Model-based Verification of Adaptive Embedded Systems under Environment Constraints [*]

Ina Schaefer
Dept. of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
schaefer@chalmers.se

Arnd Poetzsch-Heffter
Software Technology Group
TU Kaiserslautern
Kaiserslautern, Germany
poetzsch@cs.uni-kl.de

ABSTRACT

Model-based verification of adaptive embedded systems is a promising approach to deal with the increased complexity that adaptation imposes on system design. Properties of embedded systems typically depend on the environment in which they are deployed. Thus, the environment has to be considered for verification. In this paper, we propose a technique to verify properties of design-level models of adaptive embedded systems under environment constraints. We transfer ideas originating from assume-guarantee reasoning for Kripke structures to design-level models of adaptive embedded systems in order to reduce conditional validity checking to standard model checking.

Keywords

Adaptive Embedded Systems; Model-based Verification; Temporal Logic; Assume-Guarantee Reasoning

1. INTRODUCTION

Adaptation is increasingly used in embedded systems to meet the high demands on safety and availability in order to react to changing environment conditions or system failures. However, adaptation significantly complicates system development as adaptation in one system part may trigger a sequence of reconfigurations throughout the system. Model-based verification of adaptive embedded systems [9] is one approach to deal with the increased design complexity allowing to detect conceptual errors already on the design-level. Properties to be verified of design-level models of adaptive embedded systems usually depend on the environment in which a system is deployed. A property is not valid for the adaptive system in general, but only under assumptions on its environment. Hence, for model-based verification of adaptive embedded systems, environment constraints have to be taken into account. Properties of adaptive embedded systems and environment constraints can be expressed in temporal logics facilitating automated verification by model-checking. Only for linear temporal logic formulas, verification of a system property under an environment constraint can be reduced to checking the implication of the constraint and the property [6]. However, not all critical properties of adaptive systems are expressible in linear time temporal logic, such as the property $\mathsf{AF}\,\mathsf{AG}\,\varphi$ expressing that on all computation paths finally a state is reached from which all computation paths invariantly satisfy φ.

In this paper, we propose a general approach to verify adaptive embedded systems under environment constraints that allows dealing with a larger fragment of temporal logic system properties and environment constraints. We transfer ideas originating from assume-guarantee reasoning over Kripke structures [6, 4] to design-level models of adaptive embedded systems. An environment constraint is captured by a maximal environment that can exhibit any possible behavior consistent with the constraint. The maximal environment is composed with the model of the adaptive system. A property of the composition can be verified by standard model checking techniques which yields that the property is valid under the environment constraint.

This paper is organized as follows: In Section 2, we review related work. In Section 3, we explain how we formalize adaptive systems, their properties and environment constraints. In Section 4, we present our approach to verify system properties under environment constraints based on maximal environments. Section 5 concludes the paper.

2. RELATED WORK

Formal verification of adaptive systems has become an active area of research [5, 11, 1, 12] in order to guarantee critical properties of the system behaviour. However, verification of properties under environment constraints has so far not been considered in the context of adaptive embedded systems. In [8], Verilog HDL environments satisfying constraints are generated in order to verify network protocols. [7] proposes the automatic construction of specific environments for program fragments in Java. [3] verifies aspect-oriented programs by automatically generating base-programs complying to constraints. In [10], properties for Java card applets are established using constraints on applets that are loaded after deployment.

[*]This work has been supported by the Rheinland-Pfalz Research Center for Mathematical and Computational Modelling $(CM)^2$, the Deutsche Forschungsgemeinschaft (DFG) and the European project HATS, funded in the 7th Framework Program.

3. ADAPTIVE SYSTEMS

In order to analyze the behavior of a system already at design-time, we consider adaptation by *predetermined reconfiguration.* Formally, we describe the considered systems as synchronous adaptive systems (SAS) which allow modular modelling and verification of adaptation behavior. Figure 1 sketches their syntactic representation. For a full formal account, refer to [9, 1].

