VERIFICATION OF SUPERVISORY CONTROL PROPERTIES OF FINITE AUTOMATA EXTENDED WITH VARIABLES

ALEXEY VORONOV, KNUT ÅKESSON

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden
ISSN 1403-266X R003/2009

Abstract

Verification and synthesis of control logic programs using Supervisory Control Theory (SCT) is an important topic. Most SCT methods are based on finite state automata (FA). FA extended with variables (EFA) are a compact, but otherwise equivalent to FA notation, proven to be beneficial in modeling control logic systems. To use existing SCT methods with EFA, it is necessary to convert EFA to FA. In certain cases this conversion can be very time-consuming, even if the number of resulting reachable states is very small compared to the total state-set of the system. In this paper we present a way to do verification of SCT properties of EFA models without converting them to FA. Instead, we convert them to the models for Symbolic Model Verification tool NuSMV. The conversion is performed in polynomial time. Experimental results show that NuSMV effectively utilizes small reachable state-set of the system to do verification.

***Index Terms*—Formal verification, supervisory control, finite automata, manufacturing systems, control logic, model checking.**

I. Introduction

Supervisory control theory [1] supports a user in the process of generating correct control function. In SCT, Supervisor is used to enable/disable events that may occur in the Plant. Plant generates all events. Supervisors can be synthesized automatically from the specification or can be created manually.

Finite state automata are usually used as a modeling formalism in SCT. In many cases it is cumbersome to use finite automata to model signal-based manufacturing systems [2]–[4]. Various extensions were introduced to simplify modeling (for example [5]–[7]), many of which were inspired by Statecharts [8]. In this paper we will look at finite automata extended with variables [9]. They provide a convenient way to create the models, while still maintaining a possibility to convert them to ordinary finite state automata to use existing SCT algorithms. Here arises a problem: the conversion can take exponential time, even if the resulting state-space of the system is small. This paper describes approach to verification of automata extended with variables without converting them to ordinary automata.

Model checking [10] is a well-established area. It is used for verification of software and hardware systems. Good results were achieved in the area, and good end-user tools exist [11]–[13]. Most often, symbolic representation is used for verification instead of explicit representation. In this paper we reduce supervisory verification problem of automata extended with variables to the form that is suitable for symbolic model verification tool NuSMV.

The rest of the paper is organized as following. Section II gives formal definitions of automata extended with variables and their supervisory control properties. Section III defines NuSMV input model. Section IV describes the conversion algorithm, and Section V illustrates the algorithm with an example. The paper is concluded with experimental results and discussion.

II. Preliminaries

Extended Finite Automaton (EFA) is a six-tuple:

$$E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0) \rangle \tag{1}$$

where

- $Q \times V$ is the extended finite set of states, where Q is a finite set of locations and $V = V^1 \times V^2 \times ... \times V^n$ is the finite domain of definition of the variables $v = (v^1, v^2, ..., v^n)$;
- Σ is a nonempty finite set of events (the alphabet);
- $\rightarrow \subseteq Q \times V \times \Sigma \times G \times A \times Q$ is a transition relation, where G is a set of guards, and guard is a predicate on V, $A = \{\ a \mid a \text{ is a function from } V \text{ to } V\ \}$;
- q_0 is an initial location and v_0 are initial values of the variables.

Transition relation $(p, v, \sigma, g, a, q) \in \rightarrow$ means that there is a transition from state (p, v) to state $(q, a(v))$ labeled by event σ if $g(v)$ evaluates to 1 (true).

For convenience we use the symbol Ξ to denote implicit actions that update variables to their current value. Unlike explicit actions, Ξ can be overriden when EFA are synchronized.

It is possible to write transition relation $\rightarrow$ in infix form: $p \xrightarrow{\sigma}_{g/a} q$. If g is absent it assumed that it always evaluates to true. If a is absent, it is assumed that $a = (\Xi, \Xi, ...\Xi)$.

