12th IFAC/IEEE Workshop on Discrete Event Systems
Cachan, France. May 14-16, 2014

Symbolic Interpretation and Execution of Extended Finite Automata ⋆

Mohammad Reza Shoaei * **Bengt Lennartson** *

* *Department of Signals and Systems, Chalmers University of Technology, SE-412 96, Gothenburg, Sweden (email: {shoaei, bengt.lennartson}@chalmers.se).*

Abstract: We introduce a symbolic interpretation and execution technique for Extended Finite Automata (EFAs) and provide an interpreter that symbolically interprets and executes EFAs w.r.t. their (internal) variables. More specifically, the interpreter iterates over the EFA transitions, and by passing each transition, it symbolically interprets and evaluates the condition on the transition w.r.t. the known values of variables, and leaves other variables intact, and when it terminates, it returns the residual model. It is shown that the behavior of the residual system with respect to the original system is left unchanged. Finally, we demonstrate the effectiveness and necessity of the symbolic interpretation and execution combined with abstractions for the nonblocking supervisory control of two manufacturing systems.

Keywords: Discrete-event systems; symbolic interpretation; supervisory control theory.

1. INTRODUCTION

Traditionally, finite-state automata have been used for the supervisory control of discrete-event systems (DES), Cassandras and Lafortune [2008] and Wonham [2013], which has been found to be non-trivial for complex systems with data. Modeling using *Extended Finite Automata* (EFAs), i.e., an ordinary finite automaton whose transitions are augmented with *variable updates*, makes it possible to, efficiently and in a compact form, model DES that involve non-trivial data manipulation, see Skoldstam et al. [2007].

A challenge with this new control framework is to symbolically interpret and optimize the models before synthesizing the controller in order to be able to exploit various abstraction methods, such as Shoaei et al. [2012] and Mohajerani et al. [2013]; reducing the complexity and more often avoiding state-space explosion. To this end, a naive attempt would be to expand the domain of "internal" variables on every transition of the system. This is, however, not efficient (in particular, for variables with large domain) as it requires to "blindly" expand the domain, not only those particular values which are required.

To overcome this problem, we introduce a *symbolic interpretation and execution* (or just interpretation) technique for EFAs. The interpretation process is performed by an interpreter $[\![.]\!]$ that iterates over the EFA transitions and, instead of blind expansion of the domain of variables, it symbolically interprets and executes, or more specifically, partially evaluates the condition on that transitions w.r.t. the known variables value in the context. When $[\![.]\!]$ terminates, it returns the "residual" EFA model.

The overall motivation for interpretation of EFAs is that analyzing the residual models is often more efficient than analyzing the original ones, since the interpreter $[\![.]\!]$ has already pre-executed the portions of system that depend on the internal variables without computing the global (explicit) model. This pays off when, e.g, one seeks for abstraction possibilities to further reduce the complexity of the system before constructing the global model. Another application of EFA interpretation can be seen in the process of synthesizing a supervisor for EFAs using BDDs, see Miremadi et al. [2012]. In this, one can, instead of directly convert the EFA models to BDDs, first interpret and execute the (internal) variables and simplify the models, then convert the residual models to BDDs. This can, sometimes significantly, help to decrease the number of BDD variables and avoid (possible) out of memory errors.

In this paper, we provide an algorithm that implements the interpreter $[\![.]\!]$. Further, we formulate the partial evaluation (execution) process by a proof calculus, of which we show its soundness. Furthermore, for the purpose of supervisory control, we provide sufficient conditions to guarantee that the behavior of the residual system is left unchanged compared to the original system, hence resulting in maximally permissive and nonblocking control to the entire system by using the interpreted models.

We note that the proposed technique is conceptually similar to that of program execution, cf. Jones et al. [1993] and Hatcliff [2003]. In this paper, however, we provide a basic starting point to bring the advantages of the symbolic interpretation and execution to DES with data and to the best of our knowledge, it is the first attempt to use such a technique for the purpose of supervisory synthesis. This paper also demonstrates the importance of using not only abstractions, but also to include the symbolic interpretation to obtain significant state reduction before ordinary synthesis.

The rest of the paper is organized as follows. Section 2 briefly recall the predicates, their syntax and semantics, and defines EFAs. In Section 3, we introduce the symbolic interpretation and execution technique for EFAs together with a calculus that mechanizes the partial evaluation process of conditions. In Sections 4 we demonstrate the symbolic interpretation combined with abstractions for nonblocking supervisory control of two industrial manufacturing systems. Finally, we conclude our work in Section 5. The proof details are referred to the appendix.

⋆ This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers University of Technology, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA).

2. PRELIMINARIES

In this section, we recall some basic definitions and concepts to be used later.

2.1 Predicate Logic

Syntax The formulas of our logic are *quantifier-free first-order logic with equality* over a countable set V of *individual variables* $x, y, \ldots$, and a *signature* set Θ consisting of n-ary *function symbols* $f \in \Theta$, where *constants* are denoted by nullary functions, *predicate symbols* $p \in \Theta$ including the *binary equality* symbol $=$, 1, 0, and the propositional connectives $\leftrightarrow, \rightarrow, \wedge, \vee, \neg$. A *term* $t \in T_\Theta(V)$ is a (well formed) expression over symbols in Θ and V. A term is called a *ground term* if it contains no variables. *Formulas* $\phi, \psi, \ldots$ are defined inductively as follows. A formula is either an *atomic formula* $p(t_1, \ldots, t_n)$ where p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, a spacial formula $\perp$ (resp. $\top$) which is always false (resp. true), or of the form $\neg\varphi$ or $\varphi \triangleright \psi$ where $\triangleright \in \{\leftrightarrow, \rightarrow, \wedge, \vee\}$ and φ, ψ are formulas.

Semantics Terms and formulas constructed over Θ and V take on meaning when interpreted over a structure called *model*. A model is a pair $\mathcal{M} = (D, \mathcal{I})$ consisting of: A finite and nonempty set D called *domain* (or universe), where we distinguish the values of an individual variable x by a nonempty set D_x; and an *interpreter* function $\mathcal{I}$ that assigns an n-ary function $f^{\mathcal{I}} : D^n \to D$ to each n-ary function symbol $f \in \Theta$ where we regard constants (nullary functions) as just elements of D, and an n-ary relation $p^{\mathcal{I}} \subseteq D^n$ to each n-ary predicate symbol $p \in \Theta$.

Fix $\mathcal{I}$ and let D be the domain of variables. We define a *valuation* map $\eta : T_\Theta(V) \to D$ on terms $T_\Theta(V)$ over variables V. A valuation is uniquely determined by its values on V, since V generates $T_\Theta(V)$. Moreover, any map $\eta : V \to D$ extends uniquely to a valuation $\eta : T_\Theta(V) \to D$ by induction. A *substitution* is a mapping $\eta : T_\Theta(V) \to T_\Theta(V)$. For a term t, $\eta(t) = t[x/\eta(x) | \forall x \in V]$ is a new term obtained by "substituting" all (free) occurrences of x_i in t with t_i $(1 \leq i \leq n)$ and we denote by ϵ the empty substitution such that $\epsilon(t) = t$. The substitution is done for all variables in t simultaneously. Furthermore, we write $\eta[x/t]$ (or $\eta[x \mapsto t]$) to denote a new substitution μ constructed from η such that $\mu(x) = t$ and $\mu(y) = \eta(y)$ for $y \neq x$. We also write $\eta[x \mapsto \epsilon]$ to denote that we drop the substitution x/t from η. In this paper, without loss of generality, we consider valuations as substitutions where a valuation substitutes all variables to their ground terms.