A System over a set of variables Var and a set of values Val consist of a set of modules M that operate in a clocked-synchronous way. Each module $\mathsf{Module_i}$ has a set of predetermined configurations corresponding to the different behavioral variants of the module. In order to specify adaptation behavior modularly within a module separated from the functionality, each module has two disjoint sets of input, local and output variables: the functional variables var with their initial values init and the adaptive variables $\mathsf{ad\_var}$ with their initial values $\mathsf{ad\_init}$ encapsulated in the adaptation aspect adapt. The condition under which a configuration j is activated is formulated in the configuration guard $\mathsf{guard_j}$ only depending on the adaptive variables. The state transition function $\mathsf{next\_state_j}$ computes the functional local variables and the output transition function $\mathsf{next\_out_j}$ the functional output variables. Adaptive output variables are connected to adaptive input variables by the adaptive system connections $\mathsf{conn_a}$. The adaptive connections convey information about the used configurations such that connected modules can react to adaptations of other modules. Functional output variables are connected to functional input variables by the functional system connections $\mathsf{conn_d}$. Functional connections communicate functional values computed by the modules. The system input and system output variables are given by the module variables that are unconnected.

The semantics of a system is defined by a state-transition system $\mathsf{T} = (\Sigma, \mathsf{I}, \leadsto)$. The set of states Σ is the set of possible valuations of the module variables. In the initial state $\sigma_0 \in \mathsf{I}$, all variables have their initial values. A clocked-synchronous transition $\leadsto$ evolves in two steps. First, all modules read their inputs. Second, all modules synchronously perform a local transition. A local transition of a module proceeds as follows: First, the new valuation of the adaptive variables is computed by the $\mathsf{ad\_next\_state}$ and the $\mathsf{ad\_next\_out}$ functions. Then, the configuration with valid guard is activated, and the respective state and output transition functions are executed. An execution path of a system is represented by a sequence of states.

$\mathsf{Module} = (\mathsf{var}, \mathsf{init}, \mathsf{configurations}, \mathsf{adapt})$
with $\mathsf{var} \subseteq \mathsf{Var}$ and $\mathsf{init} : \mathsf{var} \to \mathsf{Val}$

$\mathsf{configurations} = \{(\mathsf{guard_j}, \mathsf{next\_state_j}, \mathsf{next\_out_j})\}$
$\mathsf{guard_j}$: a Boolean constraint over $\mathsf{ad\_var}$
$\mathsf{next\_state_j}, \mathsf{next\_out_j} : (\mathsf{var} \to \mathsf{Val}) \to (\mathsf{var} \to \mathsf{Val})$

$\mathsf{adapt} = (\mathsf{ad\_var}, \mathsf{ad\_init}, \mathsf{ad\_next\_state}, \mathsf{ad\_next\_out})$
$\mathsf{ad\_var} \subseteq \mathsf{Var}$ and $\mathsf{ad\_init} : \mathsf{ad\_var} \to \mathsf{Val}$
$\mathsf{ad\_next\_state} : (\mathsf{ad\_var} \to \mathsf{Val}) \to (\mathsf{ad\_var} \to \mathsf{Val})$
$\mathsf{ad\_next\_out} : (\mathsf{ad\_var} \to \mathsf{Val}) \to (\mathsf{ad\_var} \to \mathsf{Val})$

$\mathsf{System} = (\mathsf{M}, \mathsf{conn_a}, \mathsf{conn_d})$
where $\mathsf{M} = \{\mathsf{Module_1}, \ldots, \mathsf{Module_n}\}$

