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

An Algorithm for Compositional Nonblocking Verification of Extended Finite-State Machines

Sahar Mohajerani [*] **Robi Malik** [**] **Martin Fabian** [*]

[*] *Department of Signals and Systems,
Chalmers University of Technology, Göteborg, Sweden,
(e-mail: {mohajera,fabian}@chalmers.se)*
[**] *Department of Computer Science, University of Waikato, Hamilton,
New Zealand, (e-mail: robi@waikato.ac.nz)*

Abstract: This paper describes an approach for *compositional nonblocking verification* of discrete event systems modelled as *extended finite-state machines* (EFSM). Previous results about finite-state machines in lock-step synchronisation are generalised and applied to EFSMs communicating via shared variables. This gives rise to an EFSM-based conflict check algorithm that composes EFSMs gradually and *partially unfolds* variables as needed. At each step, components are simplified using conflict-equivalence preserving abstraction. The algorithm has been implemented in the discrete event systems tool Supremica. The paper presents experimental results for the verification of two scalable manufacturing system models, and shows that the EFSM-based algorithm verifies some large models faster than previously used methods.

Keywords: Discrete event systems, extended finite-state machines, compositional verification, nonblocking, abstraction.

1. INTRODUCTION

Many discrete event systems are safety-critical, where failures can result in huge financial losses or even human fatalities. Logical correctness is a crucial property of these systems, and formal verification is an important part of guaranteeing it. This paper focuses on the verification of the *nonblocking* property (Ramadge and Wonham, 1989).

Formal verification requires a formal model, and *finite-state machines (FSM)* are widely used to represent discrete event systems (Ramadge and Wonham, 1989). FSMs describe the behaviour of a system using *states* and *transitions* between these states. Yet, data driven systems are more naturally modelled as *extended finite-state machines (EFSM)*, which communicate through bounded discrete variables. EFSMs have been similarly defined by several researchers (Chen and Lin, 2000; Yang and Gohari, 2005; Sköldstam *et al.*, 2007; Zhaoa *et al.*, 2012; Teixeira *et al.*, 2013).

While variables simplify the modelling of discrete event systems, the verification of large systems remains a challenge due to the *state-space explosion* problem. Verification must take all possible variable values into account, which can result in a large state space. To overcome state space explosion, various approaches including symbolic model checking (Baier and Katoen, 2008; McMillan, 1993) and abstraction (Graf and Steffen, 1990; Dams *et al.*, 1994) have been proposed. Another method is compositional verification using *conflict equivalence*, which has shown impressive results for large FSM models (Flordal and Malik, 2009; Su *et al.*, 2010; Malik and Leduc, 2013).

To apply FSM-based compositional methods to systems modelled as EFSMs, the model is first converted to a set of FSMs (Sköldstam *et al.*, 2007). While the conversion preserves the modular structure, making it possible to apply FSM-based methods directly, it has the drawback of significantly increasing the number of events. In some cases, the conversion takes longer than the verification.

Recently, an adaptation of compositional verification to EFSM models was proposed (Mohajerani *et al.*, 2013*a*), which removes the need to convert EFSMs to FSMs. In that work, *symbolic observation equivalence* is used as the only abstraction method. While observation equivalence reduces the state space significantly, it is not the best possible equivalence for nonblocking verification (Malik *et al.*, 2006). Several conflict-preserving abstraction rules for FSMs are known (Flordal and Malik, 2009; Malik and Leduc, 2013) that extend beyond observation equivalence.

This paper applies the compositional framework (Flordal and Malik, 2009) to systems modelled as EFSMs communicating via shared variables. In addition to *partial unfolding*, which removes variables from the system, and *selfloop removal*, which removes transitions, the paper provides a general framework to apply every conflict-preserving FSM abstraction rule (Flordal and Malik, 2009; Malik and Leduc, 2013) directly to EFSMs. The proposed EFSM-based compositional algorithm is implemented in Supremica (Åkesson *et al.*, 2006), and has been used successfully to verify two scalable manufacturing systems.

This paper is structured as follows. Sect. 2 introduces the notation and concepts for EFSMs. Sect. 3 presents the idea of compositional nonblocking verification of EFSMs and shows how an EFSM can be simplified without converting it to an FSM. Afterwards, Sect. 4 describes the algorithm for compositional nonblocking verification of EFSM systems, Sect. 5 presents the experimental results, and Sect. 6 adds some concluding remarks.

978-3-902823-61-8/2014 © IFAC

10.3182/20140514-3-FR-4046.00039

2. PRELIMINARIES

2.1 Finite-State Machines

The standard means to model discrete event systems (Ramadge and Wonham, 1989) are *finite-state machines (FSM)*, which synchronise on shared *events* (Hoare, 1985). Events are taken from a finite alphabet Σ. The special *silent event* $\tau \notin \Sigma$ is not included in Σ unless explicitly mentioned using the notation $\Sigma_\tau = \Sigma \cup \{\tau\}$. Further, Σ^* is the set of all finite *traces* of events from Σ, including the *empty trace* ε. The concatenation of two traces $s, t \in \Sigma^*$ is written as st.

Definition 1. A *finite-state machine (FSM)* is a tuple $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, where Σ is a finite set of events, Q is a finite set of *states*, $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ is the *state transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $Q^\omega \subseteq Q$ is the set of *marked states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ_τ^* by $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. The transition relation is also defined for state sets $X, Y \subseteq Q$, for example $X \xrightarrow{s} y$ means $x \xrightarrow{s} y$ for some $x \in X$.