The *satisfaction relation* $\models$ (also called semantic entailment) is defined inductively on the structure of formulas as usual [see Gallier, 2003]. If $\eta \models \varphi$ holds, we say that φ is true (in $\mathcal{M}$) *under valuation* η, or that η *satisfies* φ (in $\mathcal{M}$). If Γ is a set of formulas, we write $\eta \models \Gamma$ if $\eta \models \varphi$ for $\varphi \in \Gamma$. If φ is true in all models, then we write $\models \varphi$ and say that φ is *valid*. Two formulas ϕ, ψ are said to be *logically equivalent*, denoted $\phi \equiv \psi$, if $\models \phi \leftrightarrow \psi$.

2.2 Proof Calculus

A proof calculus describes certain syntactic operations to be carried out on formulas. We denote by $\vdash$ a calculus containing "rules", along with some definitions that say how these rules are to be applied. The basic building blocks, to which the rules or our calculus are applied are the *sequents* of the form $\Gamma \Rightarrow \Delta$ (in the literature also denoted as $\Gamma \vdash \Delta$) where Γ and Δ contain formulas. The formulas on the left of the *sequent arrow* $\Rightarrow$ are called *antecedent* and the formulas on the right are called *succedent*. The intuitive meaning of a sequent $\phi_1, \ldots, \phi_m \Rightarrow \psi_1, \ldots, \psi_n$ is as follows: whenever all the ϕ_i of the antecedent are true, then at least one of the succedent is true, informally, $\bigwedge \phi_i \to \bigvee \psi_j$.

A *rule* (or schema) in the calculus is of the form

$$\frac{\Psi_1, \quad \Psi_2, \quad \ldots, \quad \Psi_n}{\Psi_0}$$

where $\Psi_i := \Gamma_i \Rightarrow \Delta_i$ for $0 \leq i \leq n$ denote sequents. The sequent below the line is the *conclusion* of the rule and the above sequents are its *premises*. A rule with no premises is called a *closing rule*. The meaning of the rule is that if the premises are valid, then the conclusion is also valid. However, we use it in opposite direction, that is to prove the validity of the conclusion, it suffices to prove the premises.

A sequent proof is a tree that is constructed according to a certain set of rules.

Definition 1. A *proof tree* for a formula ϕ is a finite tree where the root sequent (shown at the bottom) is annotated with $\Rightarrow \phi$; each inner node of the tree is annotated at least with a sequent; and a leaf node which may or may not be annotated with a sequent. If it is, it is the (empty) premise of one of the closing rules. A *branch* of a proof tree is a path from the root to one of the leaves. A branch is *closed* if the leaf is annotated with empty sequent. A proof tree is *closed* if all its branches are closed.

We denote by $\Psi_0 \rightsquigarrow \Psi_i$ a branch of a proof tree from the root node Ψ_0 to a node Ψ_i for some $i \in \mathcal{N} := \{0, \ldots, n\}$, where $\mathcal{N}$ is the index set of n nodes. Let $\star$ denote an empty sequent. Then, for a closed branch, we write $\Psi_0 \rightsquigarrow \Psi_i^\star$ instead of $\Psi_0 \rightsquigarrow \star$ where $\Psi_i^\star$ is the conclusion of the rule with empty premise. Further, we denote by $\pi_\phi := \{\Psi_0 \rightsquigarrow \Psi_i\}$ the set of all branches in the tree. Then, we write $\pi_\phi^\star$ when all the branches in π_ϕ are closed, or that the proof tree of ϕ is closed.

For example, consider the following proof for a formula ϕ in some calculus $\vdash$:

$$\frac{\dfrac{\Psi_3}{\Psi_1} \qquad \dfrac{\dfrac{\star}{\Psi_4} \quad \dfrac{\star}{\Psi_5}}{\Psi_2}}{\Psi_0}$$

The corresponding proof tree of the above proof has 8 nodes, $\Psi_0, \ldots, \Psi_7$, where Ψ_0 is the root node and Ψ_6, Ψ_7 denote $\star$. Further, $\pi_\phi := \{\Psi_0 \rightsquigarrow \Psi_3, \Psi_0 \rightsquigarrow \Psi_4^\star, \Psi_0 \rightsquigarrow \Psi_5^\star\}$ is the set of all branches. Clearly, π_ϕ is not closed because the branch $\Psi_0 \rightsquigarrow \Psi_3$ is not closed.

A formula ϕ is valid in proof calculus $\vdash$, denoted $\vdash \phi$, iff the proof tree for ϕ (Def. 1), is closed. Then it follows that $\vdash \phi$ iff $\pi_\phi^\star$, i.e., ϕ is valid in $\vdash$ if all branches of its proof tree are closed. If this is the case, then we simply write $\phi \vdash \pi_\phi^\star$ to denote that ϕ is valid in $\vdash$ according to a proof tree with the set of branches $\pi_\phi^\star$.

Definition 2. (Soundness). A calculus system $\vdash$ is said to be sound w.r.t. a semantics $\models$ if $\vdash \phi$ implies $\models \phi$.

In words, $\models \phi$ holds whenever $\vdash \phi$ is valid.

2.3 Extended Finite Automata

An *Extended Finite Automaton (EFA)* is a finite-state automaton whose transitions are augmented with *data*, Skoldstam et al. [2007], to symbolically represent DES. In this paper, we formulate the data flow in systems by means of predicates, henceforth *conditions*, on transitions.

EFA Syntax The behavior of DES, Wonham [2013] and Cassandras and Lafortune [2008], can be recognized by a finite-state automaton (FA) $G = \langle Q, \Sigma, \mapsto, Q^\circ, Q^m \rangle$ with the (finite) set of states Q, the (nonempty) alphabet Σ, the transition function $\delta : Q \times \Sigma \to Pwr(Q)$, where Pwr is the power set, the set of initial state Q° and a set of marked states $Q^m \subseteq Q$. In this work, marked states are irrelevant to our calculation and therefore, without loss of generality, we assume that $Q^m = Q$ and we use the tuple $\langle Q, \Sigma, \delta, Q^\circ \rangle$. We write $\delta(q, \sigma)!$ if $\delta(q, \sigma) \neq \emptyset$. The set of transitions in G is $\mapsto := \{(q, \sigma, q') \in Q \times \Sigma \times Q \mid \delta(q, \sigma)! \text{ and } q' \in \delta(q, \sigma)\}$. We sometimes write $q \overset{\sigma}{\mapsto} q'$ instead of $(q, \sigma, q') \in \mapsto$. Let Σ^* be the set of all finite strings over Σ, including the empty string ε. We write $st \in \Sigma^*$ for the concatenation of two strings $s, t \in \Sigma^*$ and $s \leq t$ when s is a *prefix* of t. Further, the notation δ is extended to strings in Σ^* in usual way [see Cassandras and Lafortune, 2008]. The closed language of the automaton G is defined by $\mathcal{L}(G) := \{u \in \Sigma^* | (\exists q^\circ \in Q^\circ; \exists p \in Q)\ q^\circ \overset{u}{\mapsto} p\}$.

Consider a set of variables V. In order to describe the data flow on transition system of EFAs, we add a second set of variables V', where each variable x in V has a corresponding (next-state) variable x' in V' over the same domain. Let $\phi_g \in \mathcal{G}_V$ denote the set of formulas over V called *guard formulas* (or just guards), and $\phi_a \in \mathcal{A}_V$ denote the set of formulas over V' and/or V called *action formulas* (or just actions). It is assumed that the actions ϕ_a are deterministic, i.e., ϕ_a is of the form $x' = t'$ for some variable x' and term t'.