Figure 1: SAS System Description

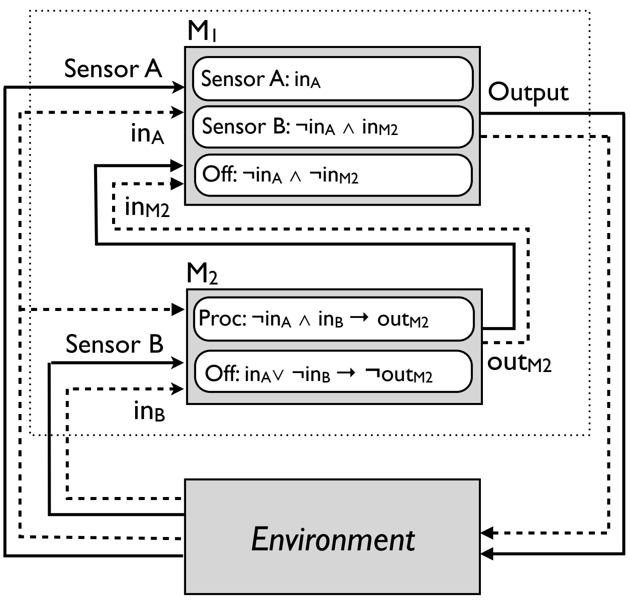

Figure 2: Example System and Environment

As an example, consider the adaptive control system depicted in Figure 2. The system has two different sensors: Sensor A requiring light for correct operation and sensor B operating without light. The system adapts its mode of operation depending on the brightness of the environment. If it is light enough, the output is directly computed by configuration $\mathsf{Sensor\,A}$ of module $\mathsf{M_1}$. If it is too dark, sensor B is used whose values have to be preprocessed by configuration Proc of module $\mathsf{M_2}$. The availability of the sensors is modelled by boolean adaptive variables $\mathsf{in_A}$ and $\mathsf{in_B}$. In Figure 2, both modules are depicted with their configurations and the associated guards. Solid lines represent functional connections, dashed lines adaptive connections. The adaptation behaviour is as follows:

- If sensor A becomes unavailable, module $\mathsf{M_1}$ enters its shutdown configuration Off, and module $\mathsf{M_2}$ adapts from configuration Off to configuration Proc.
- In the next cycle, preprocessed values are available, which is signaled by the adaptive variable $\mathsf{out_{M_2}}/\mathsf{in_{M_2}}$, such that $\mathsf{M_1}$ adapts to configuration $\mathsf{Sensor\,B}$.
- If sensor A becomes available again, $\mathsf{M_1}$ returns to configuration $\mathsf{Sensor\,A}$ and $\mathsf{M_2}$ to configuration Off.

A critical property of the system is that the output is available, i.e., module $\mathsf{M_1}$ is only in configuration Off for one cycle. This can be ensured under the constraint that the sensors A and B are not unavailable at the same time.

In order to formalize properties and environment constraints, we use a variant of the temporal logic ACTL* [2], called

$\mathsf{A}\mathcal{L}_{SAS}$ [9], that allows expressing universal temporal properties of synchronous adaptive systems. The logic $\mathsf{A}\mathcal{L}_{SAS}$ is strictly more expressive than the linear temporal logic fragment that consists of formulas $\mathsf{A}\varphi$ where φ does not contain any path quantifiers.

- The atomic propositions of $\mathsf{A}\mathcal{L}_{SAS}$ are constraints on variables and values. The atomic propositions occurring in a formula φ are denoted by $\mathsf{Atoms}(\varphi)$. The variables contained in a formula φ are denoted by $\mathsf{Var}(\varphi)$. The configuration a module M_1 uses in a cycle is captured by a special variable $\mathsf{use\_conf}_{\mathsf{M}_1}$.
- Besides Boolean connectives, we have temporal operators, e.g., $\mathsf{G}\varphi$ ("globally") denotes that φ holds on all states of a path, $\mathsf{F}\varphi$ ("finally") that φ holds eventually in a state on a path, and $\mathsf{X}\varphi$ ("next") that φ holds in the next state. Additionally, we have the path quantifier $\mathsf{A}\varphi$ ("φ holds for all paths."). Negation is only applied to atomic propositions.
- $\mathsf{T} \models \varphi$ denotes that the property φ holds in all initial states of T, and $\sigma \models \varphi$ denotes that φ holds in the state σ.

Using $\mathsf{A}\mathcal{L}_{SAS}$, the property that the module M_1 is only in configuration Off for one cycle is formulated as follows. For all paths (A) starting from the initial state, in all states (G), if M_1 uses the configuration Off, on all paths (A) in the successor state (X), M_1 does not use configuration Off.[1]

$$\psi = \mathsf{AG}\,(\mathsf{use\_conf}_{\mathsf{M}_1} = \mathsf{Off} \rightarrow \mathsf{AX}\,\mathsf{use\_conf}_{\mathsf{M}_1} \neq \mathsf{Off})$$

The environment constraint that the sensors A and B are not simultaneously unavailable is stated by the following property that has to hold on all paths (A) in all states (G).

$$\varphi = \mathsf{AG}\,((\mathsf{in_A} \wedge \neg\mathsf{in_B}) \vee (\neg\mathsf{in_A} \wedge \mathsf{in_B}))$$

