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). Casandras and Laforet [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 the interpreter 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 Hatchcliff [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 Winguist 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, . . . , and a signature set Θ consisting of n-ary function symbols f ∈ Θ, where constants are denoted by nullary functions, predicate symbols p ∈ Θ including the binary equality symbol =, 1, 0, and the propositional connectives ¬, →, ∧, ∨, ¬. A term t ∈ TΘ(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 ϕ, ψ, . . . are defined inductively as follows. A formula is either an atomic formula p(t1, . . . , tn) where p is an n-ary predicate symbol and t1, . . . , tn are terms, a special formula ⊥ (resp. ⊤) which is always false (resp. true), or of the form ¬ϕ or ϕ ∨ ψ where ⊢ is valid.

Semantics Terms and formulas constructed over Θ and V take on meaning when interpreted over a structure called model. A model is a pair M = (D, 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 Dx; and an interpreter function I that assigns an n-ary function f ∈ Dn → D to each n-ary function symbol f ∈ Θ where we regard constants (nullary functions) as just elements of D, and an n-ary relation R ⊆ Dn to each n-ary predicate symbol p ∈ Θ.

Fix I and let D be the domain of variables. We define a valuation map η : TΘ(V) → D on terms TΘ(V) over variables V. A valuation is uniquely determined by its values on V, since V generates TΘ(V). Moreover, any map η : V → D extends uniquely to a valuation η : TΘ(V) → D by induction. A substitution is a mapping η : TΘ(V) → TΘ(V). For a term t, η(t) = t[x/η(x)]∀x ∈ V is a new term obtained by “substituting” all (free) occurrences of xi in t with ti (1 ≤ i ≤ n) and we denote by e the empty substitution such that e(t) = t. The substitution is done for all variables in t simultaneously. Furthermore, we write η(x/f) (or η[x → t]) to denote a new substitution μ constructed from η such that μ(x) = t and μ(y) = η(y) for y ̸= x. We also write η[x → e] to denote that we drop the substitution x/y 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 ⊨ (also called semantic entailment) is defined inductively on the structure of formulas as usual [see Gallier, 2003]. If η ⊨ ϕ holds, we say that ϕ is true (in M) under valuation η, or that η satisfies ϕ (in M). If Γ is a set of formulas, we write η ⊨ Γ if η ⊨ ϕ for ϕ ∈ Γ. If ϕ is true in all models, then we write ⊨ ϕ and say that ϕ is valid. Two formulas ϕ, ψ are said to be logically equivalent, denoted ϕ ≡ ψ, if η ⊨ ϕ ↔ ψ.

2.2 Proof Calculus

A proof calculus describes certain syntactic operations to be carried out on formulas. We denote by ⊢ 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 Γ ⊨ Δ (in the literature also denoted as Γ ⊢ Δ) where Γ and Δ contain formulas. The formulas on the left of the sequent arrow ⊨ are called antecedent and the formulas on the right are called succedent. The intuitive meaning of a sequent ϕ1, . . . , ϕm ⊨ ψ1, . . . , ψn is as follows: whenever all the ϕi of the antecedent are true, then at least one of the succedent is true, informally, ∨ϕ1 → ψj.

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

Ψ1, Ψ2, . . . , Ψn

where Ψi := Γi ⊨ Δi for 0 ≤ i ≤ 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 ϕ; 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 Ψ0 ⊨ ψ, a branch of a proof tree from the root node Ψ0 to a node Ψ1 for some i ∈ N := {0, . . . , n}, where N is the index set of n nodes. Let * denote an empty sequent. Then, for a closed branch, we write Ψ0 ⊨ Ψ1 instead of Ψ0 ⊨ * where Ψ1 is the conclusion of the rule with empty premise. Further, we denote by σϕ := {Ψ0 ⊨ Ψ1} the set of all branches in the tree. Then, we write σϕ when all the branches in σϕ are closed, or that the proof tree of ϕ is closed.

For example, consider the following proof for a formula ϕ in some calculus ⊢:

Ψ3
Ψ4
Ψ5
Ψ0

The corresponding proof tree of the above proof has 8 nodes, Ψ0, . . . , Ψ7, where Ψ0 is the root node and Ψ6, Ψ7 denote *. Further, πϕ := {Ψ0 ⊨ Ψ3, Ψ0 ⊨ Ψ4, Ψ0 ⊨ Ψ5} is the set of all branches. Clearly, πϕ is not closed because the branch Ψ0 ⊨ Ψ3 is not closed.

