Component-based CPS Verification: A Recipe for Reusability

Andreas Müller
Johannes Kepler University Linz, Altenbergerstr. 69, 4040 Linz, Austria
andreas.mueller@jku.at

1 Overview

Cyber-physical systems (CPS) are today pervasively embedded into our lives and increasingly act in close proximity as well as with direct impact to humans. Because of their safety-criticality, we have to ensure *correctness properties*, such as *safety* and *liveness*. Thus, formal verification techniques to analyze CPS are of paramount importance to guarantee these properties.

Formal verification methods rest on *models* capturing the *discrete and continuous dynamics* of a CPS (i. e., hybrid system models), which abstract from implementation details to facilitate verification. Since formal verification of hybrid systems is known to be undecidable for realistic models, current methods make a trade-off between full automation (*model checking* and *reachability analysis* restricted to certain classes of hybrid systems, e. g., [5]) and model expressiveness (*deductive verification* of complex models mixing automated and human guided proofs, e. g., [7]). To make human guidance feasible despite complex continuous dynamics, CPS practically mandate for techniques to reduce system complexity.

We study decomposing a model into smaller components, which can be proven separately before re-composing them to a fully verified model. With current deductive verification techniques for CPS, however, the price for such component-based development is full re-verification on every composition step, which is a nuisance if human guidance is required each time.

Vision: reduce modeling and verification effort and complexity, and increase reusability by component-based CPS development.

Although component-based software engineering in general has seen extensive research, only few approaches explicitly deal with CPS, like Damm et al. [3], who propose a design methodology for hybrid systems based on sequential composition of components using alarms.

A field closely related to component-based verification is assume-guarantee reasoning (AGR), which was originally intended as a device to counteract the state explosion problem in model checking by decomposing a verification task into subtasks. In AGR, individual components are analyzed together with *assumptions* about their context and *guarantees* about their behavior (i. e., a component's "contract"). Benvenuti et al. [1] propose an approach to check these contracts for hybrid automata with non-linear dynamics, using the reachability toolbox Ariadne. AGR is often used along with abstraction/refinement ap-

proaches (e. g., [2]) and the rules are often circular in the sense that one component is verified in the context of the other and vice-versa (e. g., [5]).

Summarizing, current component-based approaches are often limited to linear dynamics (e. g., [2,3]), need to abstract away continuity (e. g., [5]) or rely on reachability analysis, over-approximation and model checking (all of the above). In the next section we will propose a *component-based* modeling and verification approach based on *deductive verification*.

2 Research Approach

We will follow a three-step research approach, where we (i) conduct initial case studies, to gain insight into decomposition options in deductive verification, (ii) develop component-based modeling and verification for hybrid systems and (iii) evaluate our findings with case studies.

For this work, we use *differential dynamic logic* ($\mathsf{d}\mathcal{L}$), which is a first-order dynamic logic that has a notation for hybrid systems as *hybrid programs* and its hybrid deductive verification tool KeYmaera [7] that allows proving correctness properties of these hybrid programs. Hybrid programs allow sequential composition, non-deterministic choice, repetition and assignment, deterministic assignments, state checks and continuous evolution. Here a hybrid interpretation of time is used, where time evolves continuously and without discretization during continuous evolution and in discrete steps otherwise. The hybrid programs can be embedded into $\mathsf{d}\mathcal{L}$ formulas using the modalities $[a]$ and $\langle a \rangle$ to reason about all runs of a hybrid program a or at least one run of a respectively.

The ultimate goal is a framework that provides a set of composition operations that transfer verified properties of the internal and external behavior of components to composites. Users then have to decompose a system into components, verify their internal and external behavior in isolation according to our framework and use the composition operations to recreate the overall system. The properties of the composed system can be derived from its components.

2.1 Initial Compositional Modeling and Verification Case Studies

Based on prior experience with road traffic[1] we will introduce compositional modeling and verification of road networks. Road network capacity analysis involves highly repetitive parts, such as traffic lights or merging roads. On that account, we coarsely approximate traffic flow using linear water tank models. These limitations allow studying component interfaces, continuous dynamics and composition in a restricted setting of a single composition operation (i. e., instantaneous, loss-less passing of flow) and simplified continuous dynamics (i. e., approximate flows with linear ordinary differential equations (ODEs)).