4. MAXIMAL ENVIRONMENTS

Only for linear time temporal logic properties, checking the validity of a property ψ under an environment constraint φ can be reduced to checking the implication $\varphi \rightarrow \psi$ [6]. In order to verify all properties of adaptive systems expressible in $\mathsf{A}\mathcal{L}_{SAS}$ under environment constraints, we transfer the maximal model technique [4] developed for assume-guarantee reasoning on Kripke structures [2] to design-level models of adaptive systems. First, we formally define the environment of a system. Second, we formalize conditional validity, i.e., that a system satisfies a property under an environment constraint. Then, we introduce maximal environments as most general environments satisfying a constraint and show that conditional validity checking can be reduced to standard model checking by composing a system with the maximal environment and verifying the property for the composition.

The environment of a system provides input to the module input variables that are not connected by the system connections and receives output from the unconnected module output variables. We define an environment as a module that closes a system such that there are no unconnected variables in the composition of the environment with the system. In the following, we denote the input variables of a module $\mathsf{M_i}$ by $\mathsf{in_i} \subseteq \mathsf{var_i}$ and the output variables by $\mathsf{out_i} \subseteq \mathsf{var_i}$. The adaptive input variables are denoted by $\mathsf{ad\_in_i} \subseteq \mathsf{ad\_var_i}$ and the adaptive output variables by $\mathsf{ad\_out_i} \subseteq \mathsf{ad\_var_i}$. The functional output variables of a system S are defined as $\mathsf{out_S} = \{v \in \bigcup_i \mathsf{out_i} \mid v \notin dom(\mathsf{conn_d})\}$. The functional system input variables are defined as $\mathsf{in_S} = \{v \in \bigcup_i \mathsf{in_i} \mid v \notin img(\mathsf{conn_d})\}$. The adaptive system input variables $\mathsf{ad\_in_S}$ and the adaptive system output variables $\mathsf{ad\_out_S}$ are defined accordingly.

Definition 1 (Environment). *A module* $\mathsf{M_E}$ *is an environment for a system* S, *if* $\mathsf{in_E} = \{\mathsf{v_e} \mid \mathsf{v} \in \mathsf{out_S}\}$, $\mathsf{out_E} = \{\mathsf{v_e} \mid \mathsf{v} \in \mathsf{in_S}\}$, $\mathsf{ad\_in_E} = \{\mathsf{v_e} \mid \mathsf{v} \in \mathsf{ad\_out_S}\}$ *and* $\mathsf{ad\_out_E} = \{\mathsf{v_e} \mid \mathsf{v} \in \mathsf{ad\_in_S}\}$. *The system* S *is composed with the envionment* $\mathsf{M_E}$ *by*

$$\mathsf{S} \parallel \mathsf{M_E} = (\mathsf{M} \cup \{\mathsf{M_E}\}, \mathsf{conn_{aE}}, \mathsf{conn_{dE}})$$

where $\mathsf{conn}_{\mathsf{a}E} = \mathsf{conn_a} \cup \{(\mathsf{v_e},\mathsf{v}) \mid \mathsf{v} \in \mathsf{ad\_in_S}\} \cup \{(\mathsf{v},\mathsf{v_e}) \mid \mathsf{v} \in \mathsf{ad\_out_S}\}$

and $\mathsf{conn}_{\mathsf{d}E} = \mathsf{conn_d} \cup \{(\mathsf{v_e},\mathsf{v}) \mid \mathsf{v} \in \mathsf{in_S}\} \cup \{(\mathsf{v},\mathsf{v_e}) \mid \mathsf{v} \in \mathsf{out_S}\}$

A system S satisfies a property ψ under an environment constraint φ if for all environments $\mathsf{M_E}$ satisfying φ, S composed with $\mathsf{M_E}$ satisfies ψ. The environment constraint φ only refers to the system variables $\mathsf{ad\_in_S} \cup \mathsf{in_S} \cup \mathsf{ad\_out_S} \cup \mathsf{out_S}$, whereas the property ψ refers to the internal system behavior expressed by all module variables. For a correct formal account, we have to transform the constraint φ over the input and output variables of S to a formula φ' over the variables of $\mathsf{M_E}$.