Definition 2. Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ be two FSMs. The *synchronous composition* of G_1 and G_2 is

$$G_1 \| G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \rightarrow, Q_1^\circ \times Q_2^\circ, Q_1^\omega \times Q_2^\omega \rangle \quad (1)$$

where

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2) \text{ if } \sigma \in \Sigma_1 \cap \Sigma_2,\ x_1 \xrightarrow{\sigma}_1 y_1,\ x_2 \xrightarrow{\sigma}_2 y_2\,;$$
$$(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2) \text{ if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\},\ x_1 \xrightarrow{\sigma}_1 y_1\,;$$
$$(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2) \text{ if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau\},\ x_2 \xrightarrow{\sigma}_2 y_2\,.$$

This paper concerns verification of the *nonblocking* property, which is commonly used in supervisory control theory of discrete event systems (Ramadge and Wonham, 1989).

Definition 3. An FSM $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ is *nonblocking* if, for every $s \in \Sigma_\tau^*$ and every $x \in Q$ such that $Q^\circ \xrightarrow{s} x$, there exists $t \in \Sigma_\tau^*$ such that $x \xrightarrow{t} Q^\omega$.

2.2 Extended Finite-State Machines

Extended finite-state machines (EFSM) are similar to conventional finite-state machines (FSM), but augmented with *updates* associated to the transitions (Chen and Lin, 2000; Sköldstam *et al.*, 2007). Updates are predicates containing variables.

A *variable* v is an entity associated with a finite domain $\mathrm{dom}(v)$ and an initial value $v^\circ \in \mathrm{dom}(v)$. Let $V = \{v_0, \ldots, v_n\}$ be the set of variables with domain $\mathrm{dom}(V) = \mathrm{dom}(v_0) \times \cdots \times \mathrm{dom}(v_n)$. An element of $\mathrm{dom}(V)$ is also called *valuation* and denoted by $\hat{v} = (\hat{v}_0, \ldots, \hat{v}_n)$ with $\hat{v}_i \in \mathrm{dom}(v_i)$. The *initial valuation* is $\hat{v}^\circ = (v_0^\circ, \ldots v_n^\circ)$. A second set of variables, called *next-state variables* and denoted by $V' = \{\, v' \mid v \in V \,\}$ with $\mathrm{dom}(V') = \mathrm{dom}(V)$, is used to describe how variables are updated by transitions.

For example, let x be a variable with domain $\mathrm{dom}(x) = \{0, \ldots, 5\}$ and initial value $x^\circ = 0$. A transition with update $x' = x + 1$ changes the variable x by adding 1 to its current value, if it currently is less than 5. Otherwise (if $x = 5$) the transition is disabled and no updates are performed. Another possibility is to write the formula $x' = \min(x+1, 5)$, in which case the transition remains enabled when $x = 5$. The update $x = 3$ disables a transition unless $x = 3$ in the current state, and the value of x in the next state is not changed. Differently, the update $x' = 3$ always enables its transition, and the value of x in the next state is forced to be 3.

The set of all update predicates using variables in V and V' is denoted by Π_V. For an update $p \in \Pi_V$, the term $\mathrm{vars}(p)$ denotes the set of all variables that occur in p, and $\mathrm{vars}'(p)$ denotes the set of all variables modified by p. For example, if $p \equiv x' = y+1$ then $\mathrm{vars}(p) = \{x, y\}$, and $\mathrm{vars}'(p) = \{x\}$. An update p with $\mathrm{vars}'(p) = \emptyset$ is called a *pure guard*. Its execution leaves all variables unchanged.

Definition 4. An *extended finite-state machine (EFSM)* is a tuple $E = \langle V, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, where V is a finite set of variables, Q is a finite set of *locations*, $\rightarrow \subseteq Q \times \Pi_V \times Q$ is the *conditional transition relation*, $Q^\circ \subseteq Q$ is the set of *initial locations*, and $Q^\omega \subseteq Q$ is the set of *marked locations*.

The expression $x \xrightarrow{p} y$ denotes the presence of a transition in E, from location x to location y with update $p \in \Pi_V$. On the occurrence of such a transition, the EFSM changes its location from x to y while updating the variables in $\mathrm{vars}'(p)$ in accordance with p; variables not contained in $\mathrm{vars}'(p)$ remain unchanged.

Usually, reactive systems are modelled as several components interacting with each other. An *EFSM system* is a collection of interacting EFSMs,

$$\mathcal{E} = \{E_1, \ldots, E_n\}\ . \quad (2)$$

The behaviour of such a system is expressed using *interleaving* semantics (Baier and Katoen, 2008). Synchronisation is achieved indirectly through the variables.

Definition 5. Given two EFSMs $E = \langle V_E, Q_E, \rightarrow_E, Q_E^\circ, Q_E^\omega \rangle$ and $F = \langle V_F, Q_F, \rightarrow_F, Q_F^\circ, Q_F^\omega \rangle$ the *composition* of E and F is

$$E \| F = \langle V_E \cup V_F, Q_E \times Q_F, \rightarrow, Q_E^\circ \times Q_F^\circ, Q_E^\omega \times Q_F^\omega \rangle\ , \quad (3)$$

where

- $(x_E, x_F) \xrightarrow{p_E} (y_E, x_F)$ if $x_E \xrightarrow{p_E}_E y_E$;
- $(x_E, x_F) \xrightarrow{p_F} (x_E, y_F)$ if $x_F \xrightarrow{p_F}_F y_F$.

To apply the nonblocking property to EFSMs, they are interpreted as FSMs. The standard approach to do this is *flattening*, which introduces states for all combinations of locations and variable values (Baier and Katoen, 2008).

Definition 6. Let $E = \langle V, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an EFSM. The *monolithic flattened* FSM of E is $U(E) = \langle \emptyset, Q_U, \rightarrow_U, Q_U^\circ, Q_U^\omega \rangle$ where

- $Q_U = Q \times \mathrm{dom}(V)$,
- $(x, \hat{v}) \xrightarrow{\tau}_U (y, \hat{w})$ if $x \xrightarrow{p} y$ and $p(\hat{v}, \hat{w}) = \text{true}$,
- $Q_U^\circ = Q^\circ \times \{\hat{v}^\circ\}$
- $Q_U^\omega = Q^\omega \times \mathrm{dom}(V)$.

