Position paper on Usable Verification

Saddek Bensalem Joseph Sifakis

Verimag Laboratory, Université Joseph Fourier Grenoble, CNRS

1 Introduction

Verification techniques have definitely found important applications. After the first two decades of intensive research and development, recent years have been characterized by a shift in focus and intensity. Today we have fairly efficient verification algorithms. However, all suffer from well-known inherent complexity limitations when applied to large systems. To cope with this complexity, we should move from monolithic verification to compositional techniques. We need divide-and-conquer approaches for inferring global properties of a system from the properties of its components. The current state-of-the-art does not meet our initial expectations. The main approach is by "assume-guarantee", where properties are decomposed into two parts. One is an assumption about the global behavior of the system within which the component resides; the other is a property guaranteed by the component when the assumption about its environment holds. As discussed in a recent paper [13], many issues make it difficult to apply assume-guarantee rules, in particular because synthesis of assumptions (when feasible) may cost as much as monolithic verification.

In our opinion, any general compositional verification theory will be highly intractable and will be of theoretical interest only. We need to study compositionality results for particular classes of properties and/or particular classes of systems. For instance, finding compositional verification rules guaranteeing deadlock-freedom or mutual exclusion instead of investigating rules for safety properties in general. Potential deadlocks can be found by analysis of dependencies induced by interactions between components. For proving mutual exclusion, a different type of analysis is needed.

The results thus obtained should allow us to identify "verifiability" conditions (i.e., conditions under which verification of a particular property and/or class of systems becomes scalable). This is similar to finding conditions for making systems testable, adaptable, etc. In this manner, compositionality rules can be turned into correct-by-construction techniques.

To illustrate the above ideas, we present a heuristic method for checking deadlock-freedom of component-based systems. The method is compositional and consists in computing a particular type of global invariants of a composite component as the conjunction of two types of invariants: invariants of its constituent components and interaction invariants. These characterize the effect of the interactions on the global behaviour. They are computed symbolically by resolving a set of boolean constraints on the global state space. The method is an iterative process for computing progressively stronger global invariants. It concludes deadlock-freedom if a global invariant is found such that it does not contain deadlock states. It has been implemented in the D-Finder tool. Experimental results on non-trivial case studies show that the method scales up smoothly and over-performs monolithic verification techniques.

The paper is structured as follows. First, we present the principle of the method and its theoretical justification. Second, we describe the general architecture of the D-Finder tool. Third, we provide experimental results

for various case studies. Finally, we conclude by a discussion on the advantages and limitations of the presented approach.

2 The principle of the method

In [1], we have presented a new and efficient method for verifying properties of concurrent systems. Key to that method is the tight approximation of the reachable states of a composite component by compositional computation of invariants based on the following rule:

$$\frac{\{B_i{<}\Phi_i{>}\}_i, \Psi \in II(\|_\gamma\{B_i\}_i, \{\Phi_i\}_i), (\bigwedge_i \Phi_i) \wedge \Psi \Rightarrow \Phi}{\|_\gamma\{B_i\}_i{<}\Phi{>}}$$

The rule allows to compute a global invariant Phi of a composite component $||(_\gamma\{B_i\}_i$ from invariants $Phi_i$of its atomic components B_i and an interaction invariant Ψ. The latter characterizes the effect of the composition of

This is a general rule for proving safety properties. We specialize it for verifying deadlock-freedom of composite components by computing a specific class of interaction invariants. The method consists in checking if the obtained global invariant $(\bigwedge_i \Phi_i) \wedge \Psi$ implies $\neg\mathcal{DIS}$, where $\mathcal{DIS}$ is a state predicate characterizing all deadlock states of the composite component.

Computing Component Invariants. The component invariants Φ_i are invariants that over-approximate the reachable states of atomic components B_i. The behavior of B_i is described as an automaton extended with local data. each one of its transitions is labeled with a port p, a guard g (boolean condition on data) and a function f (data transformation) . From a given state (control location and data valuation), the transition is enabled when an interaction involving p is enabled and its guard is true.

