Analysis of Message Passing Programs
Using SMT-Solvers

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg
Uppsala University, Sweden

Abstract. We consider message passing programs where processes communicate asynchronously over unbounded channels. The reachability problem for such systems are either undecidable or have very high complexity. In order to achieve efficiency, we consider the phase-bounded reachability problem, where each process is allowed to perform a bounded number of phases during a run of the system. In a given phase, the process is allowed to perform send or receive transitions (but not both). We present a uniform framework where the channels are assigned different types of semantics such as lossy, stuttering, or unordered. We show that the framework allows a uniform translation of bounded-phase reachability for each of the above mentioned semantics to the satisfiability of quantifier-free Presburger formulas. This means that we can use the full power of modern SMT-solvers for efficient analysis of our systems. Furthermore, we show that the translation implies that bounded-phase reachability is NP-COMPLETE. Finally, we prove that the problem becomes undecidable if we allow perfect channels or push-down processes communicating through (stuttering) lossy channels. We report on the result of applying the prototype on a number of non-trivial examples.

1 Introduction

Programs modeled as message passing processes have a wide range of applications including communication protocols [13, 5], programs operating on weak memory models [37], WEB service protocols [26], and as semantic models for modern languages such as ERLANG [30] and SCALA [31]. Typically, the processes exchange information asynchronously through a shared unbounded data structure, e.g., counters, multisets, and channels. Despite the increasing popularity of such program models, precise algorithmic analysis is still a major challenge. This is perhaps not without a good reason: it is well known that basic analysis problems (e.g., state reachability) are undecidable for processes communicating via perfect FIFO channels [13], even under the assumption that each process is finite-state. Although, checking state reachability becomes decidable for (important) special cases such as lossy FIFO channels [1], or unordered channels [25], the algorithms have very high complexity (non-primitive recursive for lossy channels [28] and EXPSPACE-HARD for unordered channels [20]).

Given the importance of concurrent software, much research has been devoted in recent years to developing practically useful algorithms. The undecidability and high complexity obstacles are usually addressed by considering different types of over- or under-approximations of system behavior (e.g., [16, 4, 9, 12, 26, 8, 11, 10, 32, 17, 15]).
One useful approach that has recently been proposed is context-bounding \cite{24}. The idea is to only consider computations performing at most some fixed number of context switches between processes. This provides a trade-off between computational complexity and verification coverage: on the one hand, context-bounded verification can be more efficient than unbounded verification; and on the other hand, many concurrency errors, such as data races and atomicity violations, are manifested in executions with few context switches \cite{22}.

In this paper, we present a new approach to model checking of concurrent processes that communicate through channels. We introduce a new bounding parameter in the behavior of such systems, namely the number of alternations between send operations and receive operations performed by each processes. We consider the bounded-phase reachability problem, where each process is restricted to performing at most \( k \) phases (for some natural number \( k \)). A phase is a run where the process performs either send or receive operations (but not both). Notice that the bounded-phase restriction does not limit the number of sends or receives, and in particular it does not put any restriction on the length of the run. Also, the number of context switches is not limited. We will present a framework and instantiate it for several variants of channel semantics, such as lossy, stuttering, and multiset that allow the messages inside the channels to be lost, duplicated, and re-ordered respectively. One main contribution of this paper is to show that our framework allows to translate (in polynomial time) the bounded-phase reachability problem to the satisfiability of quantifier-free Presburger formulas. This opens the way to leveraging the full power of state-of-the-art SMT-solvers for obtaining a very efficient solution to the bounded-phase reachability problem for all above mentioned models. We perform the translation in two steps. First, we show that bounded-phase reachability can be reduced to (general) reachability under a new restriction, namely that we only consider simple computations. A computation is simple if any (local) state of a process appears at most once along the computation. In the second step, we show that simple reachability can be captured by satisfiability of a quantifier-free Presburger formula (that we can then feed to an SMT-solver).

In order to simplify the presentation, we first describe our framework for lossy channel systems LCS. Then, we describe how the method can be modified (in a straightforward manner) to the other channel semantics. Also, as consequence of our translation, we show that bounded-phase reachability for LCS (and the other models) is \textsc{NP}-complete. This is to be contrasted with the fact that the general reachability problem is not primitive recursive.

Finally, we show undecidability of bounded-phase reachability for several cases, e.g., under the perfect channel semantics, or under the (stuttering) lossy semantics when one of the processes is allowed to have a (single) stack.

We have implemented our method in a prototype that we have applied on a number of examples with promising results. The examples span several application areas, such as WEB service protocols, communication protocols, and multithreaded programs counters. The prototype and the details of the examples and experimentation are available online (see Section \ref{sec:examples}).