The domain of variables and the initial valuation $\hat{v}_0$ are determined by variables as they are properties of variables. The variable values ensure the correct sequencing of transitions in the flattened FSM. The flattened FSM of an EFSM system $\mathcal{E} = \{E_1, \ldots, E_n\}$ is $U(\mathcal{E}) = U(E_1 \| \cdots \| E_n)$. Using these definitions, the nonblocking property is also defined for EFSMs and EFSM systems.

Definition 7. An EFSM system $\mathcal{E}$ is nonblocking if the flattening $U(\mathcal{E})$ is nonblocking.

3. EFSM-BASED COMPOSITIONAL VERIFICATION

The straightforward approach to check whether a modular system

$$\mathcal{E} = \{E_1, E_2, \ldots, E_n\} \tag{4}$$

is nonblocking, is to explicitly construct the monolithic representation of the system and check for each reachable state whether it is possible to reach a marked state. Clearly, this technique is limited by the *state space explosion* problem. In an attempt to alleviate state-space explosion, *compositional* verification (Flordal and Malik, 2009) seeks to repeatedly rewrite individual system components and, for example, replace E_1 in (4) by an *abstraction* E'_1, to analyse the simpler system

$$\mathcal{E}' = \{E'_1, E_2, \ldots, E_n\} \ . \tag{5}$$

The abstraction steps to simplify the individual components E_i must satisfy certain conditions to guarantee that the verification result is preserved. One equivalence to support nonblocking verification is *conflict equivalence.*

Definition 8. (Malik *et al.*, 2006) Two EFSMs or FSMs E_1 and E_2 are *conflict equivalent*, written $E_1 \simeq_{\text{conf}} E_2$, if for any component T, it holds that $E_1 \parallel T$ is nonblocking if and only if $E_2 \parallel T$ is nonblocking.

Due to the congruence properties of conflict equivalence (Malik *et al.*, 2006), FSM components in a composition can be replaced by conflict equivalent components while preserving the nonblocking property. It is straightforward to lift this result to EFSMs.

Proposition 1. Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ and $\mathcal{F} = \{F_1, E_2, \ldots, E_n\}$ be EFSM systems such that $E_1 \simeq_{\text{conf}} F_1$. Then $U(\mathcal{E})$ is nonblocking if and only if $U(\mathcal{F})$ is nonblocking.

If no component in a system (4) can be simplified individually, then either some EFSMs components must be composed or some variables must be expanded into locations. These operations result in new EFSMs that are abstracted again, and the procedure continues until the system is simple enough to be verified directly.

In the following, Sect. 3.1–3.3 describe the methods of partial composition, update simplification, and partial unfolding of variables. Afterwards, Sect. 3.4 and 3.5 propose two methods to simplify EFSMs. Formal correctness proofs of the key results can be found in (Mohajerani *et al.*, 2013*b*).

3.1 Partial Composition

Composition is the simplest step in compositional verification. It is always possible to replace some components of an EFSM system by their composition. This operation does not reduce the state space, but it is necessary when all other means of simplification have been exhausted. The following result follows directly from the definitions. The flattened FSMs before and after partial composition are not only equivalent with respect to nonblocking, but also identical up to isomorphism.

Proposition 2. Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be an EFSM system, and $\mathcal{F} = \{E_1 \parallel E_2, E_3, \ldots, E_n\}$. Then $U(\mathcal{E})$ is nonblocking if and only if $U(\mathcal{F})$ is nonblocking.

3.2 Update Simplification

An important aspect to reasoning about EFSM systems is the symbolic manipulation of updates, in order to keep the formulas as simple as possible.

Proposition 3. Let $E = \langle V, Q, \rightarrow_E, Q^\circ, Q^\omega \rangle$ be an EFSM with $x \xrightarrow{p}_E y$, let $\tilde{p}$ be an update logically equivalent to p such that $\text{vars}'(p) = \text{vars}'(\tilde{p})$, and let $F = \langle V, Q, \rightarrow_F, Q^\circ, Q^\omega \rangle$ such that $\rightarrow_F = (\rightarrow_E \setminus \{(x, p, y)\}) \cup \{(x, \tilde{p}, y)\}$. Then $E \simeq_{\text{conf}} F$.

An update in an EFSM can always be replaced by a logically equivalent update, provided that both updates contain the same next-state variables. The condition on the next-state variables is necessary to ensure that the set of unchanged variables, i.e., variables not occurring primed in an update, is preserved.

3.3 Partial Unfolding

Partial unfolding (Mohajerani *et al.*, 2013*a*) is the process of removing a variable from an EFSM and expanding its values into locations.

Definition 9. Let $E = \langle V, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an EFSM, and let $z \in V$. The result of *partially unfolding* z in E is the EFSM $E \setminus z = \langle V \setminus \{z\}, Q \times \text{dom}(z), \rightarrow_{\setminus z}, Q^\circ \times \{z^\circ\}, Q^\omega \times \text{dom}(z) \rangle$ where

$$(x, a) \xrightarrow{\exists z \exists z' (p \wedge z = a \wedge z' = b)}_{\setminus z} (y, b) \tag{6}$$

for all $a, b \in \text{dom}(z)$ such that $x \xrightarrow{p} y$, and such that $z \notin \text{vars}'(p)$ implies $a = b$.

A variable is called *local* in an EFSM system, if it appears in only one component. Since local variables are not needed for interaction with any other component, they can be removed by partial unfolding. The following result confirms that partial unfolding of a local variable preserves the nonblocking property of an EFSM system. Similar to partial composition, the flattened FSMs before and after partial unfolding are identical up to isomorphism.