Definition 2 (Conditional Validity). *Let* S *be a system,* $\psi \in \mathsf{A}\mathcal{L}_{SAS}$ *with* $\mathsf{Var}(\psi) \subseteq \bigcup_i \mathsf{var_i}$ *a system property and* $\varphi \in \mathsf{A}\mathcal{L}_{SAS}$ *with* $\mathsf{Var}(\varphi) \subseteq \mathsf{in_S} \cup \mathsf{ad\_in_S} \cup \mathsf{out_S} \cup \mathsf{ad\_out_S}$ *an environment constraint.* ψ *is* conditionally valid *for* S *under the environment constraint* φ, *denoted by* $\{\varphi\}\mathsf{S}\{\psi\}$, *iff for all environments* $\mathsf{M_E}$ *with* $\mathsf{T}_{\mathsf{M_E}} \models \varphi'$, *it holds that* $\mathsf{T}_{\mathsf{S}\parallel\mathsf{M_E}} \models \psi$ *where* $\varphi' = \varphi[\mathsf{v}/\mathsf{v_e}]$ *is derived from* φ *by replacing every occurrence of* $\mathsf{v} \in \mathsf{in_S} \cup \mathsf{ad\_in_S} \cup \mathsf{out_S} \cup \mathsf{ad\_out_S}$ *by* $\mathsf{v_e} \in \mathsf{var_E} \cup \mathsf{ad\_var_E}$.

A maximal environment for a constraint φ is an environment that is the most general environment satisfying φ. This means that it has all possible behaviors consistent with the constraint φ. In order to formally define maximal environments, we introduce consistent simulations as a preorder on environments capturing when one environment is more general than another, i.e., when it has more behaviors. An environment $\mathsf{M_E}$ is consistently simulated by an environment $\mathsf{M'_E}$ with respect to a property φ if all behaviors of $\mathsf{M_E}$ have a corresponding behavior in $\mathsf{M'_E}$ that cannot be distinguished by the atomic propositions of φ. The environment $\mathsf{M'_E}$ can have behaviors to which no behavior of $\mathsf{M_E}$ corresponds, such that $\mathsf{M'_E}$ is more general than $\mathsf{M_E}$.

[1]Due to the simplicity of the example, ψ has an equivalent formulation in the linear fragment of $\mathsf{A}\mathcal{L}_{SAS}$, but in practice, there are critical system properties that are not expressible in the linear fragment, e.g., $\mathsf{AF}\,\mathsf{AG}\,\varphi$.

Definition 3 (Consistent Simulation). *An environment* $\mathsf{M_E}$ *is consistently simulated by an environment* $\mathsf{M'_E}$ *with respect to a property* $\varphi \in \mathsf{A}\mathcal{L}_{SAS}$, $\mathsf{M_E} \preceq_{[\varphi]} \mathsf{M'_E}$, *iff there exists a relation* $\mathsf{R} \subseteq \Sigma \times \Sigma'$ *on the states of the transition systems* $\mathsf{T_{M_E}}$ *and* $\mathsf{T_{M'_E}}$ *such that*

1. *for each* $\sigma_0 \in \mathsf{I}$, *there exists* $\sigma'_0 \in \mathsf{I}'$ *such that* $\mathsf{R}(\sigma_0, \sigma'_0)$.
2. *for all* $i \geq 0$, *if* $\sigma_i, \sigma_{i+1} \in \Sigma$ *and* $\sigma'_i \in \Sigma'$ *with* $\mathsf{R}(\sigma_i, \sigma'_i)$ *and* $\sigma_i \rightsquigarrow \sigma_{i+1}$, *there exists* $\sigma'_{i+1} \in \Sigma'$ *with* $\sigma'_i \rightsquigarrow' \sigma'_{i+1}$ *such that* $\mathsf{R}(\sigma_{i+1}, \sigma'_{i+1})$.
3. *for all* $a \in \mathsf{Atoms}(\varphi)$, *if* $\mathsf{R}(\sigma, \sigma')$ *and* $\sigma' \models a$, *it holds that* $\sigma \models a$.

A maximal environment $\mathsf{M_E}(\varphi)$ for a constraint φ is defined as an environment that satisfies φ itself and consistently simulates all other environments satisfying φ.