We will complement the macroscopic network flow study, which is highly time-dependent, as flow accumulates over time, with a microscopic case study

[1] Gained in the research project CSI (`http://csi.situation-awareness.net`).

on autonomous cars, which mainly have to deal with their ever changing surroundings and where safety criticality is even more of an issue. Thus, we have to extend our previous approach with multiple composition operations (e. g., noisy measurements) and non-linear continuous dynamics (e. g., curved trajectories).
Status: Traffic Components with Maximum Flow. We modeled three types of flow components (traffic light, two-way-merge, two-way-split) and verified that they will not exceed their capacity for some time T_{local}, when considering the maximum possible in- and outflow of a component. Furthermore, we introduced a composition operation which results in a safe composite, if both components follow their local safety property and a simple arithmetic composition relation holds. This condition has to be checked using designated values when composing components to form the whole system. While deciding the validity of a safety property for the entire traffic network would be doubly exponential in the (assumable large) number of variables [4], the evaluation of the arithmetic condition over the reals for concrete numbers is linear in the formula size [6]. Thus, the arithmetic composition relation can be checked at scale in a modeling tool when building road networks. When it holds, the local safety property transfers to the whole system, ensuring no overflow until a time T_{global}.

2.2 Component-based CPS Development

Component-based CPS development and verification requires definition and verification of components (i. e., their *internal behavior*) and their interfaces (i. e., their *external behavior*), as well as their *verified composition*. The concepts in this section are all work in progress and the descriptions and examples illustrate ideas rather than final contributions.
Internal Behavior. While $\mathsf{d}\mathcal{L}$ is well-suited to describe hybrid systems, the intent behind variables cannot be specified (e. g., can a controller set a cars acceleration?). We envision a logic based on $\mathsf{d}\mathcal{L}$, extending it by a *type system for variables*. For implementation it is useful to distinguish between *sensors* and *actuators*, *environmental* (laws of physics) and *control* variables (set by choice). For verification, it is useful to distinguish between *readable*, *writeable*, *discrete*, and *continuous* variables. Approaches based on hybrid automata often distinguish between *input*, *control* and *output* variables.

The verification of the internal behavior of a component can make use of variable types, which are usually neglected when proving $\mathsf{d}\mathcal{L}$ formulas. For instance, the controller is only allowed to set certain variables to which it has write access (e. g., set cars acceleration, but not its position). The proof rule[2] in Fig. 1 for assigning the value of a variable y to a variable x in a component c, requires write access to x and read access to y. Similar rules could be derived for other operations (e. g., ODEs), to enforce type safety during proofs.

$$\frac{\Delta, c \downarrow x, c \uparrow y \vdash \phi_x^y, \Gamma}{\Delta, c \downarrow x, c \uparrow y \vdash [x := y]_c\ \phi, \Gamma}$$

Fig. 1. Proof Rule

[2] $c \downarrow x$ means "c has write access to x", $c \uparrow y$ means "c has read access to y"

External Behavior. The external behavior of a component is defined by its *interface*, including *contracts* on input and output ports. Similar to AGR, the interface specifies a contract (e. g., $\psi_1 \rightarrow [C_1]\phi_1$, i. e., assuming ψ_1, all runs of component C_1 guarantee ϕ_1) about what the component assumes at its input ports (assumption ψ_1) and what it guarantees at its output ports (guarantee ψ_1). The contract ψ_1 and ϕ_1 can be specified in various ways. Automata-based AGR approaches (e. g., [5]) mostly use some kind of automaton-based specification. In deductive verification, predicates over the input variables or timed regular expressions are more suitable. Considering the traffic flow example introduced initially, they provide a way of stating which flow is produced by an output for which duration of time and thus allows a detailed interface description (e. g., $(3 \cdot A; 1 \cdot B)*$ means flow A for 3 time units followed by flow B for 1 time unit repeated indefinitely). Note, that every behavior described using timed regular expressions can also be expressed using $\mathsf{d}\mathcal{L}$.

Verification. To ensure formally verified CPS components, two aspects must be considered, namely, verifying that the internal behavior actually follows its specified external behavior (e. g., $\psi_1 \rightarrow [C_1]\phi_1$) and verifying that a component's external behavior is feasible for the operational area at hand and obeys a given local safety condition (e. g., under weaker assumptions Φ_1, stronger promises Ψ_1 are guaranteed, i. e., $(\Phi_1 \rightarrow \phi_1) \wedge (\psi_1 \rightarrow \Psi_1)$, cf. "dominance" in [1]).