Proposition 4. Let $\mathcal{E} = \{E_1, \ldots, E_n\}$ be an EFSM system, $z \in \text{vars}(E_1) \setminus \bigcup_{i=2}^{n} \text{vars}(E_i)$, and $\mathcal{F} = \{E_1 \setminus z, E_2, \ldots, E_n\}$. Then $U(\mathcal{E})$ is nonblocking if and only if $U(\mathcal{F})$ is nonblocking.

After partially unfolding a local variable z, the resulting updates $\exists z \exists z' (p \wedge z = a \wedge z' = b)$ can be simplified using Prop. 3. It is enough to replace z and z' in the update p by the constants a and b, respectively, and simplify the resulting update. In this way, partial unfolding removes local variables at the price of an increase in the number of locations. Its application may be deferred in favour of other operations. On the other hand, partial unfolding often simplifies or removes some updates, making it possible to apply the abstraction methods following below.

3.4 Selfloop Removal

In compositional verification of FSM systems, selfloop transitions labelled by silent events are immediately removed because no state change is possible by these transitions. In EFSM models, selfloop transitions labelled by pure guards have the same property.

Proposition 5. Let $E = \langle V, Q, \rightarrow_E, Q^\circ, Q^\omega \rangle$ be an EFSM. Let $F = \langle V, Q, \rightarrow_F, Q^\circ, Q^\omega \rangle$ such that

$$\rightarrow_F = \rightarrow_E \setminus \{\, (x,p,x) \in \rightarrow_E \mid x \in Q \text{ and } \mathrm{vars}'(p) = \emptyset \,\} \ .$$

Then $E \simeq_{\mathrm{conf}} F$.

In combination with Prop. 1, it is clear that selfloops labelled by pure guards can be removed without affecting the nonblocking property of an EFSM system. This simple abstraction step can increase the applicability of other abstraction methods.

3.5 FSM-Based Conflict Equivalence Abstraction

Several abstraction methods for ordinary FSMs have been proposed (Flordal and Malik, 2009). The approach suggested here is to use the methods developed for FSMs to abstract EFSMs directly without flattening.

Definition 10. Let $E = \langle V, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an EFSM. The *FSM form* of E is $\varphi(E) = \langle \Sigma, Q, \rightarrow_\varphi, Q^\circ, Q^\omega \rangle$, where

$$\Sigma = \{\, p \in \Pi_V \setminus \{\mathrm{true}\} \mid x \xrightarrow{p} y \text{ for some } x, y \in Q \,\} \ ;$$

$$\rightarrow_\varphi = \{\, (x,p,y) \in \rightarrow \mid p \neq \mathrm{true} \,\} \cup \{\, (x,\tau,y) \mid x \xrightarrow{\mathrm{true}} y \,\} \ .$$

The FSM form is obtained by reinterpreting all updates of an EFSM as simple events, except for true updates, which are replaced by silent τ-transitions. Unlike in the flattened FSM, there is no need to evaluate and expand the variable values. The conversion to FSM form and back is a straightforward operation that does not incur any blowup, yet it makes it possible to apply FSM simplification to EFSMs.

Proposition 6. Let E and F be two EFSMs with FSM forms $\varphi(E)$ and $\varphi(F)$. If $\varphi(E) \simeq_{\mathrm{conf}} \varphi(F)$ then $E \simeq_{\mathrm{conf}} F$.

Prop. 6 provides a general method to abstract an EFSM while preserving conflict equivalence, which in combination with Prop. 1 guarantees that the nonblocking property of an EFSM system is unchanged.

Example 1. Consider EFSM E in Fig. 1, where initial locations have an incoming arrow and marked locations are coloured black. Assume that $\mathrm{dom}(x) = \mathrm{dom}(y) = \{0,1,2,3\}$ and $x^\circ = y^\circ = 0$, and y is a local variable. Partial unfolding of y followed by update simplification results in $E\backslash y$, shown in Fig. 1. Here, $x < 2$ is a pure guard, so the selfloop labelled by this update can be removed, resulting in E' in Fig. 1. The FSM form $\varphi(E')$ of E', also shown in Fig. 1, has the events

$$\Sigma = \{x' = 1, x' = 2\} \tag{7}$$

in addition to a τ-transition replacing the true update. States $(\mathsf{a},0)$ and $(\mathsf{b},1)$, and states $(\mathsf{c},3)$ and $(\mathsf{d},2)$ in $\varphi(E')$ can be merged using the conflict-preserving abstraction methods of *observation equivalence* and the *Active Events Rule* (Flordal and Malik, 2009). The resultant FSM is converted back to the EFSM $\widetilde{E}$ shown in Fig. 1. It follows from Props. 5 and 6 that $E\backslash y \simeq_{\mathrm{conf}} \widetilde{E}$. Note that, since the Active Events Rules requires equal sets of enabled events, states $(\mathsf{c},3)$ and $(\mathsf{d},2)$ can only be merged after selfloop removal.

4. ALGORITHM

Algorithm 1 shows how to verify the nonblocking property of an EFSM system using the above results. The algorithm repeatedly either chooses a local variable to be partially unfolded or two EFSMs to be composed, and simplifies the resultant EFSMs, until only one EFSM is left.

