Volume 2, Issue 3, March 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Infinite Automata And Formal Verification

Aditya Harbola*
School of computing
Graphic Era University,
Dehradun, UK
adityaharbola@gmail.com

Deepti Negi
School of computing
Graphic Era University, Dehradun,
UK

Deepak Harbola
Department of computer science
BCTKEC, Dwarahat
UK

Abstract- **Infinite-state automata are a new invention: they are automata that have an infinite number of states represented by words, transitions defined using rewriting, and with sets of initial and final states. Also Automata on infinite words (ω-automata) have wide applications in formal language theory as well as in modeling and verifying reactive systems. Automata on infinite objects are extensively used in system specification, verification, and synthesis. While some applications of the automata-theoretic approach have been well accepted by the industry(FSA,PDA), some have not yet been reduced to practice. Applications that involve determinization of automata on infinite words have been assumed to belong to the second category. This has to do the fact that the state space that results from determinization is awfully complex and is not open to optimizations and a symbolic implementation, and the fact that determinization requires the introduction of acceptance conditions that are more complex than the B¨uchi acceptance(B¨uchi automata) condition. Examples of applications that involve determinization and belong to the unfortunate second category include model checking of ω-regular properties, decidability of branching temporal logics, and synthesis and control of open systems.**
***Keyword's*: Automaton, Infinite automata, Model checking, Source of infinity, Presburger Temporal Logic**

1. INTRODUCTION

The theory of infinite-state automata is a new area of research. Infinite-state automata are automata with infinitely many states that can read finite words and accept or reject them, in much the same way as finite-state automata would. In order to represent infinite-state automata using finite means, the states, the transitions, and the initial and final state sets are represented symbolically. The infinite-state automata are defined by using words to represent the states of the automaton.

Automata on infinite objects are extensively used in system specification, verification, and synthesis. While some applications of the automata-theoretic approach have been well accepted by the industry (FSA, PDA), some have not yet been reduced to practice. Applications that involve determinization of automata on infinite words have been assumed to belong to the second category. This has to do the fact that the state space that results from determinization is awfully complex and is not open to optimizations and a symbolic implementation, and the fact that determinization requires the introduction of acceptance conditions that are more complex than the B¨uchi acceptance(B¨uchi automata) condition. Examples of applications that involve determinization and belong to the unfortunate second category include model checking of ω-regular properties, decidability of branching temporal logics, and synthesis and control of open systems.

Let a finite alphabet Σ as the input alphabet for the infinite-state automata. The set of states of an infinite-state automaton over Σ are words over a finite alphabet Π (which is not related to Σ). The initial and final sets of states of this automaton are defined using word-languages over Π accepted by finitely presented devices (e.g. finite-state automata over Π). Transitions between states are defined using rewriting rules that rewrite words to other words: for each $a \in \Sigma$, we have a rewrite rule that rewrites words over Π. A state $u \in \Pi^*$ leads to state $u' \in \Pi^*$ on $a \in \Sigma$ iff the rewrite rule for **a** can rewrite u to u′. There is a variety of choices for the power of rewriting, but in any case the rewriting rules are presented finitely, using finite transducers. The language accepted by an infinite state automaton is defined in the natural way: a word $w \in \Sigma^*$ is accepted if there is a path from some initial state to some final state tracing w in the automaton.

2. INFINITE AUTOMATA DEFINITION

Infinite-state automata are radically different computation models than Turing machines. The notion that rewriting words (or terms) can be a basis for defining computability. Formal languages defined using grammars (the Chomsky hierarchy) are also in the spirit of rewriting, corresponding to unrestricted grammars and hence Turing machines. While Turing machines can be viewed as rewrite systems (rewriting one configuration to another), the study of computational complexity is often based on time and space constraints on the Turing machine model, and natural counterparts to complexity classes in terms of rewrite systems don't currently exist. Infinite state automata where states are finite words, initial and final sets are defined using regular languages, and transitions are defined using rational relations, accept precisely the class of context-sensitive languages (nondeterministic linear-space languages). Rational relations are relations $R \subseteq \Pi^* \times \Pi^*$ that can be effected by finite-state automata: $(u, u') \in R$ iff the automaton can read u on an input tape and write u′ on the output tape, where the two tape heads are only allowed to move right (but can move independent

of each other).The only constraint is the power of rewriting (rational relations) and there is no perceived limit on space or time. In other words, the constraint on rewriting is a qualitative constraint with no apparent restriction on complexity.

Indeed, even establishing the upper bound, namely that these automata define languages that are accepted by linear-bounded Turing machines is non-trivial. A naive simulation of the infinite-state automaton will not work as the words that represent states on the run can be unboundedly large even for a fixed word w. Notice that we do not allow ε-transitions in infinite automata, as allowing that would make infinite automata with even regular rewriting accept the class of all recursively enumerable languages.

