Incremental System Verification and Synthesis of Minimally Restrictive Behaviours

B. Brandin and R. Malik
Software and Engineering
Siemens Corporate Research, ZT SE 4
81730 München, Germany

P. Dietrich
Fachbereich Informatik
Universität Kaiserslautern
67653 Kaiserslautern, Germany

Abstract

An incremental approach to system verification is proposed, for system behaviours and safety properties described by means of finite-string languages and finite-state automata. Properties are verified with respect to subsystems of the overall system, nevertheless allowing assertions to be made about the entire system satisfying such properties. The proposed approach considers satisfaction of properties, controllability, and synthesis as successive verification steps. Furthermore, it allows the incremental augmentation of the system to be verified: after each verification step, either the desired property is verified, or a counter example is obtained, which, together with heuristics, provides the basis for the augmentation of a given subsystem for the next verification step.

1 Introduction

The aim of system verification is to check whether a given system behaviour satisfies certain *properties*[1] [6, 9]. Typically, such a system behaviour consists of assumptions about an environment under control and of a control program acting upon this environment. Both parts of such a system behaviour can be conveniently described by means of finite-state automata.

Often, system behaviours to be verified are not described fully, but only specific behavioural aspects are considered. This is where the idea of *incremental verification* comes into play. Assume given a system behaviour and a property to be verified. Further assume that the system behaviour cannot be shown to satisfy the property. This could be for two reasons, either the system considered does not satisfy the property, or the system considered does satisfy the property, but the description of its behaviour is too incomplete to allow the property to be shown to be satisfied.

Two questions then arise: can the description of the system behaviour be completed somehow to show whether the property is satisfied or not, and if so, how?

[1]Properties often are also referred to as *requirements* or *commitments*.

At this point, we may use results from *supervisory control theory* [5, 10]. Supervisory control theory allows, given a physically possible system behaviour and a desirable system behaviour, i.e. a desired property, to check whether candidate supervisory control strategies guarantee the desired behaviour, or to synthesize minimally restrictive supervisory control strategies. Using such control theoretic results, it is possible to (a) determine whether given additional system behaviours coupled with the initially given system behaviours, guarantee that the desired property is always satisfied, or (b) synthesize additional system behaviours which, coupled with the initially given system behaviours, can guarantee that the property is satisfied.

In this paper, only the verification of so-called *safety* properties is considered. Such properties can be described modularly by automata or sets of automata states. In this case, the results presented herein provide a powerful verification setting, which allows system behaviours to be incrementally augmented in order to verify whether given behavioural requirements are satisfied, thus allowing for very efficient computation.

Based on the above, an incremental procedure is presented which considers the notions of property satisfaction, controllability and synthesis as successive verification steps. After each step, either the given behavioural requirements are verified or counter examples are obtained, which provide the basis, together with automatically implementable heuristics, for the augmentation of given subsystems for verification purposes.

The following references present a number of approaches to compositional verification and synthesis. Halbwachs [8] discusses the application of *synchronous observers* for the modular verification of safety properties, and in his synchronous framework, obtains similar theoretical results as those discussed herein within the discrete-event systems framework of [11]. An alternative approach to the simplification of large systems of automata for proving purposes is discussed in [7], using user-supplied *interface specifications*, and in [2]. The synthesis of maximally permissible behaviours of subsystems has also been suggested in [1, 12]. The use of counter examples for the model-checking of ADA programs has also been suggested in [3].

2 Preliminaries

The results presented herein stem from the *supervisory control theory* for *discrete-event systems (DES)* [11]. These are dynamic systems that evolve in accordance with the abrupt occurrence of physical events. Such systems generally encompass processes that are discrete in time and state space, often asynchronous, and typically nondeterministic. To ensure the orderly occurrence of events in such systems, i.e. according to given behavioural specifications, some degree of supervision and control is generally required [5, 11].

Discrete-event systems are controlled as generators of a formal language. The adjunction of a control structure allows varying the language generated by the system within certain limits by accordingly enabling and disabling events. The desired behaviour, i.e. the desired properties, of such controlled generators is specified by stating that their generated language must belong to some specification language [5, 11].