Algorithm 1 EFSM-based compositional verification

```
1:  input $\mathcal{E} = \{E_1, E_2, \dots, E_n\}$, $V = \{v_1, v_2, \dots, v_m\}$
2:  while $|\mathcal{E}| > 1 \lor |V| > 0$ do
3:    if there are local variables then
4:      $(v, E) \leftarrow \mathrm{selectLocalVariable}(\mathcal{E})$
5:      $V \leftarrow V \setminus \{v\}$, $\mathcal{E} \leftarrow \mathcal{E} \setminus \{E\}$
6:      $E \leftarrow \mathrm{partialUnfold}(v, E)$
7:    else
8:      $pair \leftarrow \mathrm{selectPair}(\mathcal{E})$
9:      $\mathcal{E} \leftarrow \mathcal{E} \setminus pair$
10:     $E \leftarrow \mathrm{composition}(pair)$
11:   end if
12:   $E \leftarrow \mathrm{simplify}(E)$
13:   $\mathcal{E} \leftarrow \mathcal{E} \cup \{E\}$
14: end while
15: monolithicVerification($\mathcal{E}$)
```

Partial unfolding is preferred over composition, as the potential for simplification afterwards is greater. If possible, line 4 selects a local variable v together with the EFSM E containing it. The choice of what variable to unfold significantly affects the performance. In some cases, it is better to unfold a variable with a small domain as this gives smaller EFSMs, while in some cases it is better to unfold a variable that appears frequently as this allows for more simplification. The presented implementation is based on the following heuristics.

maxO selects the local variable that occurs in the largest number of updates in its EFSM.

maxS selects the local variable that appears in the largest number of selfloops.

maxES selects the local variable with smallest product of its domain and number of locations of its EFSM.

While **maxO** simplifies more updates, which potentially results in more abstraction, **maxS** attempts to increase the performance of selfloop removal, which may also lead to more abstraction. After some experimentation, the order given above was determined the most effective, and therefore the heuristics are applied in this order. If a heuristic gives equal preference to two variables, the next one is used to break the tie.

After selection of a local variable, the variable is partially unfolded and removed from the system in lines 5 and 6. The resultant EFSM is simplified later in line 12.

If there is no local variable, the selectPair method in line 8 selects a pair of EFSMs, which are removed from $\mathcal{E}$ and composed in lines 9 and 10. To select the two EFSMs to compose, another set of heuristics is used. The following heuristics attempt to keep the number of locations or the number of shared variables as small as possible.

minV gives preference to EFSM pairs with the smallest number of non-local variables used in the EFSMs to be composed.

minF chooses a pair of EFSMs with the smallest number of other EFSMs it shares variables with.

minSynch gives preference to pairs with the smallest number of locations in their composition.

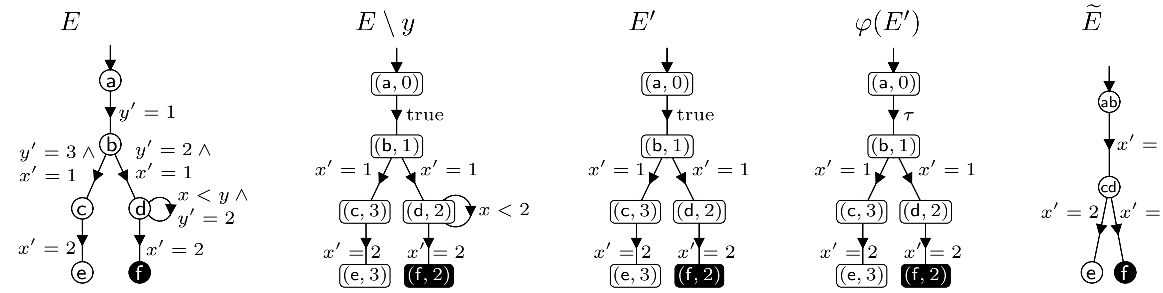

Fig. 1. Automata for Example 1.

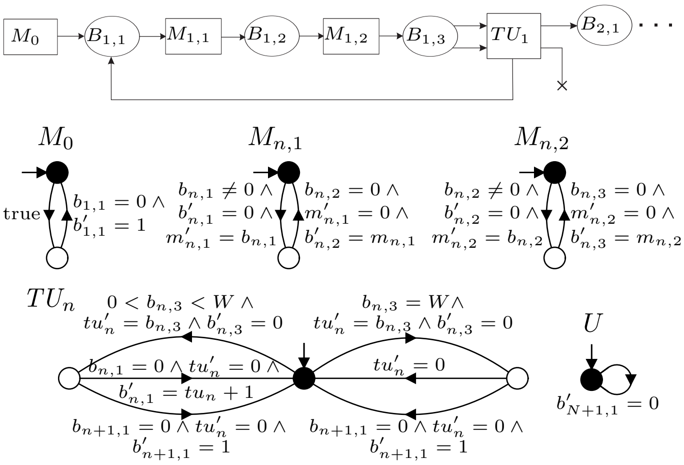
M_0
$B_{1,1}$
$M_{1,1}$
$B_{1,2}$
$M_{1,2}$
$B_{1,3}$
TU_1
$B_{2,1}$
. . .
×
M_0
$M_{n,1}$
$M_{n,2}$
true
$b_{1,1} = 0 \wedge$ $b'_{1,1} = 1$
$b_{n,1} \neq 0 \wedge$ $b'_{n,1} = 0 \wedge$ $m'_{n,1} = b_{n,1}$
$b_{n,2} = 0 \wedge$ $m'_{n,1} = 0 \wedge$ $b'_{n,2} = m_{n,1}$
$b_{n,2} \neq 0 \wedge$ $b'_{n,2} = 0 \wedge$ $m'_{n,2} = b_{n,2}$
$b_{n,3} = 0 \wedge$ $m'_{n,2} = 0 \wedge$ $b'_{n,3} = m_{n,2}$
TU_n
$0 < b_{n,3} < W \wedge$ $tu'_n = b_{n,3} \wedge b'_{n,3} = 0$
$b_{n,3} = W \wedge$ $tu'_n = b_{n,3} \wedge b'_{n,3} = 0$
$b_{n,1} = 0 \wedge tu'_n = 0 \wedge$ $b'_{n,1} = tu_n + 1$
$tu'_n = 0$
$b_{n+1,1} = 0 \wedge tu'_n = 0 \wedge$ $b'_{n+1,1} = 1$
$b_{n+1,1} = 0 \wedge tu'_n = 0 \wedge$ $b'_{n+1,1} = 1$
U
$b'_{N+1,1} = 0$

Fig. 2. Transfer line system.