Now, *conditions* $c \in \mathcal{C}_V$ are formulas of the form $c \equiv \phi_g \wedge \phi_a$. Further, we denote by $\mathrm{vars}(c)$ (resp. $\mathrm{vars}'(c)$) the set of all variables x (resp. x') appearing in c. Note that, if $V = \emptyset$ then it is assumed that $\mathcal{C}_V = \{\top, \bot\}$.

We now define extended finite automaton whose transitions are augmented with conditions.

Definition 3. (Extended Finite Automaton). An extended finite automaton is a tuple $E = \langle V, L, \Sigma, T, \ell^\circ, c^\circ \rangle$, where V is a finite set of variables, L is a finite set of locations, Σ is a nonempty finite set of events (alphabet), $T \subseteq L \times \Sigma \times \mathcal{C}_V \times L$ is the transition relation, where $\mathcal{C}_V$ is the set of conditions over $V \cup V'$, $\ell^\circ \in L$ is the initial location, and $c^\circ \in \mathcal{G}_V$ is the initial guard.

We denote by $\ell \xrightarrow{\sigma:c} \ell'$ the presence of a transition in E, from location ℓ to location ℓ' with event $\sigma \in \Sigma$ and condition $c \in \mathcal{C}_V$.

EFA Semantics An instantaneous snapshot of data flow at any moment in *executing* EFAs is determined by the values of variables. Thus our *locations* contains valuation of variables over the domain [1].

[1] Note that, in this paper we fix the interpretation over standard interpretation of arithmetic symbols. However, any other interpretations can be used as long as a proper semantics is provided.

Let η and η' be two valuations of the variables V and V', respectively, over the domain D. Then, we associate the pair (η, η') with a condition c if $(\eta, \eta') \vDash c$. If $(\eta, \eta') \vDash c$ holds, we call η and η' the present-state and the next-state valuation. For example, let x be a variable over domain $\{0, \ldots, 5\}$ and assume a transition with condition $c \equiv x > 2 \wedge x' = x + 1$. Given a present-state valuation $\eta[x/a]$, if there exists some b in the domain such that $(\eta[x/a], \eta'[x'/b]) \vDash c$, then c results in the next-state valuation $\eta'[x/b]$ whenever the transition is fired. Otherwise, if $a \leq 2$ or $a = 5$, the transition is disabled.

For a condition c and subset of variables $W \subseteq V$, let $c_{\wedge,W}$ denote a new condition

$$c_{\wedge,W} \equiv c \wedge \bigwedge_{y \in W - \mathrm{vars}'(c)} y' = y, \tag{1}$$

namely, $c_{\wedge,W}$ keeps the current value of variables in W which are not updated by c. The semantics of an EFA is given by means of an FA as follows.

Definition 4. (EFA Semantics).
Let $E = \langle V, L, \Sigma, T, \ell^\circ, c^\circ \rangle$ be an EFA. The finite-state automaton $G(E)$ of E is the tuple $\langle Q_E, \Sigma_E, \delta_E, Q_E^\circ \rangle$ with $Q_E = L \times D$, $\Sigma_E = \Sigma$, $Q_E^\circ = \{\langle \ell^\circ, \eta^\circ \rangle | \eta^\circ \vDash c^\circ \text{ for valuation } \eta^\circ\}$, and the explicit transition relation $\mapsto_E \subseteq Q_E \times \Sigma_E \times Q_E$ according to

$$\textbf{SEM}\ \frac{\ell \xrightarrow{\sigma:c} \ell', \quad (\eta, \eta') \vDash c_{\wedge,V}}{\langle \ell, \eta \rangle \overset{\sigma}{\mapsto} \langle \ell', \eta' \rangle}$$

Intuitively, states of $G(E)$ are pairs of locations ℓ and valuations η. The transitions of $G(E)$ are defined by the above inference rule, stating that whenever there exists a transition $\ell \xrightarrow{\sigma:c} \ell'$ in E and two valuations η and η' such that $(\eta, \eta') \vDash c_{\wedge,V}$, there also exists a transition $\langle \ell, \eta \rangle \overset{\sigma}{\mapsto} \langle \ell', \eta' \rangle$ in $G(E)$.

EFA Behavior and Properties The behavior of E is given by the language generated by its underlying explicit transition system $G(E)$. The *language* of E is defined as $\mathcal{L}(E) := \{u \in \Sigma^* | (\exists p \in Q_E)\ q^\circ \overset{u}{\mapsto}_E p\}$. For two EFAs E and H, we say that $E = H$ if and only if $\mathcal{L}(E) = \mathcal{L}(H)$.

EFAs, similar to ordinary finite automata, are composed by extended full synchronous composition (EFSC).

Definition 5. (EFSC).
Let $E_k = \langle V_k, L_k, \Sigma_k, T_k, \ell_k^\circ, c_k^\circ \rangle$, $k = 1, 2$, be two EFAs. The *Extended Full Synchronous Composition* of E_1 and E_2 is the tuple $E_1 \| E_2 = \langle V, L, \Sigma, T, \ell^\circ, c^\circ \rangle$, where $V = V_1 \cup V_2$, $L = L_1 \times L_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $\ell^\circ = \langle \ell_1^\circ, \ell_2^\circ \rangle$, $c^\circ = c_1^\circ \wedge c_2^\circ$, $L^m = L_1^m \times L_2^m$, and T is defined by the following rules:

$$\textbf{SYN1}\ \frac{\ell_1 \xrightarrow{\sigma:c_1} \ell_1', \quad \sigma \in (\Sigma_1 - \Sigma_2)}{\langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma:c_1} \langle \ell_1', \ell_2 \rangle}$$

$$\textbf{SYN2}\ \frac{\ell_2 \xrightarrow{\sigma:c_2} \ell_2', \quad \sigma \in (\Sigma_2 - \Sigma_1)}{\langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma:c_2} \langle \ell_1, \ell_2' \rangle}$$

$$\textbf{SYN3}\ \frac{\ell_1 \xrightarrow{\sigma:c_1} \ell_1', \quad \ell_2 \xrightarrow{\sigma:c_2} \ell_2', \quad \sigma \in (\Sigma_1 \cap \Sigma_2)}{\langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma:c_1 \wedge c_2} \langle \ell_1', \ell_2' \rangle}$$

Note that, in the rule **SYN3**, if $\nvDash c_1 \wedge c_2$ then the underlying transition in $G(E_1 || E_2)$ is not defined, see Def. 4; thus, in general, we have $\mathcal{L}(E_1 \| E_2) \neq \mathcal{L}(E_1) \| \mathcal{L}(E_2)$, where the synchronous product $\|$ for languages is defined as usual [see Wonham, 2013].

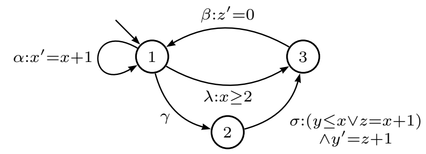

Fig. 2. EFA E in the running example.

Labeled EFA Sometimes in our computation we need to store substitutions on locations. To this end, we define labeled EFA (LEFA) as follows.

Definition 6. (Labeled EFA). A labeled EFA is an extended Kripke-structure $K = \langle V, L, \Sigma, T, \ell^\circ, c^\circ, \Lambda, \Phi\rangle$, where $\langle V, L, \Sigma, T, \ell^\circ, c^\circ\rangle$ is an EFA E, Λ is a set of location labels, and $\Phi : L \to \Lambda$ is a labeling function that associates each location to a label.