A simple generalization of regular rewriting is pushdown rewriting, where we allow the rewriting automata the use of a work stack which can be used for intermediate storage when rewriting a word to another. For example the relation $\{(w, ww^r) | w \in \Pi^*\}$(where w^r denotes the reverse of w) is not regular but can be effected by a pushdown rewrite system. However, defining infinite automata with the power of pushdown rewriting quickly leads to undecidability of the membership problem, and these automata can accept non-recursive languages.

Given a word $w \in \Sigma^*$, note that infinite automata have possibly an *infinite* number of paths on w. Hence, deciding whether w is accepted by the infinite-state automaton is in no way trivial. However, if rewriting rules can be simulated by Turing machines (which will usually be the case), the language accepted by the infinite-state automaton is recursively enumerable.

A: Infinite Automata Defined By Multi-stack Pushdown Transducers

We define now *infinite-state* automata over an alphabet Σ. The states in this automaton will correspond to words over an alphabet Π, the set of states one can transition to from a state on a letter d in Σ will be defined using a multi-stack pushdown transducer corresponding to d, and initial and final state sets will be identified using regular sets of words over Π.

Fix a finite alphabet Σ. An infinite-state pushdown transducer automaton (PTA) over Σ is a tuple A = (Π, {Td} d ∈ Σ, Init, Final), where Π is a finite alphabet, for each d ∈ Σ, Td is a pushdown transducer over Π, and Initial and Final are finite-state automata (NFAs) over Π.

B: The Power Of Multi-stack Rewriting

Multi stack rewriting is a powerful tool for infinite automata's. While infinite automata capture fairly complex languages, there has been little study done on how simple infinite automata can be designed to solve natural algorithmic problems.

C: Term-automatic Rewriting

Another way to define infinite automata is to represent states using *terms* (or trees), and use term rewriting to define relations. In [9], term automatic rewriting infinite automata are considered, and it is shown that they precisely capture ETIME (the class of languages accepted by Turing machines in time $2^{O(n)}$).

3. SOURCE OF INFINITY

There are many areas which can have infinite state machines. The states are not deterministic because of the complex nature of the problem domain. According to [3], there are at least the following five 'sources of infinity':

- Data structures over infinite domains
- Unbounded control structures
- Asynchronous communication
- Parameterization of distributed systems
- Real-time constraints

In computer science, real-time computing (RTC), or reactive computing, is the study of hardware and software systems that are subject to a "real-time constraint, operational deadlines from event to system response. Real-time programs must guarantee response within strict time constraints. Often real-time response times are understood to be in the order of milliseconds and sometimes microseconds. In contrast, a non-real-time system is one that cannot guarantee a response time in any situation, even if a fast response is the usual result.

A distributed parameter system (as opposed to a lumped parameter system) is a system whose state space is infinite-dimensional. Such systems are therefore also known as infinite-dimensional systems. Typical examples are systems described by partial differential equations or by delay differential equations.

In telecommunications, asynchronous communication is transmission of data without the use of an external clock signal, where data can be transmitted intermittently rather than in a steady stream. Any timing required to recover data from the communication symbols is encoded within the symbols. The most significant aspect of asynchronous communications is variable bit rate, or that the transmitter and receiver clock generators do not have to be exactly synchronized.

4. INFINITE AUTOMATA FORMAL VERIFICATION, APPROACHES

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. Formal verification can be helpful in proving the correctness of systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code.

The verification of systems is done by providing a formal proof on an abstract mathematical model of the system, the correspondence between the mathematical model and the nature

of the system being otherwise known by construction. Examples of mathematical objects often used to model systems are: finite state machines, labeled transition systems, Petri nets, timed automata, hybrid automata, process algebra, formal semantics of programming languages such as operational semantics, denotation semantics, axiomatic semantics and Hoare logic.

Initial work on the verification of infinite state systems was done by Bradfield and Stirling [4, 5] who established proof rules for local model checking. However, they were mainly interested in the completeness of the proof rules, and did not regard implementation issues like data structures to represent infinite sets. Some years later, Bultan, Gerber, and Pugh proposed a method for global model checking of infinite state systems by means of Presburger arithmetic [6, 7]. Local model checking for the μ-calculus using Presburger arithmetic was first studied in [8]. A technique for reachability analysis of extended finite state machines using Presburger arithmetic is also described. In this work, finite automata were used for the representation of infinite sets. The use of finite automata in global model checking is also described. Finite automata have also been used in bounded model checking, a variant of global model checking where the fix point computations are approximated according to an a priori given bound. The translation of arithmetic's to finite automata goes back to Büchi.