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

Definition 2. (Soundness) A calculus system ⊢ is said to be sound w.r.t. a semantics ⊨ if ⊢ ϕ implies ⊨ ϕ.

In words, ⊢ ϕ holds whenever ⊨ ϕ 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) \( G = (Q, \Sigma, \delta, Q^0, Q^m) \) with the (finite) set of states \( Q \), the (nonempty) alphabet \( \Sigma \), the transition function \( \delta : Q \times \Sigma \to F \cup \{Q\} \), where \( F \) is the power set, the set of initial state \( Q^0 \) and a set of marked states \( Q^m \subseteq Q \). In this work, marked states are declared to our calculation and therefore, without loss of generality, we assume that \( Q^m = Q \) and we use the tuple \((Q, \Sigma, \delta, Q^0)\).

We write \( \delta(q, \sigma) \) if \( \delta(q, \sigma) \neq \emptyset \). The set of transitions in \( G \) is \( \Sigma \). We sometimes write \( q \xrightarrow{\sigma} q' \) instead of \( (q, \sigma, q') \in \Sigma \). Let \( \Sigma^* \) be the set of all finite strings over \( \Sigma \), including the empty string \( \epsilon \). We write \( q \in \Sigma^* \) for the concatenation of two strings \( s, t \in \Sigma^* \) and \( s \leq t \) when \( s \) is a prefix of \( t \). Further, the notation \( \delta \) is extended to strings in \( \Sigma^* \) in usual way [see Cassandras and Lafortune, 2008].

The closed language of the automaton \( G \) is defined by \( L(G) := \{ u \in \Sigma^* | \exists q \in Q^0 : \exists q \in Q : q \xrightarrow{\sigma} p \} \).

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 \( \bar{V} \), where each variable \( x \) in \( V \) has a corresponding (next-state) variable \( x' \) in \( \bar{V} \) 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 \( \phi_v \in \mathcal{A}_v \) denote the set of formulas over \( \bar{V} \) and/or \( v \) called action formulas (or just actions). It is assumed that the actions \( \phi_v \) are deterministic, i.e., \( \phi_v \) 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 \( c \equiv \phi_g \land \phi_v \). Further, we denote by \( \text{vars}(c) \) (resp. \( \text{vars}(c') \)) 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 \( E = (V, \ell, \Sigma, T, \ell', c, c') \), where \( V \) is a finite set of variables, \( L \) is a finite set of locations, \( \Sigma \) 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 \( V \cup \bar{V} \), \( \ell' \in L \) is the initial location, and \( c' \in \mathcal{C}_V \) is the initial guard.

We denote by \( \ell \xrightarrow{\sigma} \ell' \) the presence of a transition in \( E \), from location \( \ell \) to location \( \ell' \) 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 for two variables \( V \) and \( V' \), respectively, over the domain \( D \). Then, we associate the pair \((\eta, \eta')\) with a condition \( c \) if \((\eta, \eta') \models c \). If \((\eta, \eta') \models c \) holds, we call \( \eta \) and \( \eta' \) 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 \( c = x > 2 \land x' = x + 1 \). Given a present-state valuation \( \eta(x = a) \), if there exists some \( b \) in the domain such that \( (\eta(x = a), \eta'(x' = b)) \models c \), 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 \( c_{\cdot W} \) denote a new condition

\[
c_{\cdot W} \equiv c \land \bigwedge_{y \in W \cup \text{vars}(c')} y' = y,
\]

naming, \( c_{\cdot W} \) 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 \( E = (V, \ell, \Sigma, T, \ell', c) \) be an EFA. The finite-state automaton \( G(E) \) of the EFA is the tuple \((Q_E, \Sigma, \delta_E, Q^0_E)\) with \( Q_E = L \times D \), \( \Sigma_E = \Sigma \), \( Q^0_E = \{(\ell^0, \eta^0) \mid \eta^0 \models c^0 \) for valuation \( \eta^0 \}) \), and the explicit transition relation \( \tau \subseteq L \times D \times L \times Q_E \times Q_E \) according to

\[
\text{SEM} \quad \langle (\ell, \eta) \xrightarrow{c} (\ell', \eta') \rangle \quad \text{if} \quad c_{\cdot W} \quad \text{in} \quad G(E).
\]

Intuitively, states of \( G(E) \) are pairs of locations \( \ell \) and valuations \( \eta \). The transitions of \( G(E) \) are defined by the above inference rule, stating that whenever there exists a transition \( \ell \xrightarrow{c} \ell' \) in \( E \) and two valuations \( \eta \) and \( \eta' \) such that \((\eta, \eta') \models c_{\cdot W} \), there also exists a transition \( (\ell, \eta) \xrightarrow{c} (\ell', \eta') \) 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

\[
L(E) := \{ u \in \Sigma^* | \exists p \in Q_E : q \xrightarrow{\sigma} p \}.
\]

For two EFAs \( E \) and \( H \), we say that \( E = H \) if and only if \( L(E) = L(H) \).