An EFA E can be transformed to a LEFA K by considering Λ be the set of all possible substitutions of variables and $\Phi(\ell) = \epsilon$ for locations ℓ. Furthermore, all concepts and notations that can be applied to EFAs are extended to LEFAs in the straightforward way. The EFSC on EFAs is simply extended to LEFAs by using $\Phi(\langle \ell_1, \ell_2\rangle)_{E_1\|E_2} = \Phi_{E_1}(\ell_1)\Phi_{E_2}(\ell_2)$, and $\Lambda_{E_1\|E_2} = \Lambda_{E_1}\Lambda_{E_2} := \{\eta_1\eta_2 | \forall\eta_1 \in \Lambda_1, \forall\eta_2 \in \Lambda_2\}$. In what follows, the letter E subscripted or not is used to represent both an EFA and its labeled EFA.

3. SYMBOLIC INTERPRETATION AND EXECUTION OF EFAS

In practice, many systems use "internal" variables. Hence, it is of great interest if we could symbolically interpret systems modeled by EFAs w.r.t. their internal variables. This can be useful for many techniques available for EFAs such as abstractions, Shoaei et al. [2012] and Mohajerani et al. [2013], and synthesis, Miremadi et al. [2008], since the interpretation process has already pre-executed the portions of system that depend on the internal variables without computing the global (explicit) model.

To this end, we introduce an *interpreter* $[\![.]\!]$ for EFAs according to the following intuition: For an EFA E with a set of variables V and a subset of (internal) variables $V_{\text{int}} \subseteq V$, the interpreter $[\![.]\!]$ starts from the initial location of E with initial substitution of variables in V_{int}; iterates over the transitions of E, and by passing each transition, it symbolically interprets and partially evaluates (executes) the condition on that transition w.r.t. the known values of variables V_{int} from the previous step, and leaves the other variables intact. Further, it stores the obtained values (ground terms) as substitutions on the locations; and when it terminates, i.e., reaching a fix point that no more condition is left on the transitions or the evaluation results in the same condition, it returns the interpreted parts of E in form of a *residual* EFA, which we denote by $[\![E]\!]_{V_{\text{int}}}$.

This section is organized as follows: First we introduce a proof calculus that formalizes the partial evaluation process. Then, we provide an algorithm that implements the interpretation process of EFAs. Throughout this section, we use EFA E in Fig. 2, using variables x, y, z with the domain $D_x := \{0, 1, 2\}$, $D_y = D_z := \{0, \ldots, 10\}$ and initial guard $c^\circ \equiv x = 0 \wedge y = 0 \wedge z = 0$, as a running example, for which we want to interpret E w.r.t. $V_{\text{int}} = \{x\}$.

Partial Evaluation For a condition c and a (present-state) substitution η, we mechanize the steps in the partial evaluation of c w.r.t. η by a proof calculus $\vdash_{V_{\text{int}}}$ according to the rules in Fig. 1. The sequents of $\vdash_{V_{\text{int}}}$ are of the form $\Gamma \Rightarrow \langle\psi, \ldots\rangle\Delta$. The element $\Delta := (\eta, c_{\wedge,V_{\text{int}}})$, which we call configuration, is a pair of substitution η together with the formula $c_{\wedge,V_{\text{int}}}$, as in Eq. (1). $\langle\psi, \ldots\rangle$ is a placeholder for formulas, which we will process, and Γ contains the processed formulas. The informal semantics of sequents $\phi_0, \ldots, \phi_m \Rightarrow \langle\psi_0, \ldots, \psi_n\rangle(\eta, c_{\wedge,V_{\text{int}}})$ corresponds to the formula

$$\bigwedge_{0\leq i\leq m} \phi_i \to \bigwedge_{0\leq j\leq n} \psi_j \wedge \dot{\eta} \wedge c_{\wedge,V_{\text{int}}}, \tag{2}$$

where $\dot{\eta} := \bigwedge_{x\in V} x = \eta(x)$ and in particular $\dot{\epsilon} := \top$.

The intuition behind the rules in Fig. 1 is the following: Rule **1** states that for a root sequent $\Rightarrow (\eta, c_{\wedge,V_{\text{int}}})$ with initial configuration $(\eta, c_{\wedge,V_{\text{int}}})$, it constructs a new sequent of the form $\Rightarrow \langle\eta(c_{\wedge,V_{\text{int}}})\rangle(\epsilon, \top)$, where $\eta(c_{\wedge,V_{\text{int}}})$ is the application of substitution η on $c_{\wedge,V_{\text{int}}}$. There are now other rules that may be applied.
Rule **2** converts conjunctions to clauses of formulas and Rule **3** converts disjunctions to premises, hence our proof branches. Rule **4** is a closing rule which is applied whenever the placeholder is exhausted.
Rule **5** takes formulas of the form $x = t$ from the placeholder; substitutes any occurrence of variable x in all formulas with term t; conjuncts it to c'; and finally moves it to the antecedent of the sequent.
Rule **6** deals with next-state variables, x'. It checks for the formulas of the form $x' = t'$ in the placeholder and if x is a variable in V_{int} and t' is in the domain of x, it extends the substitution η' by $\eta'[x' \mapsto t']$. Rule **7** takes any formula in the placeholder but instead it just conjuncts them to c'.

Note that, first applying Rule **5** and then **6** results in a "stronger" configuration since we first propagate the term t to all formulas in the current sequent and then process the other formulas. Similarly, applying Rule **5** and/or **6** first, until the placeholder is exhausted with $x = t$ and $x' = t'$, and then **7** also results in a stronger configuration. Therefore, in such cases, we always apply these rules in a way that the end result is the strongest configuration, namely in the following order: **5**, **6**, and then **7**.

Note also that, in every step of the proof, it is assumed that the formulas are presented in their simplified form, e.g., $\neg\neg\phi \equiv \phi$, $\top \vee \phi \equiv \top$, $\bot \wedge \phi \equiv \bot$, etc. For other simplification rules we refer to Gallier [2003].

We now clarify the above rules by the following example. Consider the condition $c \equiv (y \leq x \vee z = x+1) \wedge y' = z+1$ on σ-transition of EFA E (see Fig. 2) and assume $\eta := [x/1]$ is the current substitution at location 2. Then, the proof of the initial configuration $([x/1], (y \leq x \vee z = x + 1) \wedge y' = z + 1 \wedge x' = x)$ in $\vdash_{V_{\text{int}}}$ is