Two main design approaches are possible, either candidate supervisory control strategies are checked to be strategies which guarantee that given properties are never violated, or minimally restrictive supervisory control strategies are synthesized which also guarantee that the properties are never violated.

The reader is referred to [5, 10, 11, 13] for notation and background literature.

3 Incremental Verification

We present here the notion of incremental verification by considering a system G composed of automata $G_i, i \in I$, for some index set I, and a behavioural requirement R to be satisfied by G, and composed of automata $R_k, k \in K$, for some index set K.

3.1 Basic Notions

We will say a system G *satisfies* the requirements embodied by R, or simply that G satisfies R, if $L(G) \subseteq L(R)$. Thus, G satisfies R, if every string of events accepted by the automaton G is also accepted by the automaton R. Since we only consider prefix-closed languages, we can say equivalently that G satisfies R if $\mathrm{Elig}_{L(G)}(s) \subseteq \mathrm{Elig}_{L(R)}(s)$, for all $s \in L(R)$, where $\mathrm{Elig}_{L(R)}(s)$ defines the set of *eligible events*[2] after the occurrence of s in $L(R)$.

Let the system G be composed by a number of subsystems: $G = \prod_{i \in I} G_i$, for some index set I. Let E constitute a subsystem of G composed by a number of G_i's, i.e. $E = \prod_{j \in J} G_j, J \subseteq I$. Given the modular structure of G, it becomes of interest, for computational reasons, to be able to verify whether given behavioural requirements R are satisfied by the subsystem E of G, and infer from the result whether the same requirements R are satisfied by the system G itself. In other words, we would like to define conditions under which the satisfaction by E of given requirements R implies the satisfaction of the same requirements R by G. We observe that subsystem composition always restricts and never enlarges the behaviour of the composed system E. Consequently, behavioural requirements satisfied by the subsystem E should also be satisfied by the complete system G.

Proposition 3.1 Let $G = \prod_{i \in I} G_i$, $E = \prod_{j \in J} G_j$, $J \subseteq I$, and R be automata sharing the alphabet Σ. If E satisfies R, then G satisfies R.

Thus, in order to show that the system G satisfies the constraints R, it is enough to show that a subsystem E of G satisfies R. If the requirement R is given as the synchronous product of several automata, i.e. $R = \prod_{k \in K} R_k$, then the above satisfaction check may be carried out sequentially by considering in turn each $R_k, k \in K$.

3.2 Implementable System Behaviours

System behaviours cannot always be shown to satisfy given behavioural requirements. This can be for two reasons: (i) the system considered does not satisfy the requirements, or (ii) the system considered does satisfy the requirements but the description of its behaviour is too incomplete to allow the requirements to be shown to be satisfied. In order to understand why behaviours do not satisfy given behavioural requirements, it becomes important to determine whether and which additional system behaviours, coupled with the initially given system behaviours, guarantee that the behavioural requirements are not violated. In other words, it becomes of interest to determine whether through some degree of supervision and control, given system behaviours can be modified to guarantee that given behavioural requirements are not violated.

The adjunction of a control structure to a discrete-event system allows varying the language generated by the system within certain limits by accordingly enabling and disabling *controllable* events[3]. The concept of language controllability [11] allows us to characterize exactly the languages which may be kept within another language by means of control. A language $L(R)$, embodying given required behaviours, is said to be controllable with respect to the language $L(G)$, embodying the behaviour of a system G, exactly when the behaviour of G can be kept within $L(R)$ through control, notwithstanding the possible occurrence of uncontrollable events.

More precisely, an automaton R is said to be *controllable* with respect to an automaton G, if $\mathrm{Elig}_{L(G)}(s) \cap$

[2] For $L \subseteq \Sigma^*$ and $s \in \Sigma^*$, define $\mathrm{Elig}_L(s) = \{\, \sigma \in \Sigma \mid s\sigma \in \overline{L} \,\}$.