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

Definition 5. (EFSC).

Let \( E_k = (V_k, L_k, \Sigma_k, T_k, \ell_k^0, c_k^0) \), \( k = 1, 2 \), be two EFAs. The Extended Full Synchronous Composition of \( E_1 \) and \( E_2 \) is the tuple \( E_1 \| E_2 = (V, L, \Sigma, T, \ell', c^0) \), where \( V = V_1 \cup V_2 \), \( L = L_1 \times L_2 \), \( \Sigma = \Sigma_1 \cup \Sigma_2 \), \( \ell' = (\ell^0_1, \ell^0_2) \), \( c^0 = c^0_1 \cup c^2_2 \), \( L' = L_1' \times L_2' \), and \( T \) is defined by the following rules:

\[
\text{SYN}1 \quad \langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma_1} \langle \ell_1', \ell_2 \rangle, \quad \sigma \in (\Sigma_1 \cup \Sigma_2)
\]

\[
\text{SYN}2 \quad \langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma_2} \langle \ell_1, \ell_2' \rangle, \quad \sigma \in (\Sigma_2 \cup \Sigma_1)
\]

\[
\text{SYN}3 \quad \langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma_1 \cup \sigma_2} \langle \ell_1', \ell_2' \rangle, \quad \sigma \in (\Sigma_1 \cap \Sigma_2)
\]

Note that, in the rule SYN3, if \( \sigma_1 \cup \sigma_2 \) 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].
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_{int} ⊆ 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]_{res}. 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 := {0, ..., 10} and initial guard e_0 = x = 0 ∧ y = 0 ∧ z = 0, as a running example, for which we want to interpret E w.r.t. V_{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 [E]_{res} according to the rules in Fig. 1. The sequents of \I{\eta} are of the form Γ \Rightarrow (ψ, ..., δ). The element η := (η_1, c_{\eta_{int}}) which we call configuration, is a pair of substitution η together with the formula c_{\eta_{int}}, as in Eq. (1). (ψ, ..., δ) is a placeholder for formulas, which we will process, and Γ contains the processed formulas. The informal semantics of sequents φ_0, ..., φ_m \Rightarrow (φ_0, ..., φ_m)(η_1, c_{\eta_{int}}) corresponds to the formula

\[ \bigwedge_{0 \leq i < m} \varphi_i \Rightarrow \bigwedge_{0 \leq i < n} \varphi_j \land \eta \land c_{\eta_{int}}, \]

where \( \hat{\eta} := \bigwedge_{v \in V} x = \eta(x) \) in particular \( \hat{\iota} := \top \).

The intuition behind the rules in Fig. 1 is the following:

Rule 1 states that for a root sequent \( \Rightarrow (η_1, c_{\eta_{int}}) \) with initial configuration \( (η_1, c_{\eta_{int}}) \), it constructs a new sequent of the form \( \Rightarrow (η(e_{\text{int}})) (e, T) \), where \( η(e_{\text{int}}) \) is the application of substitution \( η \) on \( c_{\eta_{int}} \). 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 \( e \); 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 \( V_{\text{int}} \) and \( t' \) is in the domain of \( x \), it extends the substitution \( \eta' \) by \( \eta'[x' \mapsto t'] \). Rule 7 takes any formula in the placeholder but instead it just conjuncts them to \( e \).

Note that, first applying Rule 5 and then 6 results in a "stronger" configuration since we now 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., \( \neg \neg φ \equiv φ \), \( T \lor φ \equiv T \), \( \bot \land φ \equiv \bot \), etc. For other simplification rules we refer to Galil [2003].