$$\begin{gathered}
\mathbf{5}\;\dfrac{\mathbf{6}\;\dfrac{\mathbf{7}\;\dfrac{\mathbf{4}\;\dfrac{\star}{z{=}2,x'{=}1,y'{=}3 \Longrightarrow \langle\,\rangle([x'/1],z{=}2\wedge y'{=}3)\quad[\Psi_{11}]}}{z{=}2,x'{=}1 \Longrightarrow \langle y'{=}3\rangle(\epsilon[x'{\mapsto}1],z{=}2)}}{z{=}2 \Longrightarrow \langle [z/2]y'{=}z{+}1,[z/2]x'{=}1\rangle(\epsilon,\top\wedge z{=}2)}}{\Longrightarrow \langle z{=}2,y'{=}z{+}1,x'{=}1\rangle(\epsilon,\top)\quad[\Psi_5]} \\[2ex]
\mathbf{6}\;\dfrac{\mathbf{7,7}\;\dfrac{\mathbf{4}\;\dfrac{\star}{x'{=}1,y{\leq}1,y'{=}z{+}1 \Longrightarrow \langle\,\rangle([x'/1],y{\leq}1\wedge y'{=}z{+}1)\quad[\Psi_{10}]}}{x'{=}1 \Longrightarrow \langle y{\leq}1,y'{=}z{+}1\rangle(\epsilon[x'{\mapsto}1],\top)}}{\Longrightarrow \langle y{\leq}1,y'{=}z{+}1,x'{=}1\rangle(\epsilon,\top)\quad[\Psi_4]} \\[2ex]
\mathbf{1}\;\dfrac{\mathbf{2,2}\;\dfrac{\mathbf{3}\;\dfrac{\Psi_4 \qquad\qquad \Psi_5}{\Longrightarrow \langle (y{\leq}1\vee z{=}2),y'{=}z{+}1,x'{=}1\rangle(\epsilon,\top)\quad[\Psi_3]}}{\Longrightarrow \langle ([x/1](y{\leq}1\vee z{=}2)\wedge y'{=}z{+}1\wedge x'{=}1)\rangle(\epsilon,\top)}}{\Longrightarrow ([x/1],(y{\leq}x\vee z{=}x{+}1)\wedge y'{=}z{+}1\wedge x'{=}x)\quad[\Psi_0]}
\end{gathered} \tag{3}$$

$$\mathbf{1}\frac{\Rightarrow \langle \eta(c_{\wedge,V_{\rm int}})\rangle(\epsilon,\top)}{\Rightarrow (\eta, c_{\wedge,V_{\rm int}})} \quad \mathbf{2}\frac{\Gamma \Rightarrow \langle \psi_1, \psi_2, \cdots\rangle\Delta}{\Gamma \Rightarrow \langle \psi_1 \wedge \psi_2, \cdots\rangle\Delta} \quad \mathbf{3}\frac{\Gamma \Rightarrow \langle \psi_1, \cdots\rangle\Delta \quad \Gamma \Rightarrow \langle \psi_2, \cdots\rangle\Delta}{\Gamma \Rightarrow \langle \psi_1 \vee \psi_2, \cdots\rangle\Delta} \quad \mathbf{4}\frac{}{\Gamma \Rightarrow \langle\,\rangle\Delta}$$

$$\mathbf{5}\frac{\Gamma, x=t, \phi[x/t] \Rightarrow \langle \psi[x/t], \cdots\rangle(\eta', c' \wedge x=t)}{\Gamma, \phi \Rightarrow \langle x=t, \psi, \cdots\rangle(\eta', c')} \quad \mathbf{6}\frac{\Gamma, x'=t' \Rightarrow \langle \cdots\rangle(\eta'[x' \mapsto t'], c')}{\Gamma \Rightarrow \langle x'=t', \cdots\rangle(\eta', c')}\ \text{if } x \in V_{\rm int} \text{ and } t' \in D_x \quad \mathbf{7}\frac{\Gamma, \psi \Rightarrow \langle \cdots\rangle(\eta', c' \wedge \psi)}{\Gamma \Rightarrow \langle \psi, \cdots\rangle(\eta', c')}$$

Fig. 1. Proof rules of calculus $\vdash_{V_{\rm int}}$ for the partial evaluation of input configuration $(\eta, c_{\wedge,V_{\rm int}})$.

In (3), Ψ_0 denotes the root sequent and $\star$ denotes the empty premise. Note that, in the sequent Ψ_3, the proof branches to two sequents, Ψ_4 and Ψ_5, because of the disjunction $y \le 1 \vee z = 2$.

Theorem 1. (Soundness). If a sequent $\Gamma \Rightarrow \Delta$ is derivable in the calculus $\vdash_{V_{\rm int}}$ according to the rules in Fig. 1, then it is logically valid according to Eq. (2).

The soundness of the calculus $\vdash_{V_{\rm int}}$ provides the validity of the rules in Fig. 1. Since our proof branches only because of the disjunction in the formulas (see Rule **3**), we prove that the disjunction of all configurations whose sequent is the conclusion of the closing rule, i.e. Rule **4**, is equivalent to the initial configuration. For a sequent $\Psi := \Gamma \Rightarrow \Delta$, let $\mathit{succedent}(\Psi) = \Delta$ be a function that retrieves the succedent of Ψ. With abuse of notation, let $\mathit{succedent}^\star(\Psi_0 \rightsquigarrow \Psi_i^\star) := \mathit{succedent}(\Psi_i^\star)$ and $\mathit{succedent}^\star(\Psi_0 \rightsquigarrow \Psi_j) := \emptyset$ for $i, j \in \mathcal{N}$. Further, let $\mathit{succedent}^\star(\pi_\phi) := \{\mathit{succedent}^\star(\beta) | \text{for all branches } \beta \in \pi_\phi\}$.

Proposition 1. Let $(\eta, c_{\wedge,V_{\rm int}})$ be a configuration. Then,

$$(\eta, c_{\wedge,V_{\rm int}}) \vdash_{V_{\rm int}} \pi^\star_{(\eta,c_{\wedge,V_{\rm int}})} \implies$$
$$(\eta, c_{\wedge,V_{\rm int}}) \vDash \bigvee \mathit{succedent}^\star(\pi^\star_{(\eta,c_{\wedge,V_{\rm int}})}),$$

where $\pi^\star_{(\eta,c_{\wedge,V_{\rm int}})}$ is the proof tree (see Def. 1) of the root sequent $\Rightarrow (\eta, c_{\wedge,V_{\rm int}})$ in $\vdash_{V_{\rm int}}$.

The proof of Prop. 1 is by an induction on the structure of the proof tree of $(\eta, c_{\wedge,V_{\rm int}})$. That is, we can inductively derive the validity of $(\eta, c_{\wedge,V_{\rm int}})$ by following the leafs of the proof tree to the root sequent $\Rightarrow (\eta, c_{\wedge,V_{\rm int}})$. Hence, the proof is left out.

For example, consider the proof in (3). Since all branches of (3) are closed, we have that $\phi \vdash_{V_{\rm int}} \pi^\star_{(\eta,c_{\wedge,V_{\rm int}})} := \{\Psi_0 \rightsquigarrow \Psi^\star_{10}, \Psi_0 \rightsquigarrow \Psi^\star_{11}\}$. Consequently, it follows that

$$([x/1], (y \le x \vee z = x+1) \wedge y' = z+1 \wedge x' = x) \overset{\text{def}}{\iff}$$
$$(y \le 1 \wedge y' = z+1 \wedge x' = 1) \vee (z = 2 \wedge y' = 3 \wedge x' = 1) \vDash$$
$$\bigvee \{([x'/1], y \le 1 \wedge y' = z+1), ([x'/1], z = 2 \wedge y' = 3)\} \overset{\text{def}}{\iff}$$
$$(y \le 1 \wedge y' = z+1 \wedge x' = 1) \vee (z = 2 \wedge y' = 3 \wedge x' = 1),$$

as expected by the result of Prop. 1.

For a condition c and a present-state substitution η, we use $\eta^\star$ and $c^\star$ in $(\eta^\star, c^\star) \in \mathit{succedent}^\star(\pi^\star_{(\eta,c_{\wedge,V_{\rm int}})})$ as respectively the next-state substitutions and the residual conditions of partial evaluation of c w.r.t. η.

Interpretation Algorithm Taking a labeled EFA E and a subset of variables $V_{\rm int}$, Algorithm 1 implements the interpretation process $[\![E]\!]_{V_{\rm int}}$. From an abstract view, the algorithm collects the reachable transitions with residual conditions starting from the initial location ℓ° with initial substitution $\eta_{\rm int}$ of variables in $V_{\rm int}$.

