar{d}] \), as desired.

**Remark 4:** It can be verified that the total number of equivalence classes, i.e., \( \{[\bar{d}] \mid \bar{d} \in D\} \) is given by
\[
\sum_{t_1 \leq t_2 \leq I} \Pi_{i \in I_1, t_1 - t_2}(d(i)_{\max} - d(i)_{\min} + 1).
\]

Also note that an equivalence class \([\bar{d}]\) is unbounded if for some \( i \) either \( d(i) > d(i)_{\max} \) or \( d(i) < d(i)_{\min} \), or
equivalently if $I^\max(d) \cup I^\min(d) \neq \emptyset$. So there are a total of $\prod_{i \in I}(d(i)^\max - d(i)^\min + 1)$ number of bounded equivalence classes, each one of which is a singleton. The remaining equivalence classes are all unbounded. We denote the set of all equivalence classes by $D$, and the subset of unbounded and bounded ones by $D_u$ and $D_b$, respectively.

The second condition given below imposes restriction on the data update functions and output assignment functions.

**Condition 2:** $\forall \vec{d}, \forall e, \forall \vec{u}$:

$$(\vec{d}, \vec{u}) \in G_e(D, U) \Rightarrow$$

1. $\forall i \in I : (f_e(i)(\vec{d}, \vec{u}) \geq d(i)^\max) \lor (f_e(i)(\vec{d}, \vec{u}) \leq d(i)^\min)$

2. $h_e(\vec{d}, \vec{u}) = \{h_e(\vec{d}, \vec{u})\}$

**Proof:** By Condition 1, every guard $G_e(D, U)$ is a boolean combination of atomic guards of the type, $G_e(U), [d(i) \leq c_i], [d(i) \geq c_i]$. Now suppose the boolean combination of atomic guards is written in disjunctive normal form (DNF), then for the guard to be satisfied, one of the disjuncts in the DNF must be satisfied. Since each disjunct in a DNF is a conjunct of atomic guards (or their negations), for a disjunct in a DNF to be satisfied, each atomic guard (or its negation) in the disjunct must be satisfied. So in order to show that whenever $\vec{d}$ satisfies a guard $G_e(D, U)$, it implies that $[\vec{d}] \subseteq G_e(D, U)$, it suffices to show that whenever $\vec{d}$ satisfies an atomic guard, all elements of the equivalence class $[\vec{d}]$ satisfies an atomic guard. Clearly this holds when the atomic guard is a predicate over only the inputs. So now consider an atomic guard of the type, $[d(i) \leq c_i]$. Then by definitions of $d(i)^\min$ and $d(i)^\max$, we have $d(i)^\min \leq c_i \leq d(i)^\max$. So $\vec{d}$ satisfies $[d(i) \leq c_i]$ if and only if either $[d(i) < d(i)^\min \leq c_i]$ or $[d(i)^\min \leq d(i) \leq c_i \leq d(i)^\max]$. In the first case, $i \in I^\min(\vec{d})$, whereas in the second case $i \in I - I^\max(\vec{d}) - I^\min(\vec{d})$. If former, then it follows from the definition of equivalence class that $i \in I^\min(\vec{d})$ for all $\vec{d} \in [\vec{d}]$; whereas if latter, then again it follows from the definition of equivalence class that $i \in I - I^\max(\vec{d}) - I^\min(\vec{d})$ for all $\vec{d} \in [\vec{d}]$. Thus an atomic guard of the type $[d(i) \leq c_i]$ is satisfied by $\vec{d}$ if and only if that guard is satisfied by each element in $[\vec{d}]$. From an analogous argument, the same property is enjoyed by an atomic guard of the form $[d(i) \geq c_i]$. This establishes the first condition appearing in the theorem.