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

\[ \begin{array}{c}
\varphi_3 \Rightarrow \psi_5 \\
\Rightarrow ((y \leq x \lor z = x + 1) (x, T) [\Psi_3] \\
\Rightarrow ((x/1)(y \leq x \lor z = x + 1) (x, T) [\Psi_4] \\
\Rightarrow \psi_5 [\Psi_4] \\
\Rightarrow \psi_5 \end{array} \]
In (3), $\Psi_0$ denotes the root sequent and $\ast$ denotes the empty premise. Note that, in the sequent $\Psi_3$, the proof branches to two sequents, $\Psi_4$ and $\Psi_5$, because of the disjunction $y \leq 1 \lor z = 2$.

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

The soundness of the calculus $\vdash_{\text{Vint}}$ 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 sequent of $\Psi$. With abuse of notation, let $\text{succedent}^\ast(\Psi_0 \Rightarrow \Psi) := \text{succedent}(\Psi_0)$ and $\text{succedent}^\ast(\Psi_0 \Rightarrow \Psi_j) := \emptyset$ for $i, j \in N$. Further, let $\text{succedent}^\ast(\pi_j) := \{\text{succedent}^\ast(\beta)\}$ for all branches $\beta \in \pi_j$.

**Proposition 1.** Let $(\eta, c, \psi, \pi)\in \text{succedent}^\ast(\Psi_0 \Rightarrow \Psi_j)$. Then, $(\eta, c, \psi, \pi)\in \text{succedent}(\Psi_0 \Rightarrow \Psi_j)$, where $\text{succedent}(\Psi_0 \Rightarrow \Psi_j)$ is the proof tree (see Def. 1) of the root sequent $(\eta, c, \psi, \pi)\in \text{succedent}(\Psi_0 \Rightarrow \Psi_j)$.

The proof of Prop. 1 is by an induction on the structure of the proof tree $(\eta, c, \psi, \pi)\in \text{succedent}(\Psi_0 \Rightarrow \Psi_j)$. That is, we can inductively derive the validity of $(\eta, c, \psi, \pi)$ by following the leaves of the proof tree to the root sequent $(\eta, c, \psi, \pi)\in \text{succedent}(\Psi_0 \Rightarrow \Psi_j)$. Hence, the proof is left out.

For example, consider the proof tree in (3). Since all branches of (3) are closed, we have $\phi \vdash_{\text{Vint}} \pi_j(\eta, c, \psi, \pi) := \{\Psi_0 \Rightarrow \Psi_1 \Rightarrow \Psi_j\}$. Consequently, it follows that

$$[(x/1), (y \leq x \lor z = 1 + 1) \land y' = z + 1 \land x' = x] \in (y \leq 1 \land y' = z + 1 \land x' = x) \lor (z = 2 \land y' = 3 \land x' = 1) \in \nabla ((x'/1), y \leq 1 \land y' = z + 1), (x'/1, z = 2 \land y' = 3) \in (y \leq 1 \land y' = z + 1 \land x' = x) \lor (z = 2 \land y' = 3 \land x' = 1),$$

as expected by the result of Prop. 1.

For a condition $c$ and a present-state substitution $\eta$, we use $\eta^c$ and $\ast^c$ in $(\eta^c, c) \in \text{succedent}(\pi_j(\eta, c, \psi, \pi))$ as respectively the next-state substitutions and the residual conditions of partial evaluation of $c$ w.r.t. $\eta$.

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

**Algorithm 1 (Symbolic Interpretation of EFAs)**

**Require:** A labeled EFA $E = \{V, L, \Sigma, T, \ell^*, c^*, \ell, \Phi\}$ and a set of variables $\text{Vint} \subseteq V$.

1. **procedure** $[E]_{\text{Vint}}$:
2.   Let $\eta_{\text{int}} := \emptyset$, $\ell^* := \emptyset$, and $T^* := \emptyset$;
3.   Let $\eta^*$ be a valuation s.t. $\eta^* \models c^*$;
4.   Let $S$ be a stack of configurations;
5.   $(\forall x \in \text{Vint}) \eta_{\text{int}} := \eta_{\text{int}}[x \mapsto \eta^*(x)]; S \leftarrow (\ell^*, \eta_{\text{int}});$
6.   **repeat**
7.     $(\ell, \eta) := S.pop();$
8.     Let $(\ell, \eta)$ be a new location;
9.     if $(\ell, \eta) \in L^*$ then **continue**;
10.    $\Phi(\ell, \eta^*) := \emptyset$, $L^* \leftarrow (\ell, \eta^*)$;
11.   $T^* \leftarrow \{((\ell, \eta), \sigma, c^*, (\ell', \eta^*))\};$
12.   $S.push((\ell', \eta^*))$;
13. **end for**
14. **until** $S \neq \emptyset$;
15. **return** $E^* = \{V, L^*, \Sigma, T^*, (\ell^*, \eta_{\text{int}}), c^*, \ell, \Phi\};$
16. **end procedure**

