Network invariants for real-time systems

Olga Grinchtein1,∗ and Martin Leucker2,∗∗
1Department of Computer Systems, Uppsala University, Uppsala, Sweden. E-mail: olgag@it.uu.se
2Institut für Informatik, TU München, Munich, Germany. E-mail: leucker@in.tum.de

Abstract. We extend the approach of model checking parameterized networks of processes by means of network invariants to the setting of real-time systems. We introduce timed transition structures (which are similar in spirit to timed automata) and define a notion of abstraction that is safe with respect to linear temporal properties. We strengthen the notion of abstraction to allow a finite system, then called network invariant, to be an abstraction of networks of real-time systems. In general the problem of checking abstraction of real-time systems is undecidable. Hence, we provide sufficient criteria, which can be checked automatically, to conclude that one system is an abstraction of a concrete one. Our method is based on timed superposition and discretization of timed systems. We exemplify our approach by proving mutual exclusion of a simple protocol inspired by Fischer’s protocol, using the model checker TLV.

Keywords: Network invariants; Real-time systems; Parameterized systems

1. Introduction

Model checking [CGP99] is a method for verifying concurrent systems: computations of a high-level description of a system are compared to those formulated by a logical requirement specification to establish that they are compatible. In linear temporal logic (LTL), which was proposed for specification purposes by Pnueli [Pnu77], one defines when a single computation meets the specification. A system is then said to satisfy a specification, if every computation satisfies the specification.

Checking LTL specifications of finite-state systems is well understood [LP85, VW86, Var96]. Faced with concurrent systems consisting of an arbitrary number of processes working in parallel, however, model checking is more challenging, since we have to deal with unboundedly many states. A fruitful approach for checking these parameterized systems, as they are often called, is by use of abstraction and network invariants.

The idea of abstraction is to check a smaller, finite-state system instead of the original one. If the smaller system has more computations—including those of the original one—every linear temporal logic property it satisfies, also holds for the original system [Gru05]. For branching-time logics, similar ideas work if we restrict the logic to a universal fragment [CGL92]. Abstractions of original systems may either be found manually or using ideas of abstract interpretation. In the first case, one has to prove that the abstract system indeed comprises all computations of the original one. Basic principles underlying the construction of abstract models are understood from, e.g., [CC77, CGL92, DGG94].

Verification by means of network invariants was introduced in [WL89] and turned into a working method in [KM95]. In a nutshell, the idea can be sketched as follows. Suppose we have a finite-state process Φ, e.g., repeatedly requesting and releasing some resource. We want to reason about a setting in which an arbitrary

Correspondence and offprint requests to: M. Leucker, E-mail: leucker@in.tum.de

∗ Part of this work was done during O. Grinchtein’s stay at Weizmann Institute.
** This author was supported by the European Research Training Network “Games”.
number of instances of \( \Phi \) work in parallel. In other words, we study the system \( \Phi_1 \parallel \cdots \parallel \Phi_n \), where the number \( n \) of instances of \( \Phi \) is not known in advance. While for every \( n \), we deal with a finite-state system, it is clearly not possible to check the system iteratively for all \( n \).

Using the idea of abstraction, it suffices to find a finite-state system \( \Phi_A \) that satisfies our requirement specification and that abstracts \( \Phi_1 \parallel \cdots \parallel \Phi_n \) for arbitrary \( n \). Similar to induction over natural numbers, the latter is implied—with some further constraints—if \( \Phi_A \) is an invariant, i.e., \( \Phi_A \) is an abstraction of \( \Phi \) as well as of \( \Phi_A \parallel \Phi_A \). The first item shows that \( \Phi \parallel \cdots \parallel \Phi \) can be abstracted by \( \Phi_A \parallel \cdots \parallel \Phi_A \), which can further be abstracted by \( \Phi_A \), using the second requirement.