A. Parallel Synchronous Composition, EFA

To simplify the notation when dealing with the parallel synchronous composition of EFA, we assume that the EFA share all variables. This is no restriction since it is always possible to add don't care variables that are never updated. For the synchronous composition to exist, a necessary and sufficient condition is that shared variables must have the same initial values. The composition operator models that an event can occur in the synchronized system if and only if it can occur in all EFA that share the event.

Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0) \rangle$, $k = 1, 2$ be two EFA with the shared variables $v = (v^1, ..., v^n)$. The parallel synchronous composition of E_1 and E_2 is

$$E_1 || E_2 = \langle Q_1 \times Q_2 \times V, \Sigma_1 \cup \Sigma_2, \rightarrow, (q_0^1, q_0^2, v_0) \rangle, \tag{2}$$

where the state transition relation $\rightarrow$ is defined as

- $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \bigcap \Sigma_2$ if $\exists (p_1, v_1, \sigma, g_1, a_1, q_1) \in \rightarrow_1$, $\exists (p_2, v_2, \sigma, g_2, a_2, q_2) \in \rightarrow_2$ such that $g = g_1 \wedge g_2$ and for $i = 1...n$ and $\forall v \in V$:

$$a^i(v) = \begin{cases} a_1^i(v) & \text{if } a_1^i(v) = a_2^i(v) \\ a_1^i(v) & \text{if } a_2^i(v) = \Xi \\ a_2^i(v) & \text{if } a_1^i(v) = \Xi \\ v^i & \text{otherwise} \end{cases} \tag{3}$$

- $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \setminus \Sigma_2$ if $(p_1, \sigma, g, a, q_1) \in \rightarrow_1$ and $p_2 = q_2$;
- $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_2 \setminus \Sigma_1$ if $(p_2, \sigma, g, a, q_2) \in \rightarrow_2$ and $p_1 = q_1$.

Note that if the action function of E_1 and E_2 explicitly tries to update a shared variable to different values, the variable is not updated. This implies that the synchronized EFA may not have the intended behavior. Sufficient condition to avoid this possibility is given in [9]. This condition is a global requirement and may be computational expensive to compute.

B. Supervisory control properties

Let $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ be an extended finite automaton and $Q^m \times V^m \subseteq Q \times V$ be a set of *marked* states of the automaton. EFA is called *non-blocking* if

$$\forall (q,v) \in (Q \times V) : \exists s \in \Sigma^*, \\ \exists (q^m, v^m) \in (Q^m \times V^m).(q,v) \xrightarrow{s} (q^m, v^m) \quad (4)$$

In words, the system is non-blocking if from any state there is a path to some marked state.

Let $G = (Q \times V, \Sigma, \rightarrow, (q_0, v_0)$ be an EFA. The active event function $\Gamma : Q \times V \rightarrow 2^{\Sigma}$ of G is defined as

$$\Gamma(p,v) = \{\sigma \in \Sigma | (p,v) \xrightarrow{\sigma}\}. \quad (5)$$

Let G and K be two EFA using shared variables with domain V. Let Σ_u be the set of uncontrollable events and Σ_K be the alphabet of K. A state $(p_G, p_K, v) \in Q_G \times Q_K \times V$ in the synchronized automaton $G||K$, is controllable if the following statement holds:

$$\Sigma_K \cap \Sigma_u \cap \Gamma(p_G, v) \subseteq \Gamma(p_K, v). \quad (6)$$

Uncontrollable states are the states of $G||K$ where G allows an uncontrollable event but K disables the same event via the synchronization.

Let G be a plant and K be a specification and Σ_u be the set of uncontrollable events. K is controllable with respect to G and Σ_u iff all reachable states of $G||K$ are controllable.

III. NuSMV

NuSMV [12] is a tool for symbolic model verification. It is mature tool with numerous users and proven efficiency, and it is open source.