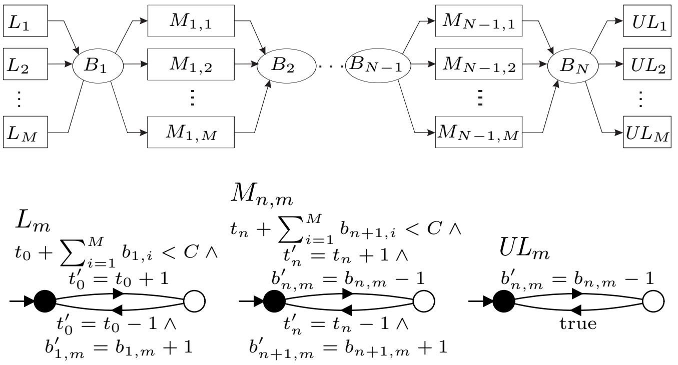
L_1
L_2
L_M
B_1
$M_{1,1}$
$M_{1,2}$
$M_{1,M}$
B_2
B_{N-1}
$M_{N-1,1}$
$M_{N-1,2}$
$M_{N-1,M}$
B_N
UL_1
UL_2
UL_M
L_m
$t_0 + \sum_{i=1}^{M} b_{1,i} < C \wedge$ $t'_0 = t_0 + 1$
$t'_0 = t_0 - 1 \wedge$ $b'_{1,m} = b_{1,m} + 1$
$M_{n,m}$
$t_n + \sum_{i=1}^{M} b_{n+1,i} < C \wedge$ $t'_n = t_n + 1 \wedge$ $b'_{n,m} = b_{n,m} - 1$
$t'_n = t_n - 1 \wedge$ $b'_{n+1,m} = b_{n+1,m} + 1$
UL_m
$b'_{n,m} = b_{n,m} - 1$
true

Fig. 3. Parallel manufacturing lines system.

minV and **minF** select a pair of EFSMs the composition of which increases the possibility of having local variables, while **minSynch** attempts to keep the number of locations as small as possible. Since unfolding local variables was found to produce more abstraction, the heuristics are used in the above order to break ties.

The resultant EFSM from partial unfolding or composition is simplified in line 12. The first abstraction rule applied by the simplify method is the removal of selfloops labelled by pure guards according to Prop. 5. Next, the EFSM is converted to FSM form, and based on Prop. 6, conflict-preserving abstractions (Flordal and Malik, 2009) are used to simplify the FSM. Then the abstracted FSM is converted back to an EFSM and added to $\mathcal{E}$ in line 13.

The loop terminates when $\mathcal{E}$ contains only one EFSM and all the variables are unfolded. The final EFSM is passed to standard nonblocking verification in line 15. Based on Props. 1-6, the result is nonblocking if and only if the original system is nonblocking.

5. EXPERIMENTAL RESULTS

The EFSM-based compositional nonblocking verification has been implemented in the discrete event systems tool Supremica (Åkesson *et al.*, 2006). This section shows some experiments to compare this algorithm with two other methods for nonblocking verification. Subsection 5.1 describes the models used for the experiments, subsection 5.2 briefly outlines the two alternative methods considered in the comparison, and subsection 5.3 presents the results.

5.1 Examples

The experiments are set up to verify the nonblocking property of the two manufacturing systems described below. Both systems have a regular structure and can be scaled up to give arbitrarily large EFSM systems that test the algorithms to their limits.

Transfer Line The model is a scalable version of the transfer line with rework cycles (Teixeira *et al.*, 2013). The system consists of N serially connected cells linked by buffers. Each cell n for $1 \le n \le N$ has two machines $M_{n,1}$, $M_{n,2}$, a test unit TU_n, and three buffers $B_{n,1}$, $B_{n,2}$, and $B_{n,3}$ of capacity 1. Initially workpieces are picked up by machine M_0 and placed in buffer $B_{1,1}$ to enter cell 1. Within cell n, machine $M_{n,1}$ takes workpieces from $B_{n,1}$, manufactures, and places them in $B_{n,2}$, from where they are picked up by $M_{n,2}$ and placed in $B_{n,3}$. Then they go to the test unit TU_n, which either approves a workpiece and sends it to the next cell via $B_{n+1,1}$, or rejects it and puts it back in $B_{n,1}$. Workpieces can be put back in $B_{n,1}$ up to $W - 1$ times; on the W-th cycle they are either approved or scrapped. The last cell N places its accepted workpieces in a final buffer $B_{N+1,1}$, from where they are removed by an unloader U. Fig. 2 shows the overview of the first cell and the EFSM model of the system.

The model uses variables $b_{n,i}$, $m_{n,i}$, and tu_n with domain $\{0, \dots, W\}$ to represent the contents of the buffers, machines, and test units. A value of 0 indicates that there is no workpiece in the corresponding unit, while a value of $k > 0$ indicates the presence of a workpiece that has entered the corresponding cell k times. The model thus has two parameters: $N \ge 1$ is the number of serially connected cells, and $W \ge 1$ is the maximum number of work cycles. The experiments use the presented model, which is nonblocking, and a blocking version obtained by removing the unloader U.

Parallel Manufacturing Lines The system consists of M parallel manufacturing lines each processing workpieces of a particular type. Each line m consists of $N+1$ serially connected machines $M_{n,m}$. The first machines in each line are called loaders L_m, and the last machines are called unloaders UL_m. The parallel lines share N buffers B_n, which can store workpieces of different type, but never more than C workpieces at a time. Fig. 3 shows the system layout and the EFSM model of the system.