One approach and formation is model checking, which consists of a systematically exhaustive exploration of the mathematical model (this is possible for finite models, but also for some infinite models where infinite sets of states can be effectively represented finitely using abstraction). Usually this consists of exploring all states and transitions in the model, by using smart and domain-specific abstraction techniques to consider whole groups of states in a single operation and reduce computing time. Implementation techniques include state space enumeration, symbolic state space enumeration, abstract interpretation, symbolic simulation, abstraction refinement. The properties to be verified are often described in temporal logics, such as linear temporal logic (LTL) or computational tree logic (CTL).

A: Model Checking

Model checking has evolved as the main technology for the verification of reactive systems like real-time systems, communication protocols, hardware circuits, and many others [1]. The specification is thereby given as a logical formula and the verification problem is viewed as the evaluation of the formula in a given model. Usually, the model is a transition system with finitely or infinitely many states. On the specification side, a lot of formalisms have been studied, and most of them can be translated to the μ-calculus. Global model checking procedures first compute those states of a transition system that satisfy a formula by means of fix point iteration [2]. In a second step, it is then checked whether this set is included in the set of initial states.

Model checking has been one of the most successful approaches to program verification during the last two decades. The size and complexity of applications which can be handled have increased rapidly through integration with symbolic techniques such as BDDs and through the use of SAT-solvers. These methods are designed to work on finite (but large) state spaces, and have been successfully used in industrial-sized projects, especially in the area of hardware verification. While the finite-state framework is well suited for reasoning about hardware circuits, it fails to deal with several essential aspects of software systems. A large amount of work has therefore been devoted to extending the capabilities of model checking in order to handle systems with infinite state spaces. Examples include timed automata, pushdown automata, models for dynamic memory architectures, communicating finite-state processes, multi-threaded systems, timed Petri nets, etc. In a parallel development, methods have been designed for the analysis of stochastic models The motivation is to capture the behaviors of systems with uncertainty such as programs with unreliable channels, randomized algorithms, and fault-tolerant systems. The underlying semantics for such systems is often defined by Markov chains.

B: Presburger Temporal Logic

In this section, we explain how Presburger arithmetic can be used for the representation and specification of infinite state systems. Presburger arithmetic is the first order theory of the natural numbers with addition as the basic operation. In contrast to the original definition, we interpret the logic over the integers rather than over the natural numbers. It is well-known that Presburger arithmetic is decidable [10,11,12,13].

Integer Kripke Structure: Given a finite set of variables V, an integer Kripke structure (IKS) is a transition system K = (S, T, R) where S is the possibly infinite set of states, T$\subseteq$S are the initial states, and R$\subseteq$S x S is the transition relation. Every state s $\in$ s is a variable assignment s : V ->Z. In addition, it is required that the set of initial states T and the transition relation R are definable in Presburger arithmetic. Moreover, R must he total.

According to the above definition, a state of an IKS is an assignment for the variables V. Such an assignment describes the current values of the system's variables. As the system proceeds with its execution, it changes some of the variables and therefore, we have a new assignment at the next point of time, Hence, the transition relation can be represented by a Presburger formula over the set of variables V U V' where V and V' are the current and the next state variables, respectively.

C: BMC of LTL^{pA} by Translation to Automata: BMC of Infinite State Systems

As the semantics of the temporal operators is the same for LTL and LTL^{pA}, the same recursion equations hold, i.e., we can still use Fp = p V XFq and Gq = p ^ XGp for unwinding the

temporal operators F and G. However, due to the presence of infinitely many states, we lose completeness for universal liveness and existential safety properties:

Theorem (Bounded Model Checking of LTL^{pA}): Given a Kripke structure K = (S, T, R), a state sεS, and a Presburger formula p ε PA, we have the following reductions:

1: K, so |= EFp holds iff some k, the Presburger formula Prefix([so,. . . : s k]) ^ Exists((s0,. . . ! sk], p) is satisfiable.

2: If there exists a number k, such that the Presburger formula Prefix([so,. . . ! s k]) ^ Forall([so) . . . , s k] , -p) is not satisfiable, then K, SO k^Fy holds. The converse need not necessarily hold.

3: If there exists a number k, such that the prefix([so,. . . , s k]) -> Exists([so:. . . , s k] , ~p) is not satisfiable, then K, s |= Gp does not hold. The converrse need not necessarily hold.

4: K, s|= k AGp does not hold iff for some k , the formula Prefix([so, . . . ; s k])-> For all([so,. , . s k]p,) is not suitable.

Module Incomplete AF:

input z : integer;
output *Tdy;*
while 2 # 0 do
if z > 0 then
else
end if:
pause
end while;
emit *rdy*
next(z) := *x* - 1
next(s) := *s* + 1
end loop
end module

Recently, several attempts have been made to consider systems which combine the above two features, i.e., systems which are infinite-state and which exhibit probabilistic behavior. For instance, the work in considers Probabilistic Lossy Channel Systems (PLCS): systems consisting of finite-state processes, which communicate through channels which are unbounded and unreliable in the sense that they can spontaneously lose messages. The motivation for these works is that, since we are dealing with unreliable communication, it is relevant to take into consideration the probability by which messages are lost inside the channels. Probabilistic Pushdown Automata (or Recursive State Machines) have been considered as natural models for probabilistic sequential programs with recursive procedures. Probabilistic extensions of Petri nets can be used as abstractions of distributed protocols with randomized components.