A component invariant Φ_i of B_i is the conjunction assertions holding at control locations, computed iteratively. If ϕ_i is the assertion associated with location l_i, ϕ_{i+1} is obtained by conjuncting ϕ_i with the post-conditions of the assertions $\_phi_j$ associated with locations l_j which are direct predecessors of l_i (see figure). Initially, locations are assigned the assertion true. This computation continuously leads to stronger invariants until eventually a fixed-point is reached. For finite state atomic components B_i, the set of the reachable states can be computed and used as it is the strongest component invariant Φ_i. See [2] for more details.

Computing Interaction Invariant. The *interaction invariant* Ψ captures constraints on the behavior of the system that are induced by the synchronization of the components. Static analysis of the parallel components and their interactions gives *Boolean behavioral constraints* [3], which are structural properties of concurrent systems that allow to relate the communication between different components with their internal transitions and hence model a unified transition relation. Solutions of BBCs can be used to symbolically compute a strong interaction invariant.

This method has been implemented in *D-Finder*, a tool which takes as input programs described in the *BIP* (Behavior, Interaction, Priority) [4] language - *BIP* is a new (but already widely used) tool for component-based design. It has been shown that *D-Finder* is capable of checking deadlock-freedom of non trivial programs. In addition to composionality, ones can exploit incrementality of the design process. Incremental system design proceeds by adding new interactions to existing sets of components. Each time an interaction is added, it is possible to verify whether the resulting system violates a given property and discover design erros as soon as they appear.

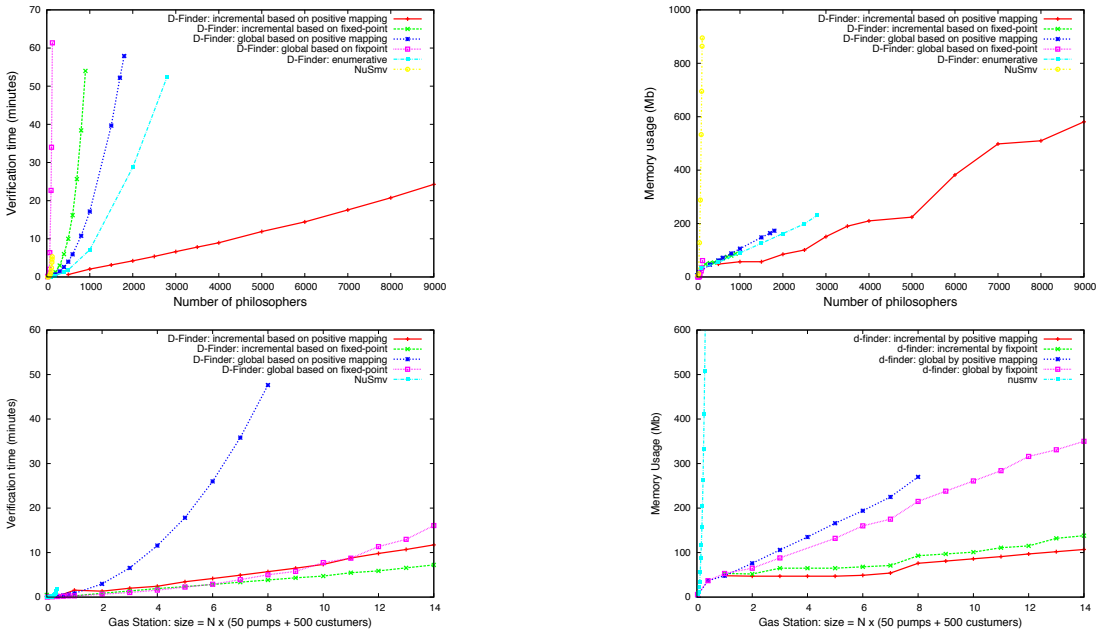

Figure 1: D-Finder results: Verification time (left) and memory usage (right) for two classical benchmarks, dining philosopher (up) and Gas Station (down)

We have proposed a new technique for incremental construction and verification of component-based systems [3]. The technique is based on the use of sufficient conditions that ensure the preservation of invariants when new interactions are added along the component construction process. When these conditions are not satisfied, new invariants are generated by reusing invariants of the interacting components. Reusing invariants reduces considerably the verification effort.