We will represent NuSMV program as a tuple:

$$P = \langle S, InpVar, D, init, t, C\rangle \quad (7)$$

where

- $S = S_1 \times S_2 \times ... \times S_m$ is a set of finite domains of definition of state-keeping variables $s_1 ... s_m$;
- $InpVar$ is a set of domains of input variables; input variables have no state and on each step can take any value that satisfies *transition constraints*;
- D is a set of expressions that specify "define identifiers" or non state-keeping variables, that can be considered as macro definitions;
- $init$ is a predicate that describes an initial state;
- t is a predicate that describes transition constraints;
- C is a set of Computation Tree Logic (CTL) expressions of specifications.

NuSMV has two types of expressions. *Simple expressions* are ordinary arithmetic and logical expressions. *Next expressions* relate current and next value of state-keeping variables. We will denote next value of a variable s by $next(s)$. We will use the following simple expressions: $==$, $\neq$, $\wedge$, $\vee$, and $+$, for equality, inequality, conjunction, disjunction and sum, respectively. One more simple expression is $case$ expression. Case expression consists of a list of pairs of predicates and simple expressions. Starting from the beginning of the list, elements are taken one-by-one until predicate that evaluates to true will be found. Second element (expression) of the pair that contains this predicate is taken as a value of the case expression. Define declarations create synonims for expressions. We will write $v := e$ to denote that name v is a macro definition for expression e.

To specify properties that are necessary to verify, NuSMV accepts Computation Tree Logic (CTL) formulas. CTL is a branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized.

We will use the following two constructs from CTL:

- AG φ - All paths Globally: φ has to hold on all paths starting from the current state and to hold on the entire subsequent path.
- EF φ - Exists path where Finally: there exists at least one path starting from the current state where φ eventually has to hold.

IV. Conversion from EFA to NuSMV

Let $E_1 ... E_l$ be l EFA with n shared variables $V^1 ... V^n$. Then NuSMV will require $m = l + n$ state-keeping variables to represent it.

$$S_i = Q_i, i = 1..l \quad (8)$$

$$U_i = V_i, i = 1..n \quad (9)$$

Events in the system can be represented by one input variable $event$ with the domain $\bigcup_{k=1..l} \Sigma_k$:

$$InpVar = \{event\} \quad (10)$$

Initial predicate will be just a conjunction of expressions stating that the state variables should have values that correspond to the initial locations of the automata and initial values of the EFA variables:

$$init = \left(\bigwedge_{i=1..l} (s_i == q_0^i) \right) \wedge \left(\bigwedge_{i=1..n} (u_i == v_0^i) \right) \quad (11)$$

For each transition, it is possible to define one non-state keeping variable, that will be true iff the transition is taken at the current step. List of all definitions for transitions of automaton k, denoted by $transitionsTaken_k$, can be defined as following:

$$transitionsTakenDef_k = \\ \{tranIsTaken_k^j := (s == p) \wedge (event == \sigma) \wedge g \\ | \ (p, \sigma, g, a, q)_k^j \in \rightarrow_k\} \quad (12)$$

where $\rightarrow_k$ denotes transition relation for automaton k.

During synchronization there could be a situation when event do not belong to the alphabet of the automaton, and the automaton should keep its location. We will add extra variables that will indicate this situations:

$$stayIsTakenDef_k = \left(stayIsTaken_k := \bigwedge_{e \in \Sigma_k} event \neq e \right) \quad (13)$$

Transition constraint expression specifies that either one of the transitions is taken and automaton changes its location to the target state of the transition, or that automaton should keep its location. This can be defined with the help of macro definitions as following:

$$tranConstr_k = \left(\bigvee_{(p,\sigma,g,a,q)^j \in \rightarrow_k} tranIsTaken_k^j \wedge (next(s_k) == q) \right) \vee (stayIsTaken_k \wedge (next(s_k) == s_k)) \quad (14)$$