In order to model and ensure safety of an entire CPS, we further need a way of safely combining the aforementioned components. Since time always passes simultaneously throughout a hybrid system, components have to be composed in parallel. We envision composition operations that go beyond loss-less instantaneous value passing and propose *composition operators*, including (i) port forwarding operators (i. e., loss-less and instant connection), (ii) operators that influence the continuous evolution of components (e. g., components evolve during communication delay), (iii) operators that affect the forwarded information (e. g., uncertainty for measurements, noise on electrical signals), and (iv) operators that perform state estimation of non-accessible values (e. g., estimate the acceleration by measuring the change of speed). An example composition operation is the noisy composition: $\underset{noise}{\multimap\!\bullet} \equiv \tilde{x} := *; ?\,|\tilde{x} - x| \leq \delta$.

The ultimate goal is to ensure that the composite system obeys to a global safety condition Φ if it is started in a safe state Ψ, i. e., $\Psi \rightarrow [C_1 \multimap\!\bullet\, C_2]\,\Phi$, which can be achieved by an assume-guarantee style rule for deductive verification using $\mathsf{d}\mathcal{L}$. If it is ensured that a component (e. g., C_1) follows its external behavior specification (i. e., it obeys its contract, e. g., $\psi_1 \rightarrow [C_1]\phi_1$) and ensures its local safety condition (e. g., $\phi_1 \rightarrow \Phi_1$), it remains to ensure provably correct composition, and (if required) automatically derive composition *proof obligations* Θ to ensure overall system correctness (e. g., $(\bigwedge_i \Phi_i) \wedge \Theta \rightarrow \Phi$).

In our initial case study on traffic networks, a composition operation connects two roads allowing cars to drive from one component to another (e. g., from a traffic light to an intersection). For a sample component (e. g., a traffic light) the contract restricts the maximum outflow and the load (i. e., ratio between used and available space on the road) of a component. Here, the local safety

property and the composition property are the same for all components, stating that it should not produce a traffic breakdown for a predefined time (i. e., the number of cars should never exceed the available space) and respectively that the outflow of a component must not be larger than the allowed inflow of the subsequently connected one. If these properties hold, the overall safety property (i. e., no overflow anywhere in the network for a predefined time) can be inferred.

2.3 Evaluation

We plan to implement a software prototype, which will include a library of components and associated composition operations. To show the applicability of the approach, we already implemented a tool called SAFE-T[3]. Based on maximum-flow components it allows combining them to larger traffic networks, while checking the aforementioned arithmetic composition condition automatically. The tool can be used to find the origin of a traffic breakdown and analyze how it will propagate through the network.

Based on the implementation, we will consider existing case studies and compare our compositional models to the original, monolithic ones specifically w.r.t. *model complexity* and *proof effort*.

Acknowledgements. Work funded by BMVIT grant FFG BRIDGE 838526, by OeAD Marietta Blau grant ICM-2014-08600 and as part of P28187-N31.

References

1. Benvenuti, L., Bresolin, D., Collins, P., Ferrari, A., Geretti, L., Villa, T.: Assume–guarantee verification of nonlinear hybrid systems with Ariadne. Int. J. of Robust and Nonlinear Control 24(4), 699–724 (2014)
2. Bogomolov, S., Frehse, G., Greitschus, M., Grosu, R., Pasareanu, C., Podelski, A., Strump, T.: Assume-guarantee abstraction refinement meets hybrid systems. In: Yahav, E. (ed.) Hardware and Software: Verification and Testing, LNCS, vol. 8855, pp. 116–131. Springer (2014)
3. Damm, W., Dierks, H., Oehlerking, J., Pnueli, A.: Towards component based design of hybrid systems: Safety and stability. In: Manna, Z., Peled, D. (eds.) Time for Verification, LNCS, vol. 6200, pp. 96–143. Springer (2010)
4. Davenport, J.H., Heintz, J.: Real Quantifier Elimination is Doubly Exponential. J. Symb. Comput. 5(1-2), 29–35 (1988)
5. Frehse, G., Zhi Han, Krogh, B.: Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In: 43rd IEEE Conf. on Decision and Control, CDC. vol. 1, pp. 479–484 Vol.1 (2004)
6. Mitsch, S., Platzer, A.: ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th Int. Conf., RV 2014. LNCS, vol. 8734, pp. 199–214. Springer (2014)
7. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)

[3] SAfe Flow-component Editor for Traffic networks,
available online: `http://www.tk.jku.at/people/mueller/publications/itsc15/`