In this way, checking a parameterized system is reduced to finding a possible network invariant that satisfies the requirement imposed on the parameterized system and proving that it is indeed a network invariant. Finding a possible invariant is usually carried out manually, and checking whether it satisfies the requirement specification can be done automatically using model checking. Proving that a system is a network invariant can be reduced to checking abstraction, which can be done automatically for finite-state systems. This approach is elaborated for checking linear-time specifications of fair discrete systems in [KP00] where also heuristics for finding invariants are given.

Traditional techniques for model checking do not admit explicit modeling of time, and are thus unsuitable for the analysis of real-time systems. Alur and Dill [AD90, AD94] introduced timed automata to model the behavior of real-time systems. Furthermore, model-checking techniques were developed. See [Alu99] for an overview.

In this paper we study the problem of reasoning about parameterized timed systems. It extends our previous paper [GL04], in which, to the best of our knowledge, the first approach for studying network invariants in the sense of [WL89] for networks of timed systems was carried out, in terms of further explanation and full proofs. We follow the framework given in [KP00], in which networks of fair discrete systems were examined. However, we enrich the underlying systems with clocks to model timed behavior. We extend the notion of abstraction and network invariant to the timed setting. A main contribution of the paper is a procedure for checking whether a given timed transition structure is an abstraction of another one.

We introduce timed transition structures which are similar to timed automata. The main differences are that we distinguish between private and global variables and that communication is by shared variables instead of message passing. Thus, our communication model is closer to Java-like concurrent programming languages. We say that \( \Phi_A \) is an abstraction of \( \Phi \) if \( \Phi_A \) is comprised of at least the computations of \( \Phi \). The idea is used, e.g., in [AL91]. We show that our notion of abstraction is safe with respect to linear temporal logic, i.e., linear-time properties of an abstract system also hold for a concrete one. Provided further environmental behavior is taken into account, we show that checking whether a system is a network invariant can be reduced to checking whether \( \Phi_A \) is an abstraction of \( \Phi \) and \( \Phi_A \parallel \Phi_A \). Note that although clocks can be understood as real valued variables, they are different from ordinary data variables since time progresses for all clocks synchronously: if time \( \delta \) passes for clock \( x \), then it also passes for clock \( y \). Treating clocks just as real valued variables would disregard this “hidden” correlation of clocks and lead to wrong conclusions. This implicit dependency of clocks is one of the obstacles to overcome when extending the approach of network invariants to the setting of real-time systems.

We provide sufficient criteria, which can be checked automatically, to conclude that a system is an abstraction of a concrete one. Our method is based on superposition [Jon94] but extended to the timed setting. The superposition of \( \Phi \) and \( \Phi_A \) is a structure similar to a timed transition structure, whose computations can be projected to computations of \( \Phi \) and \( \Phi_A \), where as best as possible, \( \Phi_A \) tries to follow the moves of \( \Phi \). We show that if the superposition satisfies certain LTL properties, \( \Phi_A \) is indeed an abstraction of \( \Phi \).

To check whether a superposition satisfies LTL properties, we use the notion of discretization of timed transition structures, developed by [GPV94] and [ABK*97]. Hereby, the infinite state space of timed transition structure is reduced to a finite-state systems, maintaining satisfaction of LTL properties. This allows us to use standard verification tools, like TLV [PS96]. Note that the method of discretization used is just one of many that turned out to be useful in our practical examples. See [OW03, Gri02] for further results on discretization as well as further references. Our approach is exemplified by proving mutual exclusion of a simple protocol inspired by Fischer’s protocol [SBM92], using the model checker TLV [PS96]. We restrict ourselves to finite domains of data variables. Using predicate abstraction (see [GS97], [KP00]), it should be possible to extend our results to systems with variables ranging over infinite domains.

Systems similar to our timed transition structures have been studied in [LS00]. The approach is based on automatic abstraction, but is limited to checking safety properties of timed systems with integer time domain. A different approach for studying parameterized systems is presented in [AJ99] and [AJ02]. It is based on finite