Each buffer is represented by m variables $b_{n,m}$ with domain $\{0,\ldots,C\}$, representing the number of type m workpieces in buffer B_n. The system is controlled such that, for each n, the number of workpieces in the machines $M_{n,m}$ and the following buffer B_{n+1} never exceeds the buffer capacity C. This control is facilitated by variables t_n with domain $\{0,\ldots,M\}$, which hold the total number of workpieces currently processed in the machines numbered n.

The model has three parameters: $M \geq 1$ is the number of parallel lines, $N \geq 1$ is the number of buffers, and $C \geq 1$ is the capacity of the buffers. It is nonblocking for all parameter values.

5.2 Other Methods

The performance of the EFSM-based compositional verification algorithm is compared to the following methods.

BDD The BDD-based algorithm converts the EFSM model to a symbolic representation in the form of Binary Decision Diagrams (BDDs) and explores the full state space symbolically (McMillan, 1993).

FSM The FSM-based compositional algorithm converts the EFSM model to a set of FSMs and then applies the compositional algorithm (Flordal and Malik, 2009). The EFSM model is *modularly* flattened by creating a collection of *location FSMs* and *variable FSMs* (Mohajerani *et al.*, 2013*a*). Location FSMs use the EFSM locations as states but replace the updates with events. Variable FSMs use the domain of a variable as their states space and keep track of the value of that variable. The flattened FSM system has events of the form $(E;\hat{v};\hat{w})$ for each update p in EFSM E and all valuations $\hat{v} \in \mathrm{dom}(\mathrm{vars}(p))$ and $\hat{w} \in \mathrm{dom}(\mathrm{vars}'(p))$ such that $p(\hat{v},\hat{w}) = \mathrm{true}$. In the worst case, the number of events created for an update is the product of the size of the domains of its variables. For example, the events created for the update $tu' = 0$ in the EFSM TU in the Transfer Line model with $W = 3$ work cycles are $(TU,0,0)$, $(TU,1,0)$, $(TU,2,0)$, and $(TU,3,0)$. An update such as $t_0 + \sum_{i=1}^{M} b_{1,i} < C$ in the Parallel Manufacturing Lines model can produce up to $M \cdot C^M$ events.

5.3 Evaluation

Figs. 4–6 show experimental results for the Transfer Line and the Parallel Manufacturing Lines, with different parameter values, using the symbolic (BDD), FSM-based compositional (FSM) and EFSM-based compositional (EFSM) algorithms.

The experiments were run on a standard desktop computer using a single core 2.67 GHz CPU. Memory usage was limited to 2 GB, and runtime was limited to 50 minutes: if a verification attempt took more resources, it was aborted.

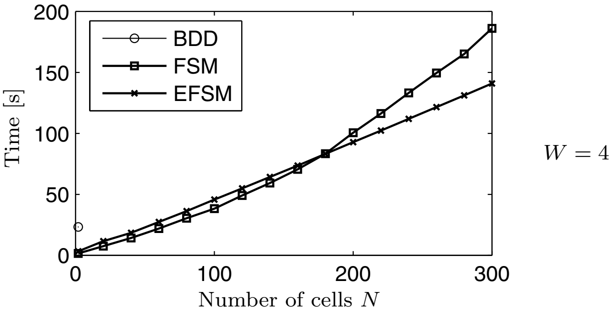

Fig. 4. Runtimes for Transfer Line, nonblocking version.

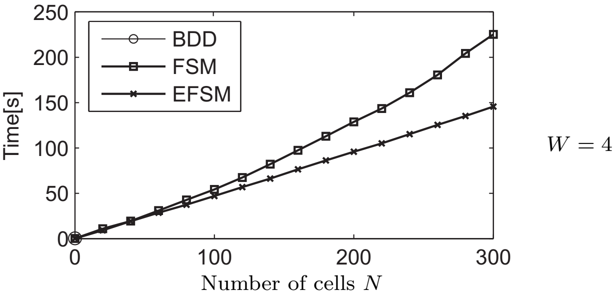

Fig. 5. Runtimes for Transfer Line, blocking version.

Figs. 4 and 5 show runtimes for the nonblocking and blocking versions of the Transfer Line example with $W = 4$ work cycles for an increasing number N of cells. The runtime increases linearly for the EFSM and quadratically for the FSM algorithm, with FSM being slightly faster for small values of N. Because of the simplicity of the updates in this model, the number of events in the flattened FSM system grows linearly in N. In this case, the FSM algorithm achieves the same abstractions as the EFSM algorithm, and it is faster initially as it avoids time-consuming update manipulations. However, the FSM-based compositional algorithm is quadratic in the number of events because of the way heuristics are evaluated, and when the number of cells is increased beyond 180 in the nonblocking version or 21 in the blocking version, the EFSM algorithm becomes faster. The BDD method cannot solve this problem for more than $N = 2$ cells, probably due to difficulties finding a workable ordering of the variables.

Fig. 6 shows runtimes for the Parallel Manufacturing Lines example with $M = 3$ parallel lines, for increasing manufacturing line lengths N and buffer capacities C. Due to the more complex updates in the model, the number of events in the flattened FSM system grows in $O(N \cdot C^{M-1})$: at $M = 3$, it grows linearly in N and quadratically in C. The diagram shows a quadratic runtime increase in N, because the FSM-based compositional algorithm is quadratic in the number of events. When C is increased, this quadratic growth causes the FSM method to fail quickly.

In contrast, the BDD algorithm copes well with an increase in C, as this only causes a moderate increase in the complexity of the symbolic model. However an increase in the number N of buffers causes a rapid increase in BDD runtimes, mainly because of a quadratic growth in the number of iterations.