[3] We distinguish *controllable* events, which can be disabled by control action, and *uncontrollable* events, which occur spontaneously and cannot be disabled [11].

$\Sigma_u \subseteq \mathrm{Elig}_{L(R)}(s)$, for all $s \in L(R)$, i.e. if the behaviour of R can never be exited by the occurrence of an uncontrollable event possible after s in $L(G)$ [11].

Thus, if R is controllable with respect to G, we can use R itself as an additional system which, coupled with G, guarantees that the behavioural requirements R are not violated.

Again, recall that subsystem composition always restricts and never enlarges the behaviour of the composed system. If through supervision and control, given behavioural requirements can be imposed on a subsystem E of G, then it should also be possible to impose these requirements on G itself through the same supervision and control means. The following result shows that, if behavioural requirements R are controllable for a subsystem E of G, these are also controllable for G.

Proposition 3.2 Let $G = \prod_{i\in I} G_i$, $E = \prod_{j\in J} G_j$, $J \subseteq I$, and R be automata sharing the alphabet Σ. If R is controllable with respect to E, then R is controllable with respect to G.

Proposition 3.3 Let $G = \prod_{i\in I} G_i$, R_1, and R_2 be automata sharing the alphabet Σ. Then $R_1 \times R_2$ is controllable with respect to G if R_1 and R_2 are controllable with respect to G.

Thus, the composition of two controllable properties with respect to G is itself controllable with respect to G. This result is particularly helpful since it shows that controllability checks may be carried out in sequence by considering in turn each $R_k, k \in K$, constituting R. Nevertheless, it is important to note that not all R_k may themselves be controllable with respect to G. Since the composition of an uncontrollable property with another may be itself controllable, the following corollary becomes of interest.

Corollary 3.4 Let $G = \prod_{i\in I} G_i$, $E = \prod_{h\in H} G_h, H \subseteq I$, $E' = \prod_{j\in J} G_j, J \subseteq I$, $P = \prod_{l\in L} R_l$, and $P' = \prod_{m\in M} R_m$ be automata sharing the alphabet Σ. If P is controllable with respect to E, and P' is controllable with respect to E', then $P \times P'$ is controllable with respect to G.

This is paraphrased by saying that given properties can be shown to be controllable with respect to G if these can be grouped into sets of properties, each set being controllable with respect to some subsystem of G.

3.3 Synthesis of Implementable Minimally Restrictive System Behaviours

In case the requirements R are not controllable with respect to G, it is possible to synthesize an additional system R' controllable with respect to G which coupled with G guarantees that $R' \times G$ satisfies R.

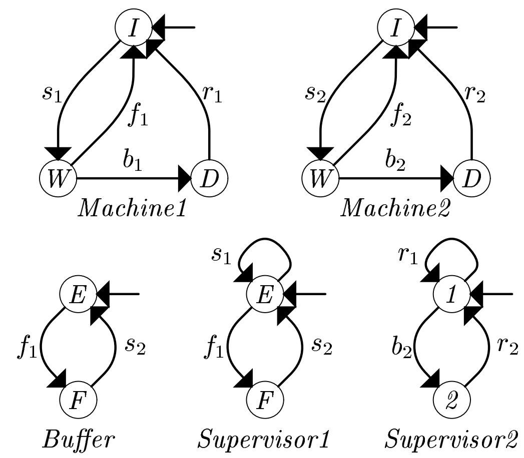

Figure 1: Small factory example.

Proposition 3.5 Let G and R be automata sharing the alphabet Σ. Let R' be such that $L(R') = \sup \mathcal{C}\,(L(R) \cap L(G))$. Then R' is controllable with respect to G, and $R' \times G$ satisfies R.

Here, $\sup \mathcal{C}\,(L(R) \cap L(G))$ represents the *supremal controllable sublanguage* of $L(R) \cap L(G)$, i.e. the largest possible controllable sublanguage in the intersection $L(R) \cap L(G)$ [11]. Therefore, R' can be thought of embodying the minimally restrictive behaviour of G satisfying R. Note that $L(R')$ may be empty, implying that there exists no additional system, controllable with respect to G, which is able to guarantee that $R' \times G$ satisfies R.