To prove that the second condition appearing in the theorem holds, we consider two cases. First when condition 2 holds in such a way that for all $i \in I$, $f_e(i)(\vec{d}, \vec{u}) = \{f_e(i)(\vec{d}, \vec{u})\}$, then this implies $f_e(\vec{d}, \vec{u}) = \{f_e(\vec{d}, \vec{u})\}$, which is the second disjunct of the second condition appearing in the theorem. The second possibility is that there exists $i \in I$ such that either $f_e(i)(\vec{d}, \vec{u}) \geq d(i)^\max$ or $f_e(i)(\vec{d}, \vec{u}) \leq d(i)^\min$. Let $I^\max := \{i \in I \mid f_e(i)(\vec{d}, \vec{u}) \geq d(i)^\max\}$ and $I^\min := \{i \in I \mid f_e(i)(\vec{d}, \vec{u}) \leq d(i)^\min\}$. Let $\vec{d} \in D$ be such that $I^\max(\vec{d}) = I^\max$ and $I^\min(\vec{d}) = I^\min$. Then it is easy to see that $[\vec{d}] \subseteq D_u$ and $f_e(\vec{d}, \vec{u}) \subseteq [\vec{d}]$.

Finally it is obvious that the second part of Condition 2 implies the third condition appearing in the theorem.

The following corollary follows from Theorems 1 and 2.

**Corollary 1:** Given an I/O-EFA $P$ satisfying Conditions 1 and 2, $P$ admits a finite late-bisimilar quotient.

**Remark 5:** It follows from the development above that each state in a finite late-bisimilar quotient of $P$ satisfying Conditions 1 and 2 is of the form $\{i\} \times [\vec{d}]$, where $[\vec{d}] \subseteq D$ is defined in Definition 6. Thus the number of states in a finite late-bisimilar quotient of $P$ is given by.

$$|L| \times (\sum_{l_1 \leq l_2 \leq l_1} \prod_{i \in I - l_2 - l_2} (d(i)^\max - d(i)^\min + 1))$$

**Remark 6:** If there are no inputs and outputs (i.e., the case of closed systems), then Conditions 1 and 2 can be simplified as follows.

**Condition 1’:** $G(D) \longrightarrow d(i) \leq c \lor d(i) \geq c \lor \neg G(D) \lor G_1(D) \land G_2(D)$, and

**Condition 2’:** $\forall \vec{d}, \forall e$:

$$\vec{d} \in G_e(D) \Rightarrow \forall i \in I : \left(f_e(i)(\vec{d}, \vec{u}) \geq d(i)^\max\right) \lor \left(f_e(i)(\vec{d}, \vec{u}) \leq d(i)^\min\right)$$

Then from Corollary 1, an EFA $P = (L, I, D, \Sigma, E, L_0, D_0)$ admits a finite late-bisimilar quotient if the two conditions stated above in this remark hold.

**Example 4:** Consider the following program:

```c
int main() {
    int d1 = 1, d2 = 2;
    while(1) {
        A: if (d1 \leq 0 && c2 \geq 0) {d1 = d1 + 1;} else break;
        B: if (d1 \geq 1 && c2 \leq 2) {d2 = 1;} else break;
        C: if (d1 \geq 1 && c2 \leq 1) {d1 = d1 + d2;} else break;
    }
    return 0;
}
```

An EFA model $P$ of the above program is shown in Figure 3. One can see by inspection that all guards satisfy Condition 1’, and $d(1)^\min = d(2)^\min = 0$ and $d(1)^\max = d(2)^\max = 2$. The equivalence classes representing the states of a finite late-bisimilar quotient are shown in Figure 3. Let $e_1, e_2$ and $e_3$ denote edges from $A$ to $B$, $B$ to $C$ and $C$ to $A$, respectively. It can be verified that Condition 2’ also holds (the details are omitted). So $P$ admits a finite late-bisimilar quotient even though the reachability set of $P$ is infinite. The finite late-bisimilar quotient system is shown in Figure 3, with the unreachable states omitted.
Remark 7: Note that the sufficient condition of Theorem 2 cannot be derived as a special case of the known sufficient conditions for finite bisimilar abstraction of a hybrid system as surveyed in [3]. There, one type of sufficient condition requires that any time a location (discrete-state) is switched, the data-variables get reset. This is clearly not required by Theorem 2 as is evident from Example 4 (see Figure 3), where transitions between locations A and B, and also between locations C and A do not reset the data-variables. Another type of sufficient condition in [3] requires an “O-minimal” structure. Theorem 2 does not impose any such requirement either. Thus the result of Theorem 2 cannot be derived as a special case of those for hybrid systems.