It is necessary to specify that on each step each automaton take only one transition. Boolean variable can be seen as an integer variable, with 1 indicating true and 0 indicating false. Having that, it is possible to do summation over them. Condition is that the sum of all transitions that are taken at the current step and indication to keep the location should be exactly one. (In the following equation only, symbol Σ will stand for the mathematical summation and not for the automaton alphabet).

$$singleTranConstr_k = 1 == \left(stayIsTaken_k + \sum_j tranIsTaken_k^j \right) \quad (15)$$

Updates of the EFA variables are more complicated than the ones for automata locations, since variable can have *don't care* updates. With such an update, the EFA variable should keep its value, but if some other automaton overrides this due to the synchronization, the variable should take the new value. That is why it is necessary to check on each step if there is an automaton that "cares" about the update of the variable. It is possible to introduce one define-variable for each pair of automaton and EFA variable that will indicate that the automaton E_k *cares* about the update of the variable v_i on the current step. We will call this variable $desire^{i,k}$. This variable can take value 0 to indicate $don't\ care$, and 1 to indicate that the automaton wants to update the variable on the current step. We will reserve value 2 to indicate conflicts.

$$desireDef^{i,k} = (desire^{i,k} := case(\{(tranIsTaken_k^j, 1) \mid (p,\sigma,g,a,q)_k^j \in \rightarrow_k, a \neq \Xi\} \cup \{(true, 0)\})) \quad (16)$$

Note that we added one element to the end of the case expression that covers all don't care updates and stay transitions.

It is also necessary to have variables for desired values, which can be defined similarly:

$$desiredValDef^{i,k} = (desiredVal^{i,k} := case(\{(tranIsTaken_k^j, a(u_i)) \mid (p,\sigma,g,a,q)_k^j \in \rightarrow_k, a \neq \Xi\} \cup \{(true, u_i)\})) \quad (17)$$

Having desires and desired values for all pairs of automata and EFA variables, it is necessary to combine all desires to get a single one for each EFA variable. Rule for combination of two desires is simple. If none of automata cares about updating, result is $don't\ care$. If only one of automata cares, the result is $care$. If both automata wants to update EFA variable to the same value, the result is $care$. And if two automata wants to update the variable to two different values, the result is $conflict$.

We define a function that will map three pairs of desire and desired value variables to two macro definitions. This will specify that the third pair of desire and desired value is a combination of the first two desires and desired values:

$$\begin{aligned} &combine((d_1, dv_1), (d_2, dv_2), (d_r, dv_r)) = \\ &d_r := case(\{ \\ &(d_1 == 0, d_2), \\ &(d_2 == 0, d_1), \\ &(d_1 == 2 \vee d_2 == 2, 2), \\ &(d_1 == 1 \wedge d_2 == 1, case(\{(dv_1 \neq dv_2, 2), \\ &\qquad (dv_1 == dv_2, 1)\}) \\ &\}) \\ &\cup \\ &dv_r := case(\{(d_r == 0 \vee d_r == 2, \Xi), \\ &\qquad (d_r == 1, case(\{(d_1 == 1, dv_1), \\ &\qquad\qquad (d_2 == 1, dv_2)\}))\}) \end{aligned} \quad (18)$$

Here we used Ξ to indicate that the value should not be changed. In programming language we can model Ξ as Nil or $null$, since it will not be necessary to check for the desired value if the desire is different from $care$.

Using this rule, we will combine desires for all automata that are related to the EFA variable. A set of all automata that are related to the EFA variable v_i can be defined as following:

$$relatedAutomata^i = \{E | E \in E_1..E_n, \exists (p,\sigma,g,a,q) \in \rightarrow_k : a(v) \neq \Xi\} \quad (19)$$

Before we can specify a combination of desires, we will need two more functions. One will map a pair of EFA variable and automaton to the pair of desire and desired value variables for them:

$$varPair(v_i, automaton_k) = (desire^{i,k}, desiredVal^{i,k})$$

Another function will map a pair of an EFA variable and a list of automata to a pair of desire and desired value variables of that list concerning the variable:

$varPairForList(v_i, L) = (desire^{i,L}, desiredVal^{i,L})$.