Algorithm 1 (Symbolic Interpretation of EFAs)

Require: A labeled EFA $E = \langle V, L, \Sigma, T, \ell^\circ, c^\circ, \Lambda, \Phi\rangle$ and a set of variables $V_{\rm int} \subseteq V$.

procedure $[\![E]\!]_{V_{\rm int}}$
  Let $\eta_{\rm int} := \epsilon$, $L^\star := \emptyset$, and $T^\star := \emptyset$;
  Let η° be a valuation s.t. $\eta^\circ \vDash c^\circ$;
  Let S be a stack of configurations;
  $(\forall x \in V_{\rm int})\ \eta_{\rm int} := \eta_{\rm int}[x \mapsto \eta^\circ(x)]$; $S \leftarrow (\ell^\circ, \eta_{\rm int})$;
  repeat
    $(\ell, \eta) := S.\mathit{pop}()$;
    Let $\langle \ell, \eta\rangle$ be a new location;
    if $\langle \ell, \eta\rangle \in L^\star$ **then continue**
    $\Phi(\langle \ell, \eta\rangle) := \eta$; $L^\star \leftarrow \langle \ell, \eta\rangle$;
    $T' := \{(\ell, \sigma, c, \ell') \in T | \forall \sigma \in \Sigma, \forall \ell' \in L\}$;
    for all $(\ell, \sigma, c, \ell') \in T'$ **do**
      for all $(\eta^\star, c^\star) \in \vdash_{V_{\rm int}}(\eta, c_{\wedge,V_{\rm int}})$ **do**
        $\Phi(\langle \ell', \eta'^\star\rangle) := \Phi(\ell')\eta^\star$; $L^\star \leftarrow \langle \ell', \eta'^\star\rangle$;
        $T^\star \leftarrow (\langle \ell, \eta\rangle, \sigma, c^\star, \langle \ell', \eta'^\star\rangle)$;
        $S.\mathit{push}((\ell', \eta'^\star))$;
      end for
    end for
  until $S \neq \emptyset$
  return $E^\star = \langle V, L^\star, \Sigma, T^\star, \langle \ell^\circ, \eta_{\rm int}\rangle, c^\circ, \Lambda, \Phi\rangle$
end procedure

procedure $\vdash_{V_{\rm int}}(\eta, c_{\wedge,V_{\rm int}})$
  $R := \emptyset$;
  if $(\eta, c_{\wedge,V_{\rm int}}) \vdash_{V_{\rm int}} \pi^\star_{(\eta,c_{\wedge,V_{\rm int}})}$ as in Fig. 1 **then**
    for all $(\eta^\star, c^\star) \in \mathit{succedent}^\star(\pi^\star_{(\eta,c_{\wedge,V_{\rm int}})})$ **do**
      if $c^\star \equiv \bot$ **then continue**
      $(\forall x \in V_{\rm int})\ \eta^\star = (\eta^\star[x/\eta^\star(x')])[x' \mapsto \epsilon]$;
      $R \leftarrow (\eta^\star, c^\star)$;
    end for
  end if
  return R
end procedure

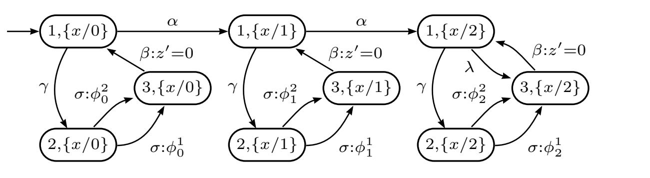

Fig. 3. The residual EFA $[\![E]\!]_x$. Here, ϕ_i^1 and ϕ_i^2 are respectively equivalent to the formula $y \le i \wedge y' = z+i$ and $z = i+1 \wedge y' = i+2$ for integer $i = 0, 1, 2$.

Let us illustrate Algorithm 1 by applying it to the EFA E in Fig. 2 for $V_{\rm int} := \{x\}$. The stack S is initialized by $(1, [x/0])$. In the first iteration, the configuration $(1, [x/0])$ is removed from the stack and a new location $\langle 1, [x/0]\rangle$ with $\Phi(\langle 1, [x/0]\rangle) = [x/0]$ is added to the set $L^\star$. Then, for outgoing transitions of location 1, i.e., $1 \xrightarrow{\alpha: x'=x+1} 1$

and $1 \xrightarrow{\gamma} 2$, Algorithm 1 obtains the residual conditions and next-state values of $x' = x + 1$ and $\top$ w.r.t. $[x/0]$ by calling $\vdash_{V_{\text{int}}}(.,.)$. The procedure $\vdash_{V_{\text{int}}}(.,.)$ implements the partial evaluation process according to the rules in Fig. 1. Consequently, $\vdash_{V_{\text{int}}}([x/0], x' = x + 1)$ and $\vdash_{V_{\text{int}}}([x/0], \top)$ return the sets $\{([x/1], \top)\}$ and $\{([x/0], \top)\}$, respectively. Note that, in line 27, the variable x' is replaced by x. Then, for the residual pair $([x/1], \top)$, Algorithm 1 creates a new location $\langle 1, [x/1]\rangle$ labeled by $\Phi(\langle 1, [x/1]\rangle) = [x/1]$ and adds it to the set $L^\star$ and stack S. Note that $\langle 1, [x/0]\rangle \neq \langle 1, [x/1]\rangle$. Further, a new transition $\langle 1, [x/0]\rangle \xrightarrow{\alpha} \langle 1, [x/1]\rangle$ is added to the (residual) transition set $T^\star$. Similarly, for $([x/0], \top)$, we have the location $\langle 2, [x/0]\rangle$ with $\Phi(\langle 2, [x/0]\rangle) = [x/0]$ and the transition $\langle 1, [x/0]\rangle \xrightarrow{\gamma} \langle 2, [x/0]\rangle$. Algorithm 1, iterates over the pairs in stack S, and when S is empty, it terminates and returns the residual EFA $[\![E]\!]_x$, see Fig. 3.

Proposition 2. (Algorithm 1 Correctness).
Let E be a labeled EFA with set of variables V over finite domain D. Let $V_{\text{int}} \subseteq V$. Algorithm 1 terminates; and when it terminates it holds that $[\![[\![E]\!]_{V_{\text{int}}}]\!]_{V-V_{\text{int}}} = [\![E]\!]_V$.

That is, interpretation of E w.r.t. V_{int} and then with the remaining variables $V - V_{\text{int}}$, results in the same EFA as interpret of E w.r.t. V returns. Since the domain of variables, D, is finite, it is straightforward to show that Algorithm 1 terminates.

A realistic DES is often composed from a group of EFA components. Let $\mathbf{DES} := \{E_1, \ldots, E_n\}$ be a discrete-event system consisting of n EFA components over the respective alphabet $\Sigma_1, \ldots, \Sigma_n$ and variables $V_1, \ldots, V_n$, for which we want to symbolically interpret it. In $\mathbf{DES}$ some of the variables might be internally updated by only one component. Let V_i^a denote the set of such variables in E_i, i.e., $V_i^a := \{x \in V_i | x \in (\text{vars}'(\mathcal{C}_{V_i}) - \bigcup_{j=1, j\neq i}^{n} \text{vars}'(\mathcal{C}_{V_j}))\}$.

Theorem 2. Let E_i $(i = 1, 2)$ be two EFAs in $\mathbf{DES}$ over V_i with $V_i^a, V_{i,\text{int}} \subseteq V_i$. Consider $E := E_1 \| E_2$ and let $V_{\text{int}} := V_{1,\text{int}} \cup V_{2,\text{int}}$. If $V_{i,\text{int}} \subseteq V_i^a$, then it holds that $[\![E_1 \| E_2]\!]_{V_{\text{int}}} = [\![E_1]\!]_{V_{1,\text{int}}} \| [\![E_2]\!]_{V_{2,\text{int}}}$.