VI. Iterative Computation of Late-Bisimilar Quotient

In this section, we first present an equivalent form of Theorem 1, which is found easier to use in the subsequent analysis. Then we present an iterative algorithm that upon termination yields a finite late-bisimilar quotient: Starting from the coarsest partition candidate that may satisfy the conditions of Theorem 1, the algorithm iteratively computes a finer partition guided by the condition of Theorem 1 until the partition converges, and in which case the given I/O-EFA admits a finite late-bisimilar quotient.

Remark 8: Given an I/O-EFA $P = (L, D, U, Y, \Sigma, E, L_0, D_0)$, we can modify $P$ to absorb the initial condition $D_0$ according to Figure 4. Then without loss of generality, we assume that $D_0$ is same as True.

For a predicate $\pi$ over the data variables, the notation $\pi(f_e(\vec{d}, \vec{u}))$, where $f_e: D \times U \rightarrow D$ is the update function associated with a certain edge $e \in E$, denotes the predicate $\pi$ with the variable $\vec{d}$ substituted with $f_e(\vec{d}, \vec{u})$. $\pi(f_e(\vec{d}, \vec{u}))$ is the predecessor of $\pi$ along the edge $e$ under the input $\vec{u}$. An equivalent form of Theorem 1 is next stated.

Theorem 3: Given an I/O-EFA $P$, it admits a finite late-bisimilar quotient if there exists a partition $\Pi$ of its data space such that $\forall \pi \in \Pi, e \in E, \vec{u} \in U$:

$$\pi \Rightarrow \forall \vec{d} \in D : \neg G_e(\vec{d}, \vec{u}) \vee [G_e(\vec{d}, \vec{u}) \wedge \exists \pi' \in \Pi, \vec{y} \in Y : \pi'(f_e(\vec{d}, \vec{u})) \wedge [h_e(\vec{d}, \vec{u}) = \vec{y}]].$$

Theorem 3 states that exists a partition $\Pi$ such that for each class $\pi \in \Pi$, each edge $e \in E$, and input $\vec{u} \in U$, either $\pi$ is stronger than $G_e(\vec{d}, \vec{u})$ (so $e$ is enabled nowhere in $\{o_e\} \times \pi$), or $\pi$ is stronger than $G_e(\vec{d}, \vec{u})$ (so $e$ is enabled everywhere in $\{o_e\} \times \pi$) and at the same time exists an equivalence class $\pi' \in \Pi$ and output $\vec{y} \in Y$ such that along edge $e$ and under input $\vec{u}$ successors of $\pi$ are contained in $\pi'$, while the same output $\vec{y}$ is produced everywhere in $\pi$.

We first show that any partition $\Pi$ that satisfies the condition of Theorem 3 must be finer than the one induced by the set of data predicates appearing as the guard and assignment conditions of an I/O-EFA, where a partition induced by a set of predicates is defined as follows.

Definition 7: Given a set of data predicates $\Theta$ defined over a data space $D$, the partition of the data space induced by $\Theta$, denoted $\Pi^\Theta$, is defined by

$$\Pi^\Theta := \left\{ \bigwedge_{\theta \in \Theta} \bigwedge_{\theta' \in \Theta - \theta} \neg \theta' \mid \hat{\Theta} \subseteq \Theta \right\}.$$

It can be easily verified that $\Pi^\Theta$ is a partition, i.e., (i) $\pi, \pi' \in \Pi^\Theta$ and $\pi \neq \pi'$ imply that $\pi \wedge \pi' = False$, and (ii) $\bigvee_{\Theta \subseteq \Theta} [\bigwedge_{\theta \in \Theta} \theta \bigwedge_{\theta' \in \Theta - \hat{\Theta}} \neg \theta'] = D.
Proposition 2: Consider an I/O-EFA $P = (L, D, U, Y, \Sigma, E, L_0, D_0)$ with a set of data predicates of the guards and the assignments: $\Theta := \bigcup_{e \in E, \vec{u} \in U, \vec{y} \in Y} \{G_e(\vec{d}, \vec{u}) \land [e], \vec{d}, \vec{u} = \vec{y}\}$. If a partition $\Pi$ satisfies the condition of Theorem 3, then $\Pi$ is finer than $\Pi^\Theta$.