Now it is possible to specify a function that will map a pair of a list of automata and a variable to another pair. Resulting pair will contain a pair of desire and desired value for the list, and a list of definitions of extra variables.

$$combM(L, v_i) = \begin{cases} (pair_r, vars_r), where \\ \quad pair_r = varPairForList(v_i, L), \\ \quad pair_{head} = varPair(v_i, head(L)), \\ \quad pair_{tail} = varPair(v_i, head(tail(L)), \\ \quad vars_r = combine(pair_{head}, pair_{tail}, pair_r), \\ \qquad\qquad if\ length(L) = 2 \\ (pair_r, vars_r), where \\ \quad pair_r = varPairForList(v_i, L), \\ \quad pair_{head} = varPair(v_i, head(L)), \\ \quad (pair_{tail}, vars_{tail}) = combM(tail(L), v_i), \\ \quad vars_r = combine(pair_{head}, pair_{tail}, pair_r) \cup vars_{tail}, \\ \qquad\qquad if\ length(L) > 2 \end{cases} \tag{20}$$

where $head$ stands for the first element of the list, and $tail$ stands for the rest of the list without the first element.

$$tail(L) = L \setminus head(L) \tag{21}$$

Then resulting desire and desired value for variable v_i can be obtained by the following formula:

$$((desire^i, desiredVal^i), extraVarsDef^i) = \begin{cases} combM(relatedAutomata^i, v_i), \\ \qquad if\ length(relatedAutomata^i) \geq 2 \\ (varPair(head(relatedAutomata^i)), \emptyset), \\ \qquad if\ length(relatedAutomata^i) = 1 \\ ((0, u_i), \emptyset) \\ \qquad if\ length(relatedAutomata^i) = 0 \end{cases} \tag{22}$$

Final transition constraint for the variable v_i have to be adjusted to deal properly with conflicts:

$$tranConstrForVar_i = next(u_i) == case(\{ (desire^i == 0 \vee desire^i == 2, u_i), (desire^i == 1, desiredVal^i)\}) \tag{23}$$

Complete predicate for transition constraints will combine all constraints for automata transitions and for EFA variables updates:

$$t = \bigwedge_{k=1..l} (transConstr_k \wedge singleTranConstr_k) \wedge \bigwedge_{i=1..n} transConstrForVar_i \tag{24}$$

Complete set of define variables will contain definitions of all transitions, including stay transitions, and all desire and desired value variables:

$$D = \bigcup_{k=1..l} (transitionsTakenDef_k \cup stayIsTakenDef_k) \cup \bigcup_{i=1..n, k=1..l} desireDef^{i,k} \cup \bigcup_{i=1..n, k=1..l} desiredValDef^{i,k} \cup \bigcup_{i=1..n} extraVarsDef^i \tag{25}$$

To specify non-blocking, the following CTL formula can be used:

$$AG\ EF\ \left(\bigwedge_{i=1..n} \bigvee_{q^m \in Q_i^m} s_i == q^m \right) \wedge \left(\bigwedge_{i=1..l} \bigvee_{v^m \in V_i^m} u_i == v^m \right) \tag{26}$$

The formula specifies that on all pathes eventually the system will have a state that is marked.