In global model checking, the syntax tree of a specification is traversed in a bottom-up manner, whereas in local model checking, a specification is evaluated top-down. For *finite state systems*, the verification problem is decidable, and for both approaches there are algorithms with essentially the same worst case asymptotic complexity. In practice, however, the runtimes of global and local model checking procedures may differ significantly. In particular, local model checking has the advantage that sub formulas can be checked by need, i.e., in the spirit of lazy evaluation. Moreover, only parts of the transition relation are required during the construction of a proof tree. In contrast, global model checking procedures usually require the construction of the complete transition relation to perform a breadth-first traversal of the state space. On the other hand, they benefit from efficient symbolic set representations like binary decision diagrams.

We concentrate on systems with a finite state control flow, but with data structures over infinite domains. Systems of this class appear in different forms, e.g., as extended finite state machines, and belong to the most powerful machine models, since they are equivalent to Turing machines. Moreover, they are frequently used in early phases of many design flows.

In general, the model checking problem is undecidable for infinite state systems, and hence, it may happen that the verification process does not terminate1. Clearly, termination of a verification procedure does not only depend on the formula and the transition system, but also on the verification procedure itself. To achieve termination, the user is often required to guide the verification process. Nevertheless, it may happen that one procedure is able to verify a specification automatically, whereas another one requires user interaction. As a result, the choice of the best procedure is not primarily a matter of efficiency as in the case of finite state systems, but rather a matter of termination.

5. CONCLUSION

We analyzed global and local model checking for the verification of infinite state systems. It turned out that under the premise of an automatic verification process, both approaches have their strengths and weaknesses. For some specifications, one approach may terminate while the other one does not, and vice versa. As a result, neither of them is clearly superior. However, even a simple combination where both approaches run in parallel allows one to check large classes of specifications. We also analyze an approach for bounded model checking of infinite state systems. As the base logic Presburger arithmetic been used to efficiently represent infinite sets. Moreover, use of a special technique *for* translating temporal logic formulas to w-automata can be done in verification of infinite state automata. This translation enables us to directly exploit safety and liveness properties contained in the specification. Liveness and safety properties are defined according to the temporal logic hierarchy. In the future, we plan to investigate how other types of specifications can be verified using bounded mode1 checking.

6. REFERENCES

[1] E. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT, London, England, 1999.
[2] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, Workshop on Logics of Programs, volume 131 of LNCS, pages 52–71, Yorktown Heights, New York, May 1981. Springer.
[3] J. Esparza. An automata-theoretic approach to software verification. In Developments in Language Theory, volume 2710 of LNCS, page 21. Springer, 2003.
[4] J. Bradfield. Verifying Temporal Properties of Systems. Progress in Theoretical Computer Science. Birkhäuser,
Boston, Basel, Berlin, 1992.
[5] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. In K. Larsen and A. Skou, editors, Workshop on Computer Aided Verification (CAV), July 1991.
[6] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using Presburger arithmetic. In O. Grumberg, editor, Conference on Computer Aided Verification (CAV), volume 1254 of LNCS, pages 400–411, Haifa, Israel, 1997. Springer
[7] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables. ACM Transactions on Programming Languages and Systems (TOPLAS), 21(4):747–789, 1999.
[8] T. Schüle and K. Schneider. Global vs. local model checking of infinite state systems. In GI/ITG/GMM Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, Kaiserslautern, Germany, February 2004.
[9] A. Meyer. Traces of term-automatic graphs. In MFCS, vol. 4708 of LNCS, pp. 489–500, 2007.
10] H. Enderton. A Marhenmticd Introduction to Lngic. Academic Press, New York, 1972.
[11] V. Ganesh, S. Berezin, and D. Dill. Deciding Presburger arithmetic by model checking and comparisons with other methods. In M. Aagaard and J. O'Leary, editors, Conference uii FomtaI Methods in Computer Aided Design (FMCAD), volume 2517 of LNCS, pages 171-186, Portland, USA, 2002. Springer.
[12] D. Oppen'. A ZzZp" upper bound on the complexity of Presburger arithmetic. Cornpurer and Sysfem Sciences, 16:323-332
[13] M. Presburger. Uher die Vollstandigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In F. Leja, editor, Sprawozdanie z I Kongreslr Matentagkdw Krajdw Sfowiariskich, pages 92-101, Warszawa, Sklad Glowny, 1929.