Proof: Pick $\pi \in \Pi$. We need to show that exists $\pi \in \Pi^\Theta$ such that $\pi \Rightarrow \pi$. Since $\pi \in \Pi$ for each $e \in E, \vec{u} \in U$, either $\pi \Rightarrow \neg G_e(\vec{d}, \vec{u})$ or exist $\pi' \in \Pi, \vec{y} \in Y$ such that $\pi \Rightarrow G_e(\vec{d}, \vec{u}) \land \pi'(f_e(\vec{d}, \vec{u})) \land [e], \vec{d}, \vec{u} = \vec{y}$. If former, we can choose $\pi = \neg G_e(\vec{d}, \vec{u})$ and if latter, we can choose $\pi = G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]$. Since $G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}] \in \Theta$, and $\neg G_e(\vec{d}, \vec{u}) = \neg \forall \vec{y} G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]$, it follows that in either case $\pi \in \Pi^\Theta$, as desired.

The above proposition suggests that it may be possible to refine the partition induced by the set of predicates appearing as guard and assignment conditions to obtain a partition satisfying Theorem 3. The algorithm we present in the following performs such a refinement. A possible way to achieve refinement is to introduce additional data predicates as suggested by the following lemma.

Lemma 2: Given two sets of data predicates $\Theta_1$ and $\Theta_2$ defined over a data space $D$ such that $\Theta_1 \subseteq \Theta_2$, then $\forall \pi_1 \in \Pi^{\Theta_1, \exists} \exists \pi_2 \in \Pi^{\Theta_2}$ such that $\pi_2 \Rightarrow \pi_1$, where $\Pi^{\Theta_1}$ and $\Pi^{\Theta_2}$ are computed by Definition 7.

Proof: Pick $\pi_1 \in \Pi^{\Theta_1}$. Then exists $\Theta_0 \subseteq \Theta_1$ such that $\pi_1 = \bigwedge_{\theta \in \Theta_0} \theta \bigwedge_{\theta' \in \Theta_0} \neg \theta'$. Define $\pi_2 = \bigwedge_{\theta' \in \Theta_2} \theta' \bigwedge_{\theta \in \Theta_2} \neg \theta$. Then we have $\pi_2 \in \Pi^{\Theta_2}$ and $\pi_2 \Rightarrow \pi_1$.

Next we present an iterative algorithm that upon termination yields a finite late-bisimilar quotient. At each iteration of the algorithm, a new set of predicates are introduced (so that a partition that is finer than the one from the previous iteration is obtained). The computation of the new set of predicates is guided by the condition of Theorem 3, and thus, the partition obtained in each iteration is guided “closer” towards a partition that satisfies Theorem 3 (if one exists).

Algorithm 1: Consider an I/O-EFA $P = (L, D, U, Y, \Sigma, E, L_0, D_0)$.

1. Let $k = 0$. $\Theta^k = \bigcup_{e \in E, \vec{u} \in U, \vec{y} \in Y} \{G_e(\vec{d}, \vec{u}) \land [e], \vec{d}, \vec{u} = \vec{y}\}$.
2. $\Theta^{k+1} = \Theta^k \bigcup_{\vec{d} \in D, \vec{u} \in U, \vec{y} \in Y, e \in E, \vec{y} \in \Pi^{\Theta^k}} \{G_e(\vec{d}, \vec{u}) \land \pi'(f_e(\vec{d}, \vec{u})) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]\}$.
3. If $\Pi^{\Theta^{k+1}} = \Pi^{\Theta^k}$, then stop; else let $k := k + 1$ and go to step 2.

Note $\Theta^k$ is infinite in general. There are special cases when it is finite: (i) When input and output sets are finite, and (ii) When guard condition is generated by the following grammar:

$$G(\vec{d}, \vec{u}) \rightarrow G(\vec{d}) \mid G(\vec{u}) \mid \neg G(\vec{d}, \vec{u}) \mid G_1(\vec{d}, \vec{u}) \land G_2(\vec{d}, \vec{u}),$$

and the output set is finite. Also note when there are no inputs and outputs, $\Theta^0$ and $\Theta^{k+1}$ are reduced to $\bigcup_{e \in E} \{G_e(\vec{d})\}$ and $\Theta^k \bigcup_{e \in E, \vec{y} \in \Pi^{\Theta^k}} \{\pi'(f_e(\vec{d}))\}$, respectively.