To specify controllability it is necessary to introduce extra define-variables. For each event for each automaton this variable will be true if the event is enabled, and false otherwise:

$$enabled_\sigma^k := \begin{cases} case(\{((s_i == p) \wedge g, 1) | (p, \sigma, g, a, q) \in \rightarrow_k\} \cup \{(1, 0)\}) \\ \qquad\qquad if\ \sigma \in \Sigma_k \\ 1\ otherwise \end{cases} \tag{27}$$

Then controllability can be expressed as following: for all uncontrollable events, if it is enabled in all plants automata, it have to be enabled in all specification automata. In other words, each uncontrollable event should be either disabled in any of the plant automata or enabled in all specification automata. Let $E^{plants}, E^{specs} \subseteq E_1 ... E_l$ be two disjoint sets of plants automata and specifications automata. Let $\Sigma_u \subseteq \bigcup_{i=1..l} \Sigma_i$ be a set of all uncontrollable events. Then the property that E^{specs} are controllable with respect to E^{plants} and Σ_u can be expressed as following:

$$AG \bigwedge_{\sigma \in \Sigma_u} (\bigvee_{E_i \in Plants} (enabled_\sigma^i == 0) \vee \bigwedge_{E_j \in Specs} (enabled_\sigma^j == 1)) \tag{28}$$

V. Example conversion

To illustrate the conversion we will use two automata that share one variable, see Fig. 1. Automaton E_1 have locations

$\{q_{11}, q_{12}\}$ with initial location q_{11}. Automaton E_2 have locations $\{q_{21}, q_{22}\}$ with initial location q_{21}. Variable v can take values 0 or 1 and have initial value 0.

Alphabet of automaton E_1 is $\Sigma_1 = \{a, b\}$. Automaton has transitions $q_{11} \xrightarrow{a}_{v==0/v:=1} q_{12}$ and $q_{12} \xrightarrow{b} q_{11}$. Alphabet of automaton E_2 is $\Sigma_2 = \{a, c\}$. Automaton has transitions $q_{21} \xrightarrow{a} q_{22}$ and $q_{22} \xrightarrow{c}_{true/v:=0} q_{21}$.

Initial predicate will be as following:

$$init = (s_1 == q_{11}) \wedge (s_2 == q_{21}) \wedge (v == 0) \quad (29)$$

Define-variables for transitions for each automaton will be:

$$\begin{aligned} &transitionsTakenDef_1 = \{ \\ &\quad tranIsTaken_1^1 := (s == q_{11}) \wedge (event == a) \wedge \\ &\qquad\qquad \wedge (v == 0), \\ &\quad tranIsTaken_1^2 := (s == q_{12}) \wedge (event == b)\} \end{aligned} \quad (30)$$

$$\begin{aligned} &transitionsTakenDef_2 = \{ \\ &\quad tranIsTaken_2^1 := (s == q_{21}) \wedge (event == a), \\ &\quad tranIsTaken_2^2 := (s == q_{21}) \wedge (event == c)\} \end{aligned} \quad (31)$$

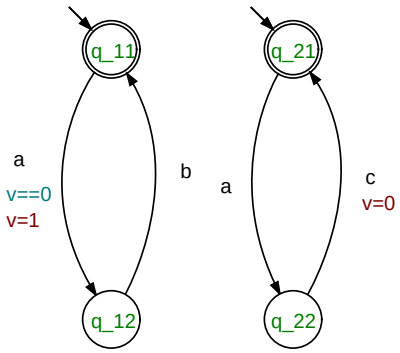
Definition of stay variable will be as following:

$$\begin{aligned} &stayIsTakenDef_1 = \\ &\quad \{stayIsTaken_1 := (event \neq a) \wedge (evnet \neq b)\} \end{aligned} \quad (32)$$

$$\begin{aligned} &stayIsTakenDef_2 = \\ &\quad \{stayIsTaken_2 := (event \neq a) \wedge (evnet \neq c)\} \end{aligned} \quad (33)$$

Transition constraints will be:

$$\begin{aligned} tranConstr_1 = &\big(tranIsTaken_1^1 \wedge (next(s_1) == q_{12})\big) \\ &\vee \big(tranIsTaken_1^2 \wedge (next(s_1) == q_{11})\big) \\ &\vee (stayIsTaken_1 \wedge (next(s_1) == s_1)) \end{aligned} \quad (34)$$