In a straightforward way, we can further extend Theorem 2 to all components in $\mathbf{DES}$. This implies that the interpretation of each component w.r.t. to their internal variables will not change the global behavior of the system.

4. APPLICATION OF SYMBOLIC INTERPRETATION

In this section, we discuss an application of symbolic interpretation in the nonblocking supervisory control of the cluster tool example in Su et al. [2010] modeled by EFAs. The cluster tool is an integrated manufacturing system used for wafer processing. It consists of one entering load lock (L_{in}) and one exit load lock (L_{out}), nine chambers (C_{ij}, where, for $i = 1, 2, 3$, we have $j = 1, 2$, and for $i = 4$, we have $j = 1, 2, 3$), three one-slot buffers (B_k for $k = 1, 2, 3$), and four transportation robots (R_i for $i = 1, 2, 3, 4$), see Fig. 4.

Fig. 5 illustrates the EFA models of R_1 and buffer B_1. In this, the Boolean variables R_{1i}, C_{11}, and C_{12} for $i = 1, \ldots, 4$, models the robot and chambers status, and the Boolean variable B_1 representing the buffers capacity of one. Also, in Fig. 5, the desired routing specification are represented by guard formulas on the EFAs. We refer to Shoaei and Lennartson [2014] for the complete EFA models of the system.

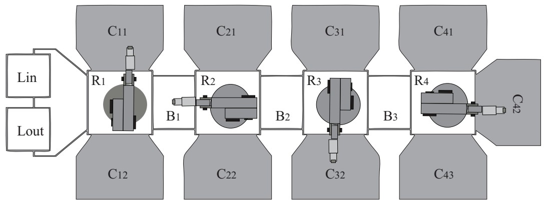

Fig. 4. Structure of Cluster Tool example.

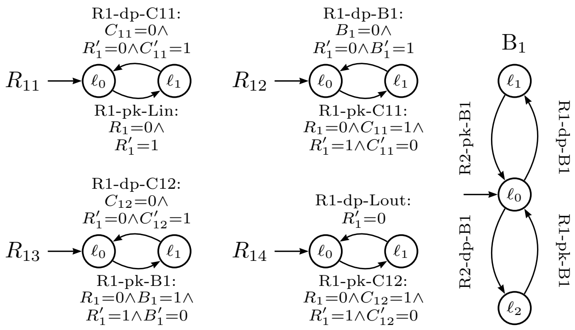

Fig. 5. EFA models of robot R_1 with the routing specification as guard formulas and buffer B_1 specification.

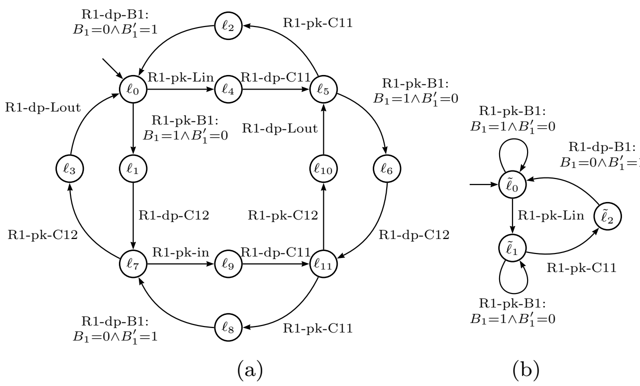

Fig. 6. (a) Residual EFA $\mathbf{R}_1^\star := [\![\mathbf{R}_1]\!]_{\{R_i, C_{11}, C_{12}\}}$ and (b) the abstracted EFA $\tilde{\mathbf{R}}_1^\star$.

We now apply the abstraction techniques in Shoaei et al. [2012] to this system. However, because of the structure of the system, none of the events can be abstracted since each of them has action formulas. To end this problem, first we synchronize the robots models. Then, we apply the proposed symbolic interpretation on these synchronized models where we obtain the residual EFAs, in which the variables R_i and C_{ij} are interpreted while the variable B_k remains. The residual EFA $[\![\mathbf{R}_1]\!]_{\{R_i, C_{11}, C_{12}\}}$ for the synchronized model $\mathbf{R}_1 := \| R_{1i}$ $(i = 1, \ldots, 4)$ is shown in Fig. 6(a), where for brevity we drop the variables value in each location.

Now for the residual models we are able to apply the abstraction since the internal variables are interpreted. For EFA $\mathbf{R}_1^\star$, the abstracted model $\tilde{\mathbf{R}}_1^\star$ is depicted in Fig. 6(b), where $\tilde{\ell}_0 = \{\ell_0, \ell_1, \ell_3, \ell_7\}$, $\tilde{\ell}_1 = \{\ell_4, \ell_5, \ell_6, \ell_8, \ell_{10}, \ell_{11}\}$, $\tilde{\ell}_2 = \{\ell_2, \ell_9\}$ denote the equivalent class of locations. Finally, we use the Supremica to synthesize the controller for the abstracted residual models. The nonblocking supervisor to achieve a nonblocking control based on the original models has 237 648 states, while the supervisor using the reduced models has 9 682 states.

5. CONCLUSION

In this paper we introduce a symbolic interpretation technique for EFAs. The interpreter symbolically interprets and executes EFAs w.r.t. their internal variables and returns the remaining parts as residual models. Furthermore, for the purpose of supervisory control, we provide sufficient conditions to guarantee that the behavior of the residual system and the original system is left unchanged, hence resulting in nonblocking supervisory control to the entire system by using the residual models. Finally, we demonstrate the effectiveness and necessity of the proposed technique combined with abstractions for nonblocking supervisory control of an industrial manufacturing system.

APPENDIX

Proof of Theorem 1

We need to show that the rules in Fig. 1 preserve the validity of the sequent $\Gamma \Rightarrow \Delta$. Recall the semantics of sequent in Eq. (2), where $\Gamma = \emptyset$ defined as $\top$. Also, it is straightforward to show that $\eta(c)$ is logically equivalent to $\dot{\eta} \wedge c$. We sketch the proof as follows. (Rule **1**) In the conclusion, we have $\top \to \dot{\eta} \wedge \hat{c} \equiv \dot{\eta} \wedge \hat{c}$, and in the premise we have $\top \to \eta(\hat{c}) \wedge \top \equiv \eta(\hat{c})$. Clearly, $\dot{\eta} \wedge \hat{c} \vDash \eta(\hat{c})$. (Rule **2**) Conjunctions in $\langle \phi_1 \wedge \phi_2, \dots \rangle$ are inductively transformed to clauses separated by comma. Hence, straightforwardly the validity is preserved. (Rule **3**) Disjunctions $\langle \phi_1 \vee \phi_2, \dots \rangle$ branch the proof. Again, it is straightforward to show that the validity is preserved. (Rule **4**) This is the closing rule which is valid by the hypotheses. (Rule **5**) For any formula of the form $x = t$, the term t is propagated (by applying the substitution $[x/t]$) to all formulas in the sequent. After that, $x = t$ is added to c'. The preservation of validity is straightforward. (Rule **6**) For any formula of the form $x' = t'$ s.t. $x \in V_{\text{int}}$ and $t' \in D_x$, the substitution η' is extended by $\eta'[x' \mapsto t']$. In this case, the preservation of validity is also straightforward. (Rule **7**) This rule is a special case of rules **5** and **6** which takes any formula in the placeholder and conjuncts to c'.

