On the Use of Automata-based Techniques in Symbolic Model Checking

Invited Address

Axel Legay\textsuperscript{1,3} Pierre Wolper\textsuperscript{2}

\textit{Institut Montefiore}
\textit{Université de Liège, Liège, Belgium}

\section*{Abstract}

Checking infinite-state systems is frequently done by encoding infinite sets of states as regular languages. Computing such a regular representation of, say, the reachable set of states of a system requires acceleration techniques that can finitely compute the effect of an unbounded number of transitions. Among the acceleration techniques that have been proposed, one finds both specific and generic techniques. Specific techniques exploit the particular type of system being analyzed, e.g. a system manipulating queues or integers, whereas generic techniques only assume that the transition relation is represented by a finite-state transducer, which has to be iterated. In this paper, we survey two generic techniques that have been presented in \cite{3} and \cite{4}. Those techniques build on earlier work, but exploit a number of new conceptual and algorithmic ideas, often induced with the help of experiments, that give it a broad scope, as well as good performance.

\textit{Keywords:} \textit{(ω−)}Regular Model Checking, Survey.

At the heart of all the techniques that have been proposed for exploring infinite state spaces, is a symbolic representation that can finitely represent infinite sets of states. In early work on the subject, this representation was domain specific, for example linear constraints for sets of real vectors. For several years now, the idea that a generic finite-automaton based representation could be used in many settings has gained ground, starting with systems manipu-
lating queues and integers [8,11,9,13], then moving to parametric systems [6], and, recently, reaching systems using real variables [10,2].

For exploring an infinite state space, one does not only need a finite representation of infinite sets, but also techniques for finitely computing the effect of an unbounded number of transitions. Such techniques can be domain specific or generic. Domain specific techniques exploit the specific properties and representations of the domain being considered and were, for instance, obtained for queues in [15,14], for integers and reals in [17,22,12], for pushdown system in [18,16], and for lossy channels in [19]. Generic techniques consider finite-automata representations and provide algorithms that operate directly on this representation, mostly disregarding the domain for which it is used.

Generic techniques appeared first in the context of the verification of systems whose states can be encoded by finite words, such as parametric systems. The idea used there is that a configuration being a finite word, a transition relation is a relation on finite words, or equivalently a language of pairs of finite words. If this language is regular, it can be represented by a finite state automaton, more specifically a finite-state transducer, and the problem then becomes the one of iterating such a transducer. Finite state transducers are quite powerful (the transition relation of a Turing machine can be modeled by a finite-state transducer), the flip side of the coin being that the iteration of such a transducer is neither always computable, nor regular. Nevertheless, there are a number of practically relevant cases in which the iteration of finite-state transducers can be computed and remains finite-state. Identifying such cases and developing (partial) algorithms for iterating finite-state transducers has been the topic, referred to as “regular model checking”, of a series of recent papers [1,20,21,5,23].

The question that initiated the work [3] presented in this talk is, whether the generic techniques for iterating transducers could be fruitfully applied in cases in which domain specific techniques had been exclusively used so far. In particular, one of our goals was to iterate finite-state transducers representing arithmetic relations (see [22] for a survey). Beyond mere curiosity, the motivation was to be able to iterate relations that are not in the form required by the domain specific results, for instance disjunctive relations. Initial results were very disappointing: the transducer for an arithmetic relation as simple as \((x, x + 1)\) could not be iterated by existing generic techniques. However, looking for the roots of this impossibility through a mix of experiments and theoretical work, and taking a pragmatic approach to solving the problems discovered, we were able to develop an approach to iterating transducers that easily handles arithmetic relations, as well as many other cases. Interestingly, it is by using a tool for manipulating automata (LASH [24]), looking at ex-
amples beyond the reach of manual simulation, and testing various algorithms that the right intuitions, later to be validated by theoretical arguments, were developed. Implementation was thus not an afterthought, but a central part of our research process.