The following result shows that a synthesized system R' controllable with respect to a subsystem E of G is controllable with respect to the entire system G.

Proposition 3.6 Let $G = \prod_{i\in I} G_i$, $E = \prod_{j\in J} G_j$, $J \subseteq I$, and R be automata sharing the alphabet Σ. Let R' be such that $L(R') = \sup \mathcal{C}\,(L(R) \cap L(E))$. Then, R' is controllable with respect to G.

3.4 Behavioural Requirements

The setting proposed here only considers *safety* behavioural requirements, i.e. desired behaviours which must never be exited by the system considered.

For illustration purposes, a simple manufacturing system consisting of two machines and a buffer of size one is used. The parts of the system are modelled in form of automata (figure 1). *Machine1*, e.g., is initially in its idle state I. It may be put into operation by event s_1, entering its working state W. When working, the machine may finish operation (f_1) and return to its idle state, or it may break down (b_1) and be repaired (r_1) later. Whenever *Machine1* finishes operation, it places a workpiece into the *Buffer*, changing the state from empty (E) to full (F). Starting *Machine2* removes the piece from the buffer.

The machines' operation and repair has to be coordinated according to the following specifications. Firstly, the buffer should never overflow or underflow, i.e. *Machine1* may not finish operating while a workpiece is present in the buffer, and *Machine2* may not start operating unless a workpiece is present in the buffer. Secondly, *Machine2* has repair and return-to-service priority over *Machine1*, i.e. in case both machines are down, *Machine2* must be repaired and returned to service first. The two modular supervisory control strategies *Supervisor1* and *Supervisor2* are assumed to implement these specifications.

Language-Based Requirements. It is often possible and convenient to express behavioural requirements in language-form represented by a corresponding generating automaton. In the above manufacturing example, the requirement that the buffer does never overflow or underflow can easily be represented in this manner and is shown in figure 1 as *Buffer*. In order to verify that $G = Machine1 \times Machine2 \times Supervisor1 \times Supervisor2$ satisfies $R = Buffer$, it is enough to find a subsystem E of G satisfying R, e.g. $E = Supervisor1$, for which it can be checked that $L(E) \subseteq L(R)$.

State-Based Requirements. It is also often possible and convenient to express behavioural requirements in the form of a set of system states [4]. We consider the following behavioural requirement: *"the buffer must be empty (E) whenever Machine1 is working (W)"*. Note that this requirement only considers the states of different automata constituting the system G and is difficult to describe in language-form. Typically, *state predicates* describing the set of system states which satisfy the behavioural requirements, can be easily defined. For example, for the behavioural requirement above, a suitable state predicate could be given by $S = \{(q_1, q_2, q_3, q_4, q_5) \in \prod_{i=1}^{5} Q_i \mid \text{if } q_1 = W \text{ then } q_3 = E\}$, where Q_1 corresponds to the state set of *Machine1*, Q_2 corresponds to the state set of *Machine2*, Q_3 corresponds to the state set of *Buffer*, Q_4 corresponds to the state set of *Supervisor1*, and Q_5 corresponds to the state set of *Supervisor2*.

Such requirements are stated formally using a property automaton $E = (\Sigma, Q, \delta_E, q_0, Q_m)$, with event alphabet Σ, state set Q, transition function $\delta_E\colon Q \times \Sigma \to Q$, initial state $q_0 \in Q$, and marked state set $Q_m \subseteq Q$. Typically this is a subsystem $E = \prod_i G_i$ of G.