(a) Automaton E_1 (b) Automaton E_2

Fig. 1. Example automata

$$\begin{aligned} tranConstr_2 = &\big(tranIsTaken_2^1 \wedge (next(s_2) == q_{22})\big) \\ &\vee \big(tranIsTaken_2^2 \wedge (next(s_2) == q_{21})\big) \\ &\vee (stayIsTaken_2 \wedge (next(s_2) == s_2)) \end{aligned} \quad (35)$$

Single transition constraints:

$$\begin{aligned} &singleTranConstr_1 = \\ &(tranIsTaken_1^1 + tranIsTaken_1^2 + stayIsTaken_1 == 1) \end{aligned} \quad (36)$$

$$\begin{aligned} &singleTranConstr_2 = \\ &(tranIsTaken_2^1 + tranIsTaken_2^2 + stayIsTaken_2 == 1) \end{aligned} \quad (37)$$

Desire definitions for automaton E_1 will contain only two elements, since only one transition modifies the variable. Desired value during update is set to 1 :

$$\begin{aligned} desireDef^{v,E_1} = (&desire^{v,E_1} := case(\{ \\ &(tranIsTaken_1^1, 1), (true, 0)\})) \end{aligned} \quad (38)$$

$$\begin{aligned} desiredValDef^{v,E_1} = (&desiredVal^{v,E_1} := case(\{ \\ &(tranIsTaken_1^1, 1), (true, v)\})) \end{aligned} \quad (39)$$

$$\begin{aligned} desireDef^{v,E_2} = (&desire^{v,E_2} := case(\{ \\ &(tranIsTaken_2^2, 1), (true, 0)\})) \end{aligned} \quad (40)$$

$$\begin{aligned} desiredValDef^{v,E_2} = (&desiredVal^{v,E_2} := case(\{ \\ &(tranIsTaken_2^2, 0), (true, v)\})) \end{aligned} \quad (41)$$

Both automata are related to the variable, that is why $relatedAutomata^v = \{E_1, E_2\}$. Combined desire and desired value for the variable, as well as extra definitions, could be obtained as following:

$$
\begin{aligned}
&((desire^{v}, desiredVal^{v}), extraVarsDef^{v}) = \\
&= combM(\{E_1, E_2\}, v) \\
&= (pair_r, vars_r) \\
&= (varPairForList(v, \{E_1, E_2\}), \\
&combine(varPair(v, E_1), varPair(v, E_2), \\
&varPairForList(v, \{E_1, E_2\}))) \\
&= ((desire^{v,\{E_1,E_2\}}, desiredVal^{v,\{E_1,E_2\}}), \\
&combine((desire^{v,E_1}, desiredVal^{v,E_1}), \\
&(desire^{v,E_2}, desiredVal^{v,E_2}), \\
&(desire^{v,\{E_1,E_2\}}, desiredVal^{v,\{E_1,E_2\}}))) \\
&= ((desire^{v,\{E_1,E_2\}}, desiredVal^{v,\{E_1,E_2\}}), \\
&\{desire^{v,\{E_1,E_2\}} := case(\{(desire^{v,E_1} == 0, desire^{v,E_2}), \\
&(desire^{v,E_2} == 0, desire^{v,E_1}), \\
&(desire^{v,E_1} == 2 \vee desire^{v,E_2} == 2, 2) \\
&(desire^{v,E_1} == 1 \wedge desire^{v,E_2} == 1, case(\{ \\
&(desiredVal^{v,E_1} \neq desiredVal^{v,E_2}, 2), \\
&(desiredVal^{v,E_1} == desiredVal^{v,E_2}, 1)\}))\}), \\
&desiredVal^{v,\{E_1,E_2\}} := case(\{ \\
&(desire^{v,\{E_1,E_2\}} == 0 \vee desire^{v,\{E_1,E_2\}} == 2, v), \\
&(desire^{v,\{E_1,E_2\}} == 1, desiredVal^{v,E_1})\})\}) \quad (42)
\end{aligned}
$$