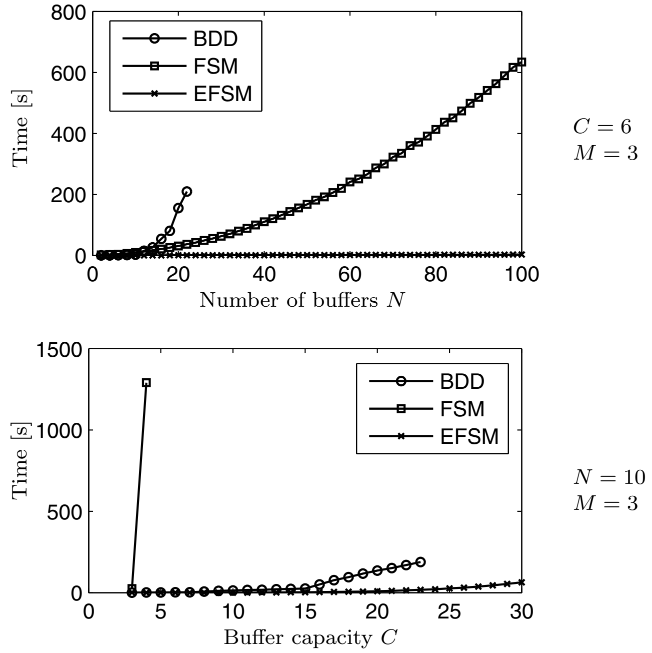

Fig. 6. Runtimes for Parallel Manufacturing Lines.

The EFSM-based compositional algorithm handles the growth in both parameters well. As the model has a simpler structure than the Transfer Line, it only needs to compose very few EFSMs at a time, and the problems associated with large event numbers are avoided thanks to partial unfolding and update simplification.

6. CONCLUSIONS

A general framework for compositional nonblocking verification of discrete event systems modelled as extended finite-state machines (EFSM) is presented. The framework makes it possible to apply the abstraction methods developed for ordinary finite-state machines to EFSMs, without the need to flatten the system and the associated overhead. The algorithm has been implemented and its performance compared with two other well-developed algorithms. The experimental results suggest that the EFSM-based algorithm can outperform FSM-based and BDD-based method for large systems with complex update formulas on their transitions.

Future work includes generalising the method for systems modelled as *extended finite-state automata*, which communicate via variables and shared events. In addition, extending the method to support supervisor synthesis (Ramadge and Wonham, 1989) for EFSMs is interesting.

REFERENCES

Åkesson, Knut, Martin Fabian, Hugo Flordal and Robi Malik (2006). Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*. Ann Arbor, MI, USA. pp. 384–385.

Baier, Christel and Joost-Pieter Katoen (2008). *Principles of Model Checking*. MIT Press.

Chen, Y. and F. Lin (2000). Modeling of discrete event systems using finite state machines with parameters. In: *Proc. 2000 IEEE Int. Conf. Control Applications (CCA)*. Anchorage, AK, USA. pp. 941–946.

Dams, Dennis, Orna Grumberg and Rob Gerth (1994). Abstract interpretation of reactive systems: Abstractions preserving ∀CTL*, ∃CTL* and CTL*. In: *Proc. IFIP WG2.1/WG2.2/WG2.3 Working Conf. Programming Concepts, Methods and Calculi (PROCOMET)* (E.-R. Olderog, Ed.). IFIP Transactions. Elsevier.

Flordal, Hugo and Robi Malik (2009). Compositional verification in supervisory control. *SIAM J. Control and Optimization* **48**(3), 1914–1938.

Graf, Susanne and Bernhard Steffen (1990). Compositional minimization of finite state systems. In: *Proc. 1990 Workshop on Computer-Aided Verification*. Vol. 531 of *LNCS*. Springer. pp. 186–196.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Malik, Robi and Ryan Leduc (2013). Compositional nonblocking verification using generalised nonblocking abstractions. *IEEE Trans. Autom. Control* **58**(8), 1–13.

Malik, Robi, David Streader and Steve Reeves (2006). Conflicts and fair testing. *Int. J. Found. Comput. Sci.* **17**(4), 797–813.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer.

Mohajerani, Sahar, Robi Malik and Martin Fabian (2013*a*). Compositional nonblocking verification for extended finite-state automata using partial unfolding. In: *Proc. 9th Int. Conf. Automation Science and Engineering, CASE 2013*. Madison, WI, USA. pp. 942–947.

Mohajerani, Sahar, Robi Malik and Martin Fabian (2013*b*). Partial unfolding for compositional nonblocking verification of extended finite-state machines. Working Paper 01/2013, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand.

Ramadge, Peter J. G. and W. Murray Wonham (1989). The control of discrete event systems. *Proc. IEEE* **77**(1), 81–98.

Sköldstam, M., K. Åkesson and M. Fabian (2007). Modeling of discrete event systems using finite automata with variables. In: *Proc. 46th IEEE Conf. Decision and Control, CDC '07*. pp. 3387–3392.

Su, Rong, Jan H. van Schuppen, Jacobus E. Rooda and Albert T. Hofkamp (2010). Nonconflict check by using sequential automaton abstractions based on weak observation equivalence. *Automatica* **46**(6), 968–978.

Teixeira, Marcelo, Robi Malik, José E. R. Cury and Max H. de Queiroz (2013). Variable abstraction and approximations in supervisory control synthesis. In: *2013 American Control Conf.*. Washington, DC, USA. pp. 120–125.

Yang, Y. and R. Gohari (2005). Embedded supervisory control of discrete-event systems. In: *Proc. 1st Int. Conf. Automation Science and Engineering, CASE 2005*. Edmonton, AB, Canada. pp. 410–415.

Zhaoa, Junhui, Yi-Liang Chen, Zhong Chen, Feng Lin, Caisheng Wang and Hongwei Zhang (2012). Modeling and control of discrete event systems using finite state machines with variables and their applications in power grids. *Systems & Control Letters* **61**(1), 212–222.