Now let $S \subseteq Q$ be a set of safe system states representing given behavioural requirements such that $q_0 \in S$. Intuitively, the state-based requirements embodied by S can be transformed into language-based requirements by constructing a subautomaton R of E, in which only the states of E, which are also in S, are reachable. Then, the language $L(R)$ of R characterizes in language-form the original state-based behavioural requirements. R is called the *legal subautomaton* of E with respect to S, and defined as $R = (\Sigma, Q, \delta_R, q_0, Q_m)$, where

$$\delta_R(q,\sigma) = \begin{cases} \delta_E(q,\sigma) & \text{if } \delta_E(q,\sigma) \in S \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Proposition 3.7 Let $E = (\Sigma, Q, \delta_E, q_0, Q_m)$ be an automaton. Let $S \subseteq Q$, and let R be the *legal subautomaton* of E with respect to S. Then we have $\delta_E(q_0, s) \in S$ for all $s \in L(E)$ if and only if $L(E) \subseteq L(R)$.

Thus, in order to check whether all reachable states of the system G satisfy the property given by S, it suffices to check language inclusion for the legal subautomaton R.

As an extension to the above, behavioural requirements may be expressed as a set $S \subseteq Q \times \Sigma$ of automaton states and events exiting such states. The intended semantics of such requirements is that, in each state of Q, only the events explicitly considered in S are allowed.

4 Application of the Theoretical Results

State-of-the-art verification tools do not only verify properties, but typically provide counter examples as diagnostic for the user in case the property is not satisfied. In the present setting, counter examples constitute the basis for incremental verification by allowing to judiciously augment a subsystem in order to satisfy a property of interest.

Now let E be a subsystem not satisfying a property R, and s be a counter example, i.e. a string of events violating R but accepted by E. Furthermore, let G_i be a component which accepts s. Then, E augmented with G_i, i.e. $E \times G_i$, will also accept s. Consequently, given a counter example string s not satisfying a property of interest, and a set G_i, $i \in I$, of candidate components to augment E, only the G_i's which do not accept s should be considered to augment E. E is iteratively checked to verify the property of interest, and augmented when needed: if the property is not verified, i.e. a counter example is found, E is augmented with components not accepting the counter example. The procedure is repeated until, either the property considered is verified, or all candidate components accept the counter example obtained. In the first case, a subsystem E is found which satisfies the property considered, allowing us to conclude from Proposition 3.1 that the overall system G satisfies this property. Otherwise, the counter example considered is accepted by the overall system G, which therefore cannot satisfy the property considered. This counter example, together with the information about which components constitute E, may be used for diagnosis, in order to understand why the property is not satisfied.

Please note that E such that $L(E) = \Sigma^*$, i.e. the automaton accepting all possible strings in Σ^*, can be considered initially: during the first verification iteration, E will be verified not to satisfy the property of interest, thereby producing a counter example s violating the property considered; subsequently components not accepting s can chosen to augment E.

In order to illustrate the above, we show that our manufacturing examples satisfies the property $R = \mathit{Buffer}$, i.e. that the buffer of the controlled workcell never overflows or underflows. We begin with $R = \mathit{Buffer}$ and E such that $L(E) = \Sigma^*$, and accordingly try to show that $\Sigma^* \subseteq L(\mathit{Buffer})$. As expected, we obtain a counter example $s = f_1 f_1$ not accepted by *Buffer*. Consequently, we search for a component G_i not accepting s. Since both *Machine1* and *Supervisor1* do not accept s, we arbitrarily select *Machine1*, let $E := E \times \mathit{Machine1} = \mathit{Machine1}$, and try to show $L(\mathit{Machine1}) \subseteq L(\mathit{Buffer})$. The counter example $s' = s_1 f_1 s_1 f_1$ is obtained. Again, we search for a component G_i not accepting s'. This time, the only candidate turns out to be *Supervisor1*. We let $E := E \times \mathit{Supervisor1} = \mathit{Machine1} \times \mathit{Supervisor1}$ and establish that $L(E) \subseteq L(\mathit{Buffer})$. By proposition 3.1 we conclude that $L(G) \subseteq L(\mathit{Buffer})$, i.e. the overall system satisfies the requirement considered.