22. **procedure** $\vdash_{\text{Vint}}(\eta, c, \psi, \pi)$
23.   $R := \emptyset$;
24.   if $(\eta, c, \psi, \pi) \vdash_{\text{Vint}} \pi_j(\eta, c, \psi, \pi)$ as in Fig. 1 then
25.     **for all** $(\eta^c, c^*) \in \text{succedent}(\pi_j(\eta, c, \psi, \pi))$
26.     if $c^* \models \ell^*$ then **continue**;
27.     $(\forall x \in \text{Vint}) \eta^c := (\eta^c[x/\eta^c(x')])[x' \mapsto \ell]$;
28.     $R \leftarrow (\eta^c, c^*)$;
29. **end for**
30. **end if**
31. **return** $R$;
32. **end procedure**

**Fig. 3.** The residual EFA $[E]_x$. Here, $\phi_0^1$ and $\phi_0^2$ are respectively equivalent to the formula $y \leq i \land y' = z + i$ and $z + i = i + z = i + 2$ for integer $i = 0, 1, 2$.

Let us illustrate Algorithm 1 by applying it to the EFA $E$ in Fig. 2 for $\text{Vint} := \{x\}$. 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 $(1, [x/0])$ with $\Phi((1, [x/0])) = [x/0]$ is added to the set $L^*$. Then, for outgoing transitions of location 1, i.e., $1 \xrightarrow{\alpha x'} = z + 1 \xrightarrow{1}$.
Algorithm 1 obtains the residual conditions and next-state values of $x' = x + 1$ and $\top$ w.r.t. $[x/0]$ by calling $\vdash_{\text{int}} \ldots$. The procedure $\vdash_{\text{int}} \ldots$ implements the partial evaluation process according to the rules in Fig. 1. Consequently, $\vdash_{\text{int}} ([x/0], x' = x + 1)$ and $\vdash_{\text{int}} ([x/0], \top)$ 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 $([1, x/1])$ labeled by $\Phi(1, [x/1]) = [x/1]$ and adds it to the set $L^*$ and stack $S$. Note that $1, [x/1] \neq ([1, x/1])$. Further, a new transition $([1, x/0]) \rightarrow ([1, x/1])$ is added to the (residual) transition set $T^*$. Similarly, for $([x/0], \top)$, we have the location $([2, x/0])$ with $\Phi(2, [x/0]) = [x/0]$ and the transition $([1, x/0]) \rightarrow ([2, x/0])$. Algorithm 1 iterates over the pairs in stack $S$, and when $S$ is empty, it terminates and returns the residual EFA $[E]_{\text{int}}$, see Fig. 3.

**Proposition 2.** (Algorithm 1 Correctness)

Let $E$ be a labeled EFA with set of variables $V$ over finite domain $D$. Let $\vdash_{\text{int}} \subseteq V$. Algorithm 1 terminates, and when it terminates it holds that $[E]_{\text{int}} V \vdash_{\text{int}} = [E] V$.

That is, interpretation of $E$ w.r.t. $\vdash_{\text{int}}$ and then with the remaining variables $V - \vdash_{\text{int}}$, 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 $\text{DESS} := \{E_1, \ldots, E_n\}$ 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 $\text{DESS}$ some of the variables might be internally updated by only one component. Let $V_1^a$ denote the set of such variables in $E_i$, i.e., $V_1^a := \{x \in V_1 | x \in (\text{vars}'(C_{V_1}) - \bigcup_{i,j \neq i} \text{vars}'(C_{V_j}))\}$.

**Theorem 2.** Let $E_i (i = 1, 2)$ be two EFAs in $\text{DESS}$ over $V_1$ with $V_1^a, V_{\text{int}} \subseteq V_1$, Consider $E := E_1 \| E_2$ and let $\vdash_{\text{int}} := \vdash_{1, \text{int}} \| V_2, \text{int}$. If $V_{\text{int}} \subseteq V_1^a$, then it holds that $[E_1]_{\text{int}} V_1, \text{int} = [E_1 V_1, \text{int}] [E_2 V_2, \text{int}]$.