Proof of Proposition 2

We prove by induction on $w \in \Sigma^*$ in $G(E)$. Let $V = \{x_1, \dots, x_m, x_{m+1}, \dots, x_n\}$ be the set of variables in E. Assume $V_{\text{int}} = \{x_1, \dots, x_m\}$ and let $V'_{\text{int}} = V - V_{\text{int}} := \{x_{m+1}, \dots, x_n\}$. (Base) $w = \varepsilon$. By the hypothesis, ℓ° is the same. On lhs, $[\![E]\!]_{V_{\text{int}}}$ gives $\eta_l = [x_1/t_1, \dots, x_m/t_m]$ for $t_i \in D$ $(i = 1, \dots, m)$ and then by $[\![[\![E]\!]_{V_{\text{int}}}]\!]_{V'_{\text{int}}}$ we get $\eta^\circ_{l'} = \eta^\circ_l[x_{m+1}/t_{m+1}, \dots, x_n/t_n]$ for $t_j \in D$ $(j = m+1, \dots, n)$. On the rhs, $[\![E]\!]_V$ gives $\eta^\circ_r = [x_1/t_1, \dots, x_n/t_n]$. Hence, $\eta^\circ_l \eta^\circ_{l'} = \eta^\circ_r$ as expected. (Induction) $w = w'\sigma$ for some $w' \in \Sigma^*_E$ and $\sigma \in \Sigma$. We have $\langle \ell^\circ, \eta^\circ \rangle \xrightarrow{w'} \langle \ell, \eta \rangle \xrightarrow{\sigma} \langle \ell', \eta' \rangle$ for some valuations η and η'. The proof of this part is similar to the (Base) for $\langle \ell', \eta' \rangle$, hence it is left out.

Proof of Theorem 2

We sketch the proof as follows. Let $i = 1, 2$ be an index and let x_i be two distinct variables that appear in E_i. Further, consider $t_i := \ell_i \xrightarrow{\sigma:\phi_i} \ell'_i$ s.t. $t_i \in E_i$, $\sigma \in \Sigma_1 \cup \Sigma_2$, and $x_1, x_2 \in \text{vars}(\phi_i) \cup \text{vars}'(\phi_i)$, and let $t_{12} \in E_1 \| E_2$, where $t_{12} := \langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma:\phi_1 \wedge \phi_2} \langle \ell'_1, \ell'_2 \rangle$. Then the proof follows on considering the different cases of x_i membership in sets V^a_i. We write $\ell \xrightarrow{\sigma:c} \ell' \in E$ to the existence of the transition $\ell \xrightarrow{\sigma:c} \ell'$ in transition set of E. ($\mathbf{x_i \in V^a_i}$) : Then $\tilde{t}_i := \tilde{\ell}_i \xrightarrow{\sigma:\phi_i[x_i/t_i]} \tilde{\ell}'_i \in [\![E_i]\!]_{x_i}$ and $\tilde{t}_{12} := \widetilde{\langle \ell_1, \ell_2 \rangle} \xrightarrow{\sigma:\phi_1 \wedge \phi_2[x_1/t_1, x_2/t_2]} \widetilde{\langle \ell'_1, \ell'_2 \rangle} \in [\![E_1 \| E_1]\!]_{\{x_1, x_2\}}$. Clearly, by Def. 5, $\{\tilde{t}_1\} \| \{\tilde{t}_2\}$ result in a similar transition as $\tilde{t}_{12}$ up to renaming the locations, i.e., for the transition labels we have $\{\sigma : \phi_1[x_1/t_1]\} \| \{\sigma : \phi_2[x_2/t_2]\} = \sigma : \phi_1 \wedge \phi_2[x_1/t_1, x_2/t_2]$. ($\mathbf{x_1 \in V^a_1, x_2 \notin V^a_1 \cup V^a_2}$) : Then $\tilde{t}_1 := \tilde{\ell}_1 \xrightarrow{\sigma:\phi_1[x_1/t_1]} \tilde{\ell}'_1 \in [\![E_1]\!]_{x_1}$, $\tilde{t}_2 := \tilde{\ell}_2 \xrightarrow{\sigma:\phi_2} \tilde{\ell}'_2 \in [\![E_2]\!]_\emptyset$, and $\tilde{t}_{12} := \widetilde{\langle \ell_1, \ell_2 \rangle} \xrightarrow{\sigma:\phi_1 \wedge \phi_2[x_1/t_1]} \widetilde{\langle \ell'_1, \ell'_2 \rangle} \in [\![E_1 \| E_1]\!]_{\{x_1\}}$. Again, by Def. 5, $\{\tilde{t}_1\} \| \{\tilde{t}_2\}$ result in a similar transition as $\tilde{t}_{12}$ up to renaming the locations, namely, $\{\sigma : \phi_1[x_1/t_1]\} \| \{\sigma : \phi_2\} = \sigma : \phi_1 \wedge \phi_2[x_1/t_1]$. ($\mathbf{x_i \notin V^a_i}$) : Then $\tilde{t}_i \in [\![E_i]\!]_\emptyset$ and $\tilde{t}_{12} \in [\![E_1 \| E_1]\!]_\emptyset$ are the same as t_1, t_2, and t_{12}, respectively, which by the hypothesis assumptions are similar. Other cases of x_i membership in V^a_i and for different conditions ϕ_i can be shown similarly, the proof of which is left out.

REFERENCES

Christos G Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer US, Boston, MA, 2nd edition, 2008.

Jean H. Gallier. *Logic For Computer Science*. Addison-Wesley Wokingham, 2003.

John Hatcliff. An Introduction to Online and Offline Partial Evaluation Using a Simple Flowchart Language. In John Hatcliff, Torben ÆMogensen, and Peter Thiemann, editors, *Partial Eval.*, volume 1706 of *Lecture Notes in Computer Science*, pages 20–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Peter Sestoft, 1993.

Sajed Miremadi, Knut Akesson, and Bengt Lennartson. Extraction and representation of a supervisor using guards in extended finite automata. In *9th Int. Work. Discret. Event Syst.*, pages 193–199, May 2008.

Sajed Miremadi, Bengt Lennartson, and Knut Akesson. A BDD-Based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Trans. Control Syst. Technol.*, 20(6):1421–1435, November 2012.

Sahar Mohajerani, Robi Malik, and Martin Fabian. Compositional nonblocking verification for extended finite-state automata using partial unfolding. In *IEEE Int. Conf. Autom. Sci. Eng.*, pages 930–935, August 2013.

Mohammad Reza Shoaei and Bengt Lennartson. On the Computation of Natural Observers for Extended Finite Automata. In *19th World Congr. Int. Fed. Autom. Control*, page 8, 2014.

Mohammad Reza Shoaei, Lei Feng, and Bengt Lennartson. Supervisory control of extended finite automata using transition projection. In *51st IEEE Conf. Decis. Control*, pages 7259–7266, December 2012.

Markus Skoldstam, Knut Akesson, and Martin Fabian. Modeling of discrete event systems using finite automata with variables. In *46th IEEE Conf. Decis. Control*, pages 3387–3392, 2007.

Rong Su, Jan H. van Schuppen, and Jacobus E. Rooda. Aggregative Synthesis of Distributed Supervisors Based on Automaton Abstraction. *IEEE Trans. Automat. Contr.*, 55(7):1627–1640, July 2010.

W. M. Wonham. *Supervisory Control of Discrete Event Systems*. Department of Electrical and Computer Engineering, University of Toronto, 2002-2012, 2013.