Figure 1 shows deadlock verification times as a function of complexity measured by the number of composed components for i) monolithic verification by using NuSmv; ii) compositional verification; iii) incremental verification.

3 The D-Finder tool

Figure 2 gives an overview of the main modules of the tool: *Model*, *Analysis*, and *Expression* handling. The *Model* block handles the parsing of the *BIP* code into an internal model and provides the means to compute Φ, Ψ, and $\mathcal{DIS}$. These results are passed to the *Analysis* block, which performs further steps like the generation of possible deadlocks and, most recently, generation of counterexamples for Boolean systems (CEX). The *Expression* block allows uniform handling of expressions with data (using the Eclipse Modeling Framework, EMF), and a more succinct representation as BDDs for Boolean systems. Both can be used interchangeably with the respective tools being called transparently for actual computations. Figure 2 shows the use of external tools for models with non-Boolean data; otherwise, BDDs are used in all computation steps. Additionally to the method for computing the interaction invariants Ψ proposed in [1], we now also provide modules for BDD

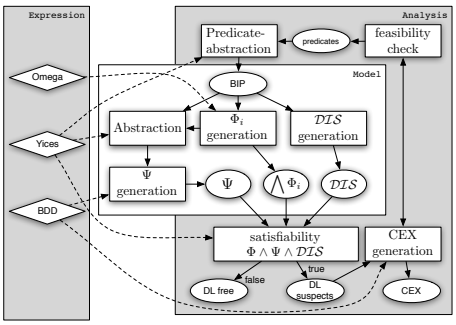

Figure 2: Structure of the D-Finder tool

```
#> dfinder -f p1000.bip --incr_file incr_15.incr
   --method=pm --analysis=dl
#  overall analysis                              :
#    compute II using incremental pm             :
#      Eliminate Variables Abstraction(Phil...   :
#        Compute CI for Philosopher              : 0:01
#      Eliminate Variables Abstraction(Phil...   : 0:02
...
#      get common locations                      : 0:03
#      compute BBCs[0]                           : 0:01
...
#      integrate for increment[1]                : 0:00
...
#      dual compuatation                         : 0:00
#      concretization                            : 0:02
#    compute II using incremental pm             : 0:41
#    incremental DIS                             : 0:24
     Found 1 deadlocks:
#  overall analysis                              : 1:07
```

Figure 3: Call from the command line

based *fixed-point* and *positive mapping* methods in global and incremental versions.

An excerpt of a call to *D-Finder* with 1000 Philosophers in 20 increments (and hence 20 intermediary interaction invariants) is shown in Figure 3. The first step is the computation of an abstraction without variables (using the post conditions from Φ computation (CI) to split the states), followed by the local computations (BCC) for each of the increments and their integration. Computation of the dual and mapping to the concrete values finishes the computation of Ψ (II), which is used to directly compute the intersection with $\mathcal{DIS}$. Finally, the tool successfully reports one deadlock.

4 Case Study: The DALA Robot

We applied the rigorous BIP design flow for the development of a new version of the functional layer of the DALA robot controller [5]. This was initially developed by using the GeNoM framework [6]. The design flow for the functional layer of the robot involves the following steps:

1. Hierarchical decomposition of the functional layer into components. The overall architecture can be represented as a tree. Its root is the function layer and the leaves correspond to atomic components. The grammar below shows how the designed system can be obtained as the incremental composition of components:

   $$\begin{aligned} \textit{Functional Layer} &::= (\ \textit{Module}\ )^+ \\ \textit{Module} &::= (\ \textit{Service}\ )^+ \ .\ (\ \textit{Execution-Task}\ )^+ \ .\ (\ \text{Poster}\ )^+ \\ \textit{Service} &::= (\ \text{Service-Controller}\ )\ .\ (\ \text{Activity}\ ) \\ \textit{Execution-Task} &::= (\ \text{Timer}\ )\ .\ (\ \text{Scheduler-Activity}\ ) \end{aligned}$$

2. Description of the behavior of each atomic component,
3. Description of composite components as the composition of atomic components by using only interactions and priorities, without adding additional behavior. This is possible because BIP is expressive enough for expressing any kind of coordination by using only architectural constraints.