In a straightforward way, we can further extend Theorem 2 to all components in $\text{DESS}$. 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 ($L_{\text{in}}$) and one exit load lock ($L_{\text{out}}$), nine chambers ($C_{ij}$, where, for $i = 1, 2, 3$, we have $j = 1, 2$, and for $i = 4$, we have $j = 1, 2, 3, 4$), 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 $R_{11}, C_{11}$, and $\ell_2$ 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 Lennartsson [2014] for the complete EFA models of the system.

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 $[R_1]_{\{R,C_{11},C_{12}\}}$ for the synchronized model $R_1 := \| R_{11}$ ($i = 1, \ldots, 4$) 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 $R_1^* := \{R_1\} \{R,C_{11},C_{12}\}$, the abstracted model $R_1^*$ is depicted in Fig. 6(b), where $\ell_0 = \{\ell_0, \ell_1, \ell_2, \ell_3\}, \ell_1 = \{\ell_4, \ell_5, \ell_6, \ell_7, \ell_8, \ell_9, \ell_{10}, \ell_{11}\}, \ell_2 = \{\ell_2, \ell_3\}$ 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 $\Rightarrow$. Also, it is straightforward to show that $\eta(c)$ is logically equivalent to $\hat{\eta} \land c$. We sketch the proof as follows. (Rule 1) In the conclusion, we have $\top \Rightarrow \hat{\eta} \land c \equiv \hat{\eta} \land c$, and in the premise we have $\top \Rightarrow \eta(c) \land \equiv \eta(c)$. Clearly, $\hat{\eta} \land c \equiv \eta(c)$. (Rule 2) Conjunctions $(\phi_1 \land \phi_2, \ldots)$ are inductively transformed to clauses separated by comma. Hence, straightforwardly the validity is preserved. (Rule 3) 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 $t' \in \mathcal{D}_t$, the substitution $\eta'$ is extended by $\eta'[x' \rightarrow t']$. 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 $V = \{x_1, \ldots, x_m, x_{m+1}, \ldots, x_n\}$ be the set of variables in $E$. Assume $V_{\text{int}} = \{x_1, \ldots, x_n\}$ and let $V_{\text{st}} = V - V_{\text{int}} = \{x_{m+1}, \ldots, x_n\}$. (Base) $w = \varepsilon$. By the hypothesis, $\ell^{c}$ is the same. On l.h.s, $[E]_{V_{\text{int}}}$ gives $\eta = [x_1/t_1, \ldots, x_m/t_m]$ for $t_i \in D$ ($i = 1, \ldots, m$) and then by $[E]_{V_{\text{st}}}$, we get $\eta'_w = \eta'_w [x_{m+1}/t_1, \ldots, x_n/t_n]$ for $t_j \in D$ ($j = m + 1, \ldots, n$). Hence, $\eta'_w \eta'_w = \eta'_w$ as expected. (Induction) $w = w'\sigma$ for some $w' \in \Sigma^*$ and $\sigma \in \Sigma$. We have $\langle \ell, \eta' \rangle \xrightarrow{\omega} \langle \ell, \sigma \rangle \xrightarrow{\omega} \langle \ell', \eta' \rangle$ for some valuations $\eta$ and $\eta'$. The proof of this part is similar to the (Base) for $(\ell', \eta')$, 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 different variables that appear in $E_i$. Further, consider $t_i := \ell_i \sigma_i \rightarrow \ell_i$ s.t. $t_i \in E_i$, $\sigma_i \in \Sigma_1 \cup \Sigma_2$, and $x_1, x_2 \in \text{vars}(\phi'_i) \cup \text{vars}(\phi_i)$, and let $t_{12} \in E_1 \cup E_2$, where $t_{12} := \langle \ell_1, \ell_2 \rangle \sigma_{1} \sigma_{2} \rightarrow \langle \ell_1, \ell_2 \rangle$. Then the proof follows on considering the different cases of $x_i$ member in sets $V_a$. We write $\ell \sigma_{1} \sigma_{2} \rightarrow \ell' \in E$ to the existence of the transition $\ell[\sigma_{1} \rightarrow \ell'[\sigma_{2} \rightarrow \ell' \in E]$. Clearly, by Def. 5, $\{t_1\} \{t_2\}$ result in a similar transition as $t_{12}$ up to renaming the locations, i.e., for the transition labels we have $\{ \phi_i \} [x_i/t_i]$. Clearly, by the hypothesis as similar arguments are valid for $\phi_2$. Finally, we demonstrate the effectiveness and necessity of the proposed technique combined with abstractions for nonblocking supervisory control of an industrial manufacturing system.

REFERENCES