Remark 9: The complexity of the partition computed by Algorithm 1 can be specified in terms of the sizes of the inputs, outputs and edges: It can be noted that $|\Theta^0| = O(|U||Y||E|)$, and $|\Theta^{k+1}| = O(|U||Y||E||\Theta^k|)$. Thus if Algorithm 1 terminates in $n$-iterations, then $|\Theta^n| = O((|U||Y||E|)^{n+1})$, which determines the complexity of the computed partition.

Theorem 4: An I/O-EFA satisfies condition of Theorem 3 if and only if Algorithm 1 terminates.

Proof: ($\Rightarrow$) Suppose that the algorithm terminates in the $k$th step, and suppose for contradiction that the condition of Theorem 3 is not satisfied by $\Pi^{\Theta^k}$. Then exists $\pi \in \Pi^{\Theta^k}, e \in E, \vec{u} \in U$ such that $\pi \Rightarrow \neg G_e(\vec{d}, \vec{u})$ and for all $\pi \in \Pi^{\Theta^k}, \vec{y} \in Y$, it holds that $\pi \Rightarrow G_e(\vec{d}, \vec{u}) \land \pi'(f_e(\vec{d}, \vec{u})) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]$. Since $\pi \in \Pi^{\Theta^k}$ and for each $\vec{u} \in U, e \in E, G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}] \in \Theta^k \subseteq \Theta^k$, it must be the case that $\pi'(f_e(\vec{d}, \vec{u}))$ not in $\Theta^k$ for all $\pi \in \Pi^{\Theta^k}$. If latter, we $\pi \in \Pi^{\Theta^k}$. Then, it follows that $\Theta^k \neq \Theta^{k+1}$, a contradiction.

($\Rightarrow$) Suppose exists a partition $\Pi$ such that the condition of Theorem 3 holds. We first show using induction on $k$ that $\Pi^{\Theta^k}$ is coarser than $\Pi$. The base step ($k = 0$) holds from the definition of $\Theta^0$ and Proposition 2. For the induction step, we suppose as part of the induction hypothesis that $\Pi^{\Theta^k}$ is coarser than $\Pi$, and pick $\pi \in \Pi$. Then we need to show that exists $\pi \in \Pi^{\Theta^{k+1}}$ such that $\pi \Rightarrow \pi$. Since $\pi \in \Pi$ for each $e \in E, \vec{u} \in U$, either $\pi \Rightarrow \neg G_e(\vec{d}, \vec{u})$ or exist $\pi' \in \Pi, \vec{y} \in Y$ such that $\pi \Rightarrow G_e(\vec{d}, \vec{u}) \land \pi'(f_e(\vec{d}, \vec{u})) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]$. If former, we can choose $\pi = \neg G_e(\vec{d}, \vec{u})$ and if latter, we can choose $\pi = G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]$. Since $\Theta^k \bigcup_{e \in E} \{G_e(\vec{d}, \vec{u}) \land \pi'(f_e(\vec{d}, \vec{u})) \land [h_e(\vec{d}, \vec{u}) = \vec{y}]\}$, it follows that in either case $\pi \in \Pi^{\Theta^{k+1}}$, as desired. Finally we show that $k$ such that $\Pi^{\Theta^k} = \Pi^{\Theta^{k+1}}$. Suppose not true, i.e., for each $k$, $\Pi^{\Theta^{k+1}}$ is strictly finer than $\Pi^{\Theta^k}$. Then since $\Pi$ is a finite partition (satisfying the condition of Theorem 3), exists $k$ such that $|\Pi^{\Theta^k}| > |\Pi|$. This contradicts the fact that $\Pi$ is finer than $\Pi^{\Theta^k}$ for all $k$.■

Since Theorem 3 applies to a larger class of I/O-EFAs as compared to Theorem 2, Theorem 4 suggests that the set of I/O-EFAs identified by Algorithm 1 subsumes that identified by Theorem 2. So if the condition of Theorem 2 does not apply, Algorithm 1 may be used. The following example illustrates Algorithm 1.