Using this approach we have been able to:

- built a complete BIP model of the functional layer for the DALA robot, from which we generated C++ code (500K lines),
- formally verify by using D-Finder that the BIP model is deadlock-free and also that it satisfies other safety properties such as data freshness. Results about the models complexity and verification times are reported in figure 1.

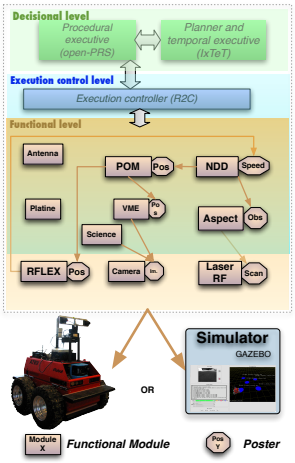

Figure 4: An overview of the DALA system

- synthesize of a controller that encodes and enforces safety properties, thereby facilitating the development of safe and dependable robotic architectures,
- run experiments with the code generated automatically from the BIP model on the DALA rover, and to demonstrate via fault injections that the BIP engine successfully stops the robot from reaching undesired/unsafe states.

module	component	location	interaction	states	time (minutes)
SICK	43	213	202	$2^{20} \times 3^{29} \times 34$	1:22
Aspect	29	160	117	$2^{17} \times 3^{23}$	0:39
NDD	27	152	117	$2^{22} \times 3^{14} \times 5$	8:16
RFLEX	56	308	227	$2^{34} \times 3^{35} \times 1045$	9:39
Battery	30	176	138	$2^{22} \times 3^{17} \times 5$	0:26
Heating	26	149	116	$2^{17} \times 3^{14} \times 145$	0:17
Platine	37	174	151	$2^{19} \times 3^{22} \times 35$	0:59

Table 1: Deadlock-freedom checking results on DALA modules. For every module, we provide the number of atomic components, the (overall) number of control locations, the number of interactions, an over-approximation of the number of reachable states and the total verification time. All modules have been proven deadlock-free.

5 Discussion

Most usable verification techniques consist in computing invariants (static analysis, abstract interpretation, model checking for safety properties). Depending on the type of properties to be verified, the invariants are expressed in an adequately chosen language e.g. boolean algebra, Presburger arithmetic, polynomial constraints.

We show that deadlock-freedom can be checked compositionally to overcome limitations of monolithic verification. A key idea is separate computation of component invariants and interaction invariants. The latter characterize how the product state space is restricted by interactions. Moreover, interaction invariants can be computed symbolically from a set of boolean constraints on n variables where n is bounded by the total number of control locations of the system.

Deadlocks are the most common sources of errors in asynchronous component-based systems where interaction is based on strong synchronization. The application of the proposed technique to systems modelled in BIP proves to be particularly effective as shown by experimental results. BIP uses a high level language to describe incrementally the coordination of components by using hierarchically structured synchronization constraints. Deadlocks are the most likely design errors induced by such constraints.

The application of our approach for other properties e.g. mutual exclusion will need a different type of analysis to compute interaction invariants and also a more expressive language for their description.

References

[1] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional verification for component-based systems and application. In *ATVA*, pages 64–79, Seoul, 2008.

[2] Saddek Bensalem and Yassine Lakhnech. Automatic generation of invariants. *Formal Methods in System Design*, 15(1):75–92, 1999.

[3] S. Bensalem, M. Bogza, A. Legay, T. H. Nguyen, J. Sifakis, and R. Yan. Incremental component-based construction and verification using invariants. In *FMCAD*, 2010.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06*, pages 3–12, Washington, DC, USA, 2006.

[5] A. Basu, S. Bensalem, M. Gallien, F. Ingrand, C. Lesire, T.H. Nguyen, and J. Sifakis. Incremental component-based construction and verification of a robotic system. In *18th European Conf. on Artificial Intelligence (ECAI)*, 2008.

[6] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation*, Atlanta, USA, 1994.