Similarly, counter examples to R is controllable with respect to E can be used to determine which automata G_i not considered in E should be considered. Again, E such that $L(E) = \Sigma^*$ can be considered initially. If R is not controllable with respect to E, E is augmented with a G_i which does not accept the counter example obtained. This step is repeated until either R is shown to be controllable with respect to E, or until $E = G$.

As seen in the above example, it is possible that several components G_i do not accept a given counter example. For computational reasons, it may not be interesting to augment E with all such G_i's, but to select one, according to some heuristic aimed at focusing the search for "relevant" components.

The synthesis result R' for E and R can also be used to determine which automata G_i not considered in E should be considered. In the above example, let $E = \mathit{Machine1} \times \mathit{Machine2}$ and $R = \mathit{Buffer}$. E is not controllable with respect to R. The synthesis result $R' = \mathit{Supervisor3}$ for E and R is shown in figure 2. The string $s = s_1 f_1 s_1$ possible in $L(E)$ is seen not belong to $L(R')$ which indicates that the event s_1 should be disabled after the occurrence of the string $s_1 f_1$. Consequently, an additional component which does not accept s must be found to augment E. As above, *Supervisor1* is seen not to accept $s = s_1 f_1 s_1$, since it disables s_1 after the occurrence of $s_1 f_1$. By letting $E = \mathit{Supervisor1}$, it can be shown that R is controllable with respect to E, so that R is controllable with respect to G.

The steps of property satisfaction and controllability checking, and synthesis can be combined to form an incremental approach to controller synthesis. After each step, either the behavioural requirements are verified or counter examples are obtained, which provide the basis for the augmentation of the subsystem E of G. If a given subsystem E of G does not satisfy a given property R, E can be extended by comprising more subsystems, or R can be checked to be controllable with respect to E, so that E composed with R satisfies R. In case R is not controllable with respect to E, E can again be extended by comprising more subsystems, or an additional component R' can be synthesized, which composed with E satisfies R.

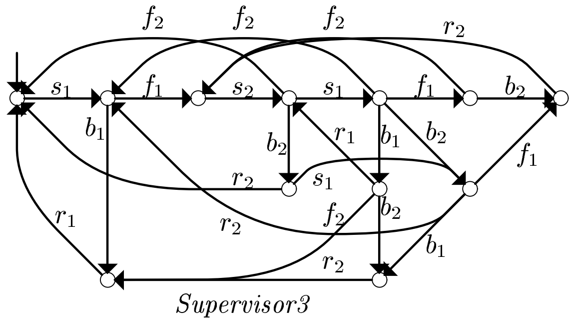

Figure 2: Synthesis result for small factory.

Problem	Size	LCE	MCE	MB
Train testbed	10^{27}	—	84259	152716
Car-locking	10^{32}	13524	6052	6981
Workcell	10^{64}	975	81646	—

Figure 3: Experimental results.

5 Experimental Results

The incremental verification approach has been applied to a number of complex examples. Significant improvements in computation times were achieved in comparison to other non-incremental verification methods.

Figure 3 shows the results obtained for three examples: a train coordination testbed, a central car-locking system, and a complex manufacturing workcell. Non-incremental verification failed in all three examples. Incremental verification was successful, although different component-selection heuristics yielded very different results and did not always succeed. The *Size* column shows the theoretical state-space size for each example. The *LCE*, *MCE*, and *MB* columns show for the three different heuristics considered, the number of states of $E \times R$ constructed[4] to prove property satisfaction or controllability, using state-enumeration based algorithms. Note that the number of constructed states constitutes a good indicator, with respect to both time and memory, of the computation effort required.

[4] over all iterations

Furthermore, in the context of the central car-locking system example, a non-trivial safety requirement was verified non-incrementally (using BDD-based algorithms) using all 53 automata in approximately one hour. The same requirement was verified incrementally, in approximately fifteen seconds, by identifying in four iterations a suitable subsystem E composed of six automata.