Example 5: Consider the I/O-EFA $P$ shown in Figure 5. Then

$$G_e(\vec{d}, \vec{u}) \land [h_e(\vec{d}, \vec{u}) = \vec{y}] = \begin{cases} \forall_{\vec{y} \in [-\infty, \infty]} & (2 \leq d \leq 3) \land [1 + d = y] \quad \text{if } u \text{ mod } 2 = 1 \\ \forall_{\vec{y} \in [-\infty, \infty]} & (2 \leq d \leq 3) \land [d = y] \quad \text{if } u \text{ mod } 2 = 0 \end{cases}$$
It can be verified that the condition of Theorem 2 does not apply.

The predicate \([2 \leq d \leq 3] \land [1 + d = y]\) is equal to \([d = 2], [d = 3]\), and False, when \(y = 3, y = 4\), and \(y \notin \{3, 4\}\). On the other hand, \([2 \leq d \leq 3] \land [d = y]\) is equal to \([d = 2], [d = 3]\), and False, when \(y = 2, y = 3\), and \(y \notin \{2, 3\}\), respectively. The computation of \(\Theta^0 = \{[d = 2], [d = 3]\}\). It follows that \(\Pi\Theta^0 = \bigcup_{i=1}^{2} \pi_i\), where \(\pi_1 = [d <\ 2], \pi_2 = [d = 2], \pi_3 = [d = 3]\), and \(\pi_4 = [d > 3]\).

The computation of \(\Theta^1 = \Theta^0\), and so Algorithm 1 terminates. The finite late-bisimilar quotient is shown in Figure 5.

Remark 10: Algorithm 1 presented above is similar in spirit to the algorithm Closure1 presented in [14]. The main difference is that Algorithm 1 has been developed for applying it to an I/O-EFA, which is a more structured object than a symbolic transition system. (Note that a symbolic transition system does not have the notions of variables, guard conditions, assignment functions, or update functions.)

VII. CONCLUSION

We considered a model for the reactive untimed infinite state systems, called input/output extended finite automaton (I/O-EFA), which is an automaton extended with discrete variables such as inputs, outputs, and data. The modeling of a class of reactive software as I/O-EFAs is explained in Appendix A. We presented a sufficient condition under which such a model admits a finite late-bisimilar quotient (and hence also a finite bisimilar quotient as late-bisimilarity implies bisimilarity). We then identified a class of I/O-EFAs for which a partition satisfying our sufficient condition can be constructed by inspecting the structure of the given I/O-EFA. The identified class of I/O-EFAs has the property that any I/O-EFA in this class is bisimilar to a finite-state system and hence decidable. For cases when a given I/O-EFA fails to belong to the said class, we presented an iterative refinement algorithm whose termination guarantees the existence of a finite bisimilar abstraction. The results are illustrated through examples that model reactive software. The application of the theory presented in the paper to a practical setting will be an interesting future direction to pursue. Also, our aim in the paper is to determine whether or not a system possesses a finite late-bisimulation quotient, and we are not concerned about finding a minimal finite quotient. Clearly, a minimal quotient is finite if and only if a finite quotient exists. So once a finite quotient has been found, further analysis may be carried out to find a minimal one, which is again a future direction for research.

APPENDIX

A reactive software reacts to the environmental stimuli (inputs) and affects the environmental state by producing responses (outputs). Examples of reactive programs are human-machine interfaces, computer games, etc. The C language is widely used in many applications of reactive programming. We describe the syntax of a certain class of reactive software (that can be viewed as a subclass of the C programs) and show how it can be modeled using I/O-EFAs. A program consists of a sequence of statements, a grammar for which is presented below. The statements include formula expressions and predicates, grammars for which are also presented below.

Definition 8: The grammar for defining program statements is defined in terms of those for formula expressions and predicates:

- Grammar for formula expressions (for defining predicates): \(\xi \rightarrow n \in \mathcal{N} | \bar{d} | f(d, u) | \xi_1 + \xi_2 | \xi_1 - \xi_2 | \xi_1 \times \xi_2\).
- Grammar for predicates (for specifying conditions in program statements): \(\theta \rightarrow \text{true} | \xi_1 \leq \xi_2 | \neg \xi | \xi_1 \land \xi_2\).
- Grammar for program statements: \(\eta \rightarrow \bar{d} := \xi | \eta_1 ; \eta_2 \) if \(\theta\) then \(\eta_1\) else \(\eta_2\) while \(\theta\) do \(\eta\) scan \& \(d(i)\) print \(\xi\).