Definition 4 (Maximal Environment). *The environment* $\mathsf{M_E}(\varphi)$ *of a system* S *is a maximal environment for the property* $\varphi \in \mathsf{A}\mathcal{L}_{SAS}$, *iff* $\mathsf{T}_{\mathsf{M_E}(\varphi)} \models \varphi$, *and for all environments* $\mathsf{M'_E}$ *with* $\mathsf{T_{M'_E}} \models \varphi$, *it holds that* $\mathsf{M'_E} \preceq_{[\varphi]} \mathsf{M_E}(\varphi)$.

The existence of maximal environments can be established by tableau-based approaches [2] and automata-theoretic results [6]. Verification of a system property under an environment constraint can be reduced to a standard model checking problem using maximal environments.

Theorem 1 (Validity by Max. Environments). *Let* S *be a system,* $\psi \in \mathsf{A}\mathcal{L}_{SAS}$ *a property of* S *with* $\mathsf{Var}(\psi) \subseteq \bigcup_i \mathsf{var_i}$ *and* $\varphi \in \mathsf{A}\mathcal{L}_{SAS}$ *an environment constraint with* $\mathsf{Var}(\varphi) \subseteq \mathsf{ins_S} \cup \mathsf{ad\_ins_S} \cup \mathsf{outs_S} \cup \mathsf{ad\_outs_S}$. *It holds that*

$$\{\varphi\}SAS\{\psi\} \quad \textit{iff} \quad \mathsf{T}_{\mathsf{S}\|\mathsf{M_E}(\varphi)} \models \psi$$

For the linear fragment of $\mathsf{A}\mathcal{L}_{SAS}$ and for the branching time fragment of $\mathsf{A}\mathcal{L}_{SAS}$ (using only combinations of a path quantifier and a temporal operator), maximal environments can be constructed automatically by tableau-based approaches [9, 6, 4, 2].

The separation of functionality and adaptation behavior in synchronous adaptive systems can be exploited to make conditional validity checking more efficient. If a property and the environment constraint only refer to the adaptation behavior as in the example, it is sufficient to consider the adaptation behavior of the system and to construct a maximal environment only for the adaptive input and output variables.

5. CONCLUSION

We have presented an general approach to verify temporal properties of adaptive systems under environment constraints. The technique is integrated in the AMOR framework [9]. This framework provides verification complexity reductions, i.e., slicing, data abstraction and decomposition techniques, making large adaptive system models amenable to model checking. Because the composition of a system with a maximal environment is a standard SAS, the AMOR reduction techniques can be immediately applied for simplifying verification. For future work, we aim at generating environment constraints that are necessary for the validity of system properties in order to reason in which environments a system can safely be deployed.

6. REFERENCES

[1] R. Adler, I. Schaefer, T. Schuele, and E. Vecchie. From Model-Based Design to Formal Verification of Adaptive Embedded Systems. In *ICFEM*, 2007.

[2] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT, 1999.

[3] M. Goldman and S. Katz. MAVEN: Modular Aspect Verification. In *TACAS*, 2007.

[4] O. Grumberg and D. Long. Model Checking and Modular Verification. *ACM TOPLAS*, 16(3):843–871, 1994.

[5] S. S. Kulkarni and K. Biyani. Correctness of Component-Based Adaptation. In *CBSE*, 2004.

[6] O. Kupferman and M. Vardi. An automata-theoretic approach to modular model checking. *ACM TOPLAS*, 22(1), 2000.

[7] C. Pasareanu, M. Dwyer, and M. Huth. Assume-Guarantee Model Checking of Software: A Comparative Case Study. In *SPIN Workshop*, 1999.

[8] H. Peng, Y. Mokhtari, and S. Tahar. Environment Synthesis for Compositional Model Checking. In *ICCD*, 2002.

[9] I. Schaefer. *Integrating Formal Verification into the Model-based Development for Adaptive Embedded Systems.* PhD thesis, University of Kaiserslautern, 2008.

[10] C. Sprenger, D. Gurov, and M. Huisman. Compositional Verification for Secure Loading of Smart Card Applets. In *MEMOCODE*, 2004.

[11] E. Strunk. *Reconfiguration Assurance in Embedded System Software.* PhD thesis, University of Virginia, Charlottesville, USA, 2005.

[12] J. Zhang, H. Goldsby, and B. Cheng. Modular verification of dynamically adaptive systems. In *AOSD*, 2009.