The results obtained show that incremental verification typically performs considerably better than non-incremental verification, but also suggest that it may be useful to test various heuristics in case a verification attempt fails. Although the *MCE* heuristic was the only consistently successful heuristic in proving property satisfaction or controllability, it was not always seen to be the most efficient.

Independently of the issue of incremental vs. non-incremental verification, experience shows that state-enumeration based algorithms are often more efficient than symbolic algorithms as long as the state space considered remains small. This is typically the case for the incremental verification of systems composed of a number of smaller subsystems. Accordingly, state-enumeration based algorithms were used in the above examples. Symbolic approaches (e.g. BDD-based algorithms) yielded qualitatively similar results.

6 Discussion and Conclusions

The theoretical results presented herein are shown to provide a powerful setting for incremental verification of safety properties. System behaviours can be incrementally augmented to verify whether given behavioural requirements are satisfied by the overall system, thus allowing for efficient computation. Behavioural requirements are verified with respect to subsystems of the overall system, nevertheless allowing assertions to be made about the overall system satisfying these requirements.

Furthermore, the theoretical results presented provide the setting for the synthesis of minimally restrictive system behaviours which, coupled with initially given system behaviours, guarantee that the behavioural requirements are not violated.

An incremental procedure is presented which considers satisfaction of properties, controllability and synthesis as successive verification steps. After each step, either the behavioural requirements are verified or counter examples are obtained, which provide the basis, together with heuristics, for the augmentation of given subsystems for verification purposes. If a given subsystem does not satisfy a property of interest, the subsystem can be extended, or the property can be checked to be controllable, so that the subsystem, composed with the property, satisfy the property. In case the property is not controllable, the subsystem can again be extended or an additional component can be synthesized, which composed with the subsystem satisfies the property.

In a number of complex examples, significant improvements in computation times have been achieved, using the incremental approach proposed.

References

[1] A. Aziz, F. Balarin, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential synthesis using S1S", Proc. Int. Conf. on CAD, 612–617, 1995.

[2] A. Aziz, V. Singhal, G. M. Swamy, R. K. Brayton, "Minimizing Interacting Finite State Machines, A Compositional Approach to Language Containment", Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors (ICCD '94), 255–261, 1994.

[3] T. Bultan, J. Fischer, R. Gerber, "Compositional verification by model checking for counter-examples", ACM SIGSOFT Software Engineering Notes, **21** (3), 224–238, 1996.

[4] B. A. Brandin, "Combined Language and State Based Supervisory Control Synthesis", Report ZFE T SE 1-1996-BB-1, Siemens Corporate Research, 1996.

[5] B. A. Brandin, "The real-time supervisory control of an experimental manufacturing cell", IEEE Trans. Robotics and Automation, **12** (1), 1–14, 1996.

[6] E. M. Clarke, E. A. Emerson, A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications", ACM Trans. Programming Languages and Systems, **8** (2), 244–263, 1986.

[7] S. Graf, B. Steffen, "Compositional minimization of finite state systems", Proc. 1990 Workshop on Computer-Aided Verification, 186–196, 1990.

[8] N. Halbwachs, "Synchronous observers and the verification of reactive systems", Proc. 3rd Int. Conf. on Algebraic Methodology and Software Technology (AMAST '93), Springer, Twente, 1993.

[9] K. L. McMillan, "Symbolic Model Checking", Kluwer Academic Publishers, Dordrecht–Boston–London, 1993.

[10] P. J. Ramadge, W. M. Wonham, "Supervisory control of a class of discrete-event systems", SIAM J. Control and Optimization, **25** (1), 206–230, 1987.

[11] P. J. Ramadge, W. M. Wonham, "The control of discrete-event systems", IEEE Proceedings, **77** (1), 81–98, 1989.

[12] Y. Watanabe, R. K. Brayton, "The maximum set of permissible behaviours for FSM networks", Proc. Int. Conf. on CAD, Santa Clara, 316–320, 1993.

[13] W. M. Wonham, P. J. Ramadge, "Modular supervisory control of discrete event systems", Maths. of Control, Signals and Systems, **1** (1), 13–30, 1988.