Symbolic systems, explicit properties: on hybrid approaches for LTL symbolic model checking

Roberto Sebastiani · Stefano Tonetta · Moshe Y. Vardi

Abstract In this work we study hybrid approaches to LTL symbolic model checking; that is, approaches that use explicit representations of the property automaton, whose state space is often quite manageable, and symbolic representations of the system, whose state space is typically exceedingly large. We compare the effects of using, respectively, (i) a purely symbolic representation of the property automaton, (ii) a symbolic representation, using logarithmic encoding, of explicitly compiled property automaton, and (iii) a partitioning of the symbolic state space according to an explicitly compiled property automaton. We apply this comparison to three model-checking algorithms: the doubly-nested fixpoint algorithm of Emerson and Lei, the reduction of emptiness to reachability of Biere et al., and the singly-nested fixpoint algorithm of Bloem et al. for weak automata. The emerging picture from our study is quite clear, hybrid approaches outperform pure symbolic model checking, while partitioning generally performs better than logarithmic encoding. The conclusion is that the hybrid approaches benefit from state-of-the-art techniques in semantic compilation of LTL properties. Partitioning gains further from the fact that the image computation is applied to smaller sets of states.

Keywords Linear-time logic · Symbolic model checking · Property-driven partitioning

1 Introduction

Linear-temporal logic (LTL) [30] is a widely used logic to describe infinite behaviors of discrete systems. Verifying whether an LTL property is satisfied by a finite transition system is a core problem in Model Checking (MC). The key idea of the automata-theoretic approach to MC is that LTL formulas can be compiled into equivalent automata with fairness conditions, i.e. conditions on which infinite words are accepted. Standard techniques consider the formula \( \varphi \) that is the negation of the desired behavior and construct a Generalized Büchi automaton (GBA) \( A_\varphi \) with the same language. Then, they compute the product of this automaton \( A_\varphi \) with the system \( M \) and check for emptiness. To check emptiness, one has to compute the set of fair states, i.e. those states of the product automaton that are extensible to a fair path. The main obstacle to model checking is the state-space explosion; that is, the product is often too large to be handled.

Explicit-state model checking uses highly optimized LTL-to-GBA compilation, cf. [14,15,17,20,21,23,24,32,35], which we refer to as semantic compilation. Such compilation may involve an exponential blow-up in the worst case, though such blow-up is rarely seen in practice. Emptiness checking is performed using either a nested depth-first search [13,33] or an optimized decomposition into strongly connected components [11,25]. To deal with the state-explosion problem, various state-space reductions are used, e.g. [29,37].

Symbolic model checking (SMC) [2] tackles the state-explosion problem by representing the product automaton symbolically, usually by means of (ordered) BDDs. The compilation of the property to symbolically represented GBA is purely syntactic, and its blow-up is linear (which
induces an exponential blow-up in the size of the state space), cf. [9]. Symbolic model checkers typically compute the fair states by means of some variant of the doubly-nested-fixpoint Emerson–Lei algorithm (EL) [16,18,31]. For “weak” property automata, the doubly-nested fixpoint algorithm can be replaced by a singly-nested fixpoint algorithm [6]. An alternative algorithm [1] reduces emptiness checking to reachability checking (which requires a singly-nested fixpoint computation) by doubling the number of symbolic variables.

Extant model checkers use either a pure explicit-state approach, e.g. in SPIN [26], or a pure symbolic approach, e.g. in NuSMV [8]. Between these two approaches, one can find hybrid approaches, in which the property automaton, whose state space is often quite manageable, is represented explicitly, while the system, whose state space is typically exceedingly large, is represented symbolically. For example, the singly-nested fixpoint algorithm of [6] is based on an explicit construction of the property automaton. (See [3,12] for other hybrid approaches.)

In [34], motivated by previous work on generalized symbolic trajectory evaluation (GSTE) [40], we proposed a hybrid approach to LTL model checking, referred to as property-driven partitioning (PDP). In this approach, the property automaton \( A_p \) is constructed explicitly, but its product with the system is represented in a partitioned fashion. If the state space of the system is \( S \) and that of the property automaton is \( B \), then we maintain a subset \( Q \subseteq S \times B \) of the product space as a collection \( \{ Q_b : b \in B \} \) of sets, where each \( Q_b = \{ s \in S : (s, b) \in Q \} \) is represented symbolically. Thus, in PDP we maintain an array of BDDs instead of a single BDD to represent a subset of the product space. Based on extensive experimentation, we argued in [34] that PDP is superior to SMC, in many cases demonstrating exponentially better scalability.

While the results in [34] are quite compelling, it is not clear why PDP is superior to SMC. On one hand, one could try to implement PDP in a purely symbolic manner by ensuring that the symbolic variables that represent the property-automaton state space precede the variables that represent the system state space in the BDD variable order. This technique, which we refer to as top ordering, would, in effect, generate a separate BDD for each block in the partitioned product space, but without generating an explicit array of BDDs, thus avoiding the algorithmic complexity of PDP. It is possible that, under such variable order, SMC would perform comparably (or even better) than PDP. On the other hand, it is possible that the reason underlying the good performance of PDP is not the partitioning of the state space, but, rather, the explicit compilation of the property automaton, which yields a reduced state space for the property automaton. So far, however, no detailed comparison of hybrid approaches to the pure symbolic approach has been published. (VIS [4] currently implements a hybrid approach to LTL model checking. The property automaton is compiled explicitly, but then represented symbolically, using the so-called logarithmic encoding, so SMC can be used. No comparison of this approach to SMC, however, has been published.) Interestingly, another example of property-based partitioning can be found in the context of explicit-state model checking [22].

In this paper, we undertake a systematic study of this spectrum of representation approaches: purely symbolic representation (with or without top ordering), symbolic representation of semantically compiled automata (with or without top ordering), and partitioning with respect to semantically compiled automata (PDP). An important observation here is that PDP is orthogonal to the choice of the fixpoint algorithm. Thus, we can study the impact of the representation on different algorithms; we use here EL, the reduction of emptiness to reachability of [1], and the singly-nested fixpoint algorithm of [6] for weak property automata. The focus of our experiments is on measuring scalability. We study scalable systems and measure how running time scales as a function of the system size. We are looking for a multiplicative or exponential advantage of one algorithm over another one.

The emerging picture from our study is quite clear, hybrid approaches outperform pure SMC. Top ordering generally helps, but not as much as semantic compilation. PDP generally performs better than symbolic representation of semantically compiled automata (even with top ordering). The conclusion is that the hybrid approaches benefit from state-of-the-art techniques in semantic compilation of LTL properties. Such techniques includes preprocessing simplification by means of rewriting [15,32], postprocessing state minimization by means of simulations [15,17,21,32], and midprocessing state minimization by means of alternating simulations [20,24]. In addition, empty-language states of the automata can be discarded. PDP gains further from the fact that the image computation is applied on smaller sets of states. The comparison to SMC with top ordering shows that managing partitioning symbolically is not as efficient as managing it explicitly.

This paper extends the work presented in [36] by giving:

- a more detailed description of PDP and its relationship with search algorithms; in particular, the algorithm that reduces liveness to safety is described in details both for the explicit-state case and for property-driven partitioning.
- a deeper analysis of the results; in particular, the comparison of the different search algorithms is shown and discussed.

The outline of the paper is as follows. Section 2 contains required background on explicit-state and symbolic model checking. Section 3 describes hybrid approaches to symbolic