From this direct calculation, we obtained definitions of two extra variables, $desire^{v,\{E_1,E_2\}}, desiredVal^{v,\{E_1,E_2\}}$, and found out that this variables should be used in the transition constraint for the variable:

$$
\begin{aligned}
&tranConstrForVar_i = (next(v) == case(\{ \\
&(desire^{v,\{E_1,E_2\}} == 0 \vee desire^{v,\{E_1,E_2\}} == 2, v), \\
&(desire^{v,\{E_1,E_2\}} == 1, desiredVal^{v,\{E_1,E_2\}})\})) \quad (43)
\end{aligned}
$$

Automaton E_1 has location q_{11} as a marked location, and automaton E_2 have q_{21} marked. Then non-blocking property can be specified as following:

$$
AG\ EF\ (s_1 == q_{11} \wedge s_2 == q_{21}) \quad (44)
$$

VI. Experimental Results

The method was implemented and compared with the conversion of EFA to ordinary automata and verification. Example was taken from IEC 61499 program verification. EFA model consisted of 112 automata and variables. Conversion to ordinary automata took more than 15 minutes on Pentium IV 2.4 GHz with 2 GB RAM. The system had 10^8 potential states, but only 249 reachable states. Verification of ordinary automata system was done in a fraction of a second. With the proposed method conversion from EFA model to NuSMV model was done in less than a second, and NuSMV verification was done in a fraction of second again. This shows that on some examples there is a few orders of magnitude speed-up.

VII. Conclusions

Proposed method gives a possibility to convert expressive EFA models to the models that can be verified by the state-of-the-art symbolic model verification tools. This conversion can be done much more efficiently than the conversion to ordinary finite state automata. Possible disadvantage of this approach is that it is not possible to do supervisory synthesis on NuSMV models that was possible with ordinary automata.

References

[1] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] S. Balemi, G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin, "Supervisory control of a rapid thermal multiprocessor," *Automatic Control, IEEE Transactions on*, vol. 38, no. 7, pp. 1040–1059, July 1993.

[3] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete eventsystems," in *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, vol. 3, Tampa, FL, USA, 1998, pp. 3305–3310.

[4] X.-R. Cao, G. Cohen, A. Giua, W. M. Wonham, and J. H. van Schuppen, "Unity in diversity, diversity in unity: Retrospective and prospective views on control of discrete event systems," *Discrete Event Dynamic Systems*, vol. 12, no. 3, pp. 253–264, 2002.

[5] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *Proceedings of the 2000 IEEE International Conference on Control Applications*, Anchorage, Alaska, USA, September 2000.

[6] Y. Yang and P. Gohari, "Embedded supervisory control of discrete-event systems," in *Proceedings of the 2005 IEEE international conference on automation Science and engineering*, Edmonth, Canada, August 2005.

[7] B. Gaudin and P. Deussen, "Supervisory control on concurrent discrete event systems with variables," in *American Control Conference, 2007. ACC '07*, July 2007, pp. 4274–4279.

[8] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.

[9] M. Sköldstam, K. kesson, and M. Fabian, "Modeling of discrete event systems using automata with variables," in *Proceedings of the 46th IEEE Conference on Decision and Control*, 2008, pp. 3387–3392.

[10] E. Clarke and E. Emerson, "Synthesis of synchronization skeletons for branching time temporal logic," in *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1981, vol. 131.

[11] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on the*, 1990, pp. 428–439.

[12] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource tool for symbolic model checking," in *Proceeding of International Conference on Computer-Aided Verification (CAV 2002)*, Copenhagen, Denmark, July 2002.

[13] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Möller, P. Pettersson, C. Weise, and W. Yi, "UPPAAL: now, next, and future," in *Modeling and verification of parallel processes*, ser. Lecture Notes In Computer Science. New York, NY, USA: Springer-Verlag New York, Inc., 2001, pp. 99–124.