Figure 6 shows the translation rules from program statements to I/O-EFAs. Using these translations any program generated using the grammar of Definition 8 can be translated into an I/O-EFA.

Example 6: We illustrate the translation rules by constructing and I/O-EFA model starting from an example program. The program repeatedly accepts two (positive) integers entered from keyboard by user and outputs their greatest common divisor (gcd). User has the option to terminate the process by entering ‘0’. The associated I/O-EFA model is shown in Figure 7.

REFERENCES


\( y := d + u \mod 2 \)
\( d := u \mod 5 \)
\( 2 \leq d \leq 3 \)
\( u \mod 10 = 3, 8 \)
\( u \mod 10 = 0, 1, 5, 6 \)
\( u \mod 10 = 3, 8 \)
\( u \mod 10 = 4, 9 \)
\( u \mod 10 = 2, 7 \)

Fig. 5. I/O-EFA \( P \) (left) and its finite late-bisimilar quotient (right)

Fig. 6. Translation rule from program statements to I/O-EFAs

Fig. 7. I/O-EFA of gcd program of Example 6
TABLE II  

<table>
<thead>
<tr>
<th>$\alpha$ mod 10</th>
<th>$G_{c} \land \pi_1(f_{c}) \land {h_{c} = y}$</th>
<th>$G_{c} \land \pi_2(f_{c}) \land {h_{c} = y}$</th>
<th>$G_{c} \land \pi_3(f_{c}) \land {h_{c} = y}$</th>
<th>$G_{c} \land \pi_4(f_{c}) \land {h_{c} = y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>$F \land {d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
</tr>
<tr>
<td>3</td>
<td>$F \land {d = 2}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>$F \land {d = 2}$</td>
<td>${d = 2}$</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
<td>${d = 2}, {d = 3}$</td>
</tr>
<tr>
<td>6</td>
<td>$F \land {d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
</tr>
<tr>
<td>7</td>
<td>$F \land {d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
</tr>
<tr>
<td>8</td>
<td>$F \land {d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
</tr>
<tr>
<td>9</td>
<td>$F \land {d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
<td>${d = 2}$</td>
</tr>
</tbody>
</table>


Changyan Zhou (S04-M08) received the B.S. degree in Mechanical Engineering from Northwestern Polytechnical University, Xian, China, the M.S. degree in Mechatronic Engineering from the Harbin Institute of Technology, Harbin, China, and the Ph.D. degree in Electrical and Computer Engineering from Iowa State University, Ames, IA, in 1993, 1996, and 2007, respectively. From 2006-2007 she held a visiting position at the Department of Industrial and Enterprise Systems Engineering of University of Illinois at Urbana-Champaign. Since 2008 she has been with Magnatech LLC at East Granby, CT. Her research interests include control and diagnosis of embedded/event-driven systems.

Ratnesh Kumar (S87-M90-SM00-F07) received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology at Kanpur, India, in 1987, and the M.S. and the Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin, Texas, in 1989 and 1991, respectively. From 1991-2002 he was on the faculty of University of Kentucky, and since 2002 he is on the faculty of the Iowa State University. He has held visiting position at the Institute of Systems Research at the University of Maryland at College Park, the Applied Research Laboratory at the Pennsylvania State University, the NASA Ames Research Center, the Argonne National Laboratory West, and the United Technology Research Center. He was a recipient of the Microelectronics and Computer Development (MCD) Fellowship from the University of Texas at Austin, and was awarded the Lalit Narain Das Memorial Gold Medal for the Best EE Student and the Ratan Swapur Memorial Gold Medal for the Best All-rounder Student from the Indian Institute of Technology at Kanpur, India. He is a recipient of the NSF Research Initiation Award, NASA-ASEE summer faculty fellowship award, and coauthor of the book Modeling and Control of Logical Discrete Event Systems, Kluwer Academic Publishers, 1995. He serves on the program committee for the IEEE Control Systems Society, the International Workshop on Discrete Event Systems, and the IEEE Workshop on Software Cybernetics. He is or has been an associate editor of *SIAM Journal on Control and Optimization, IEEE Transactions on Robotics and Automation, Journal of Discrete Event Dynamical Systems, and IEEE Control Systems Society*. He is a Fellow of the IEEE.