The general approach that has been taken is similar to the one of [21] in the sense that, starting with a transducer $T$, we compute powers $T^i$ of $T$ and attempt to generalize the sequence of transducers obtained in order to capture its infinite union. This is done by comparing successive powers of $T$ and attempting to characterize the difference between powers of $T$ as a set of states and transitions that are added. If this set of added states, or *increment*, is always the same, it can be inserted into a loop in order to capture all powers of $T$. However, for arithmetic transducers comparing $T^i$ with $T^{i+1}$ did not yield an increment that could be repeated, though comparing $T^{2^i}$ with $T^{2^{i+1}}$ did. So, a first idea we used is not to always compare $T^i$ and $T^{i+1}$, but to extract a sequence of samples from the sequence of powers of the transducer, and work with this sequence of samples. Given the binary encoding used for representing arithmetic relations, sampling at powers of 2 works well in this case, but the sampling approach is general and different sample sequences can be used in other cases. Now, if we only consider sample powers $T^{i_k}$ of the transducers and compute $\bigcup_k T^{i_k}$, this is not necessarily equivalent to computing $\bigcup_i T^i$. Fortunately, this problem is easily solved by considering the reflexive transducer, i.e. $T_0 = T \cup T_I$ where $T_I$ is the identity transducer, in which case working with an infinite subsequence of samples is sufficient.

Once the automata in the sequence being considered are constructed and compared, and that an increment corresponding to the difference between successive elements has been identified, the next step is to allow this increment to be repeated an arbitrary number of times by incorporating it into a loop. There are some technical issues about how to do this, but no major difficulty. Once the resulting “extrapolated” transducer has been obtained, one still needs to check that the applied extrapolation is safe (contains all elements of the sequence) and is precise (contains no more). An easy to check sufficient condition for the extrapolation to be safe is that it remains unchanged when being composed with itself. Checking preciseness is more delicate, but we have developed a procedure that embodies a sufficient criterion for doing so. The idea is to check that any behavior of the transducer with a given number $k$ of copies of the increment, can be obtained by composing transducers with less than $k$ copies of the increment. This is done by augmenting the transducers to be checked with counters and proving that one can restrict these counters to a finite range, hence allowing finite-state techniques to be used.

---

4 as an example, it is often linear when considering parametric systems
Taking advantage of the fact that our extrapolation technique works on automata, not just on transducers, we consider computing reachable states both by computing the closure of the transducer representing the transition relation, and by repeatedly applying the transducer to a set of initial states. The first approach yields a more general object and is essential if one wishes to extend the method to the verification of liveness properties ([1,25]), but the second is often less demanding from a computational point of view and can handle cases that are out of reach for the first. Preciseness is not always possible to check when working with state sets rather than transducers, but this just amounts to saying that what is computed is possibly an overapproximation of the set of reachable states, a situation which is known to be pragmatically unproblematic.

Going further, the problem of using regular model checking technique for systems whose states are represented by infinite (omega) words has been addressed. This makes the representation of sets of reals possible as described in [2,12]. To avoid the hard to implement algorithms needed for some operations on infinite-word automata, only omega-regular sets that can be defined by weak deterministic Büchi automata [7] are considered. This is of course restrictive, but as is shown in [2], it is sufficient to handle sets of reals defined in the first-order theory of linear constraints. Moreover using such a representation leads to algorithms that are very similar to the ones used in the finite word case, and allows us to work with reduced deterministic automata as a normal form. Due to these advantages and properties, one can show that the technique developed for the finite word case can directly be adapted to weak deterministic Büchi automata up to algorithmic modifications [4].

Our technique has been implemented in a prototype that relies in part on the LASH package. This prototype has been tested on several case studies coming from different horizons. In our experiments, we were able to iterate a variety of arithmetic (integer or real) transducers. We were also successful on disjunctive relations that could not be handled by earlier specific techniques such as [17,12]. The technique was also successfully applied to examples of parametric systems and to the analysis of Petri nets. Moreover models of hybrid systems, including a leaking gas burner and an alternating bit protocol with timers were also considered.

Attempting to verify infinite-state systems while working exclusively with automata-theoretic representations and algorithms can appear as a somewhat quixotic endeavor. However, practical results clearly shown their interest, and are thus a motivation for new developments [27,26,28].
References


