The Role of Buchi's Automata in Computing Science

E. Allen Emerson

University of Texas at Austin

Automata on infinite objects (strings) were introduced in the early 1960's by Buchi [Bu60], motivated by issues in mathematical logic, viz., the decidability of his monadic second order theory of one successor (S1S). This logical theory was actually referred to by Buchi as the Sequential Calculus because of its potential utility for describing the behavior of sequential circuits. Somewhat later, Muller [Mu63] proposed that automata on infinite strings might themselves be applicable to describing the behavior of circuits, in particular, nonstabilizing asynchronous digital circuits. By the end of the 1960's, Rabin [Ra69] had shown how automata on infinite trees, a natural but very powerful generalization of Buchi's automata on strings, could be used to show decidability of a surprisingly rich class of logical theories including the basic monadic second order theory of multiple successors (SnS).

We emphasize that the above automata, while they accept input strings of infinite length ω (or input trees with paths of length ω), are still finite state devices. For example, a Buchi string automaton with acceptance condition specified by a set of states GREEN, accepts an input string iff there is a path through the automaton starting from its initial state leading through a GREEN state infinitely often. For this reason, testing nonemptiness of such an automaton (i.e., whether there exists some infinite input string the automaton accepts) is a decidable problem which amounts to checking if there is a path through the automaton leading to a cycle that includes a GREEN state.

In the 1970's there was relatively little interest in these automata. There was some theoretical work on automata with infinite state spaces such as pushdown tree automata. However, the decision problems usually became undecidable. Thus, while of some theoretical interest, it did not appear to have major impact on Computing Science.

However, over the past twelve years (1977-1989) there has been a significant resurgence of interest in finite automata on infinite strings and trees (henceforth referred to simply as automata), owing to their close relationship with nonterminating concurrent computer programs and formalisms for specifying and reasoning about such programs.

An ordinary sequential program has a transformational character. It starts executing in an initial state corresponding to a given set of input data values. It then executes for some finite number of steps, whereupon it terminates in a particular final state corresponding to a certain set of output data values. Its semantics can be described as a relation from initial states to final states. Most such programs are deterministic, so the relation is actually a function (or transformation). It is thus possible to specify correct behavior using precondition/postcondition pairs which assert that if the program starts executing in a state satisfying the precondition it will terminate in a final state satisfying the postcondition. Examples of such programs abound and include most data processing programs; for a trivial example consider the program which takes as input a postive integer and outputs the integer squared.

In contrast, a continuously operating concurrent program exhibits ongoing behavior that is ideally nonterminating. Examples of such programs include computer operating systems, network communication protocols, air traffic control systems, and nonstabilizing asynchronous digital circuits. For these programs, termination represents an abnormal condition while normal behavior entails executing forever. Thus, such programs are not naturally described using the initial state/final state semantics used for transformational programs. The semantics of these programs is instead described in terms of infinite computation sequences recording the states the program goes through as it executes. For a given initial state, there are in general infinitely many computation sequences beginning with that state, because the programs are usually highly nondeterministic. This reflects our modelling of the execution of the parallel program $P_1//...//P_k$ composed of the sequential programs $P_1,...,P_k$ in the usual way, which is to treat it as the nondeterministic interleaving of the atomic steps (individual indvisible instructions) of each sequential program P_i. A concurrent program currently in a particular initial state nondeterministically selects which sequential program to execute a step of next. After executing that step the program is in a new state from which it makes another choice of next step. And so on. Thus, the set of all possible computation sequences the program may follow starting from a given state can be seen to in general contain infinitely many sequences. In many cases, additional nondeterminism arises from the programs interaction with the external world (e.g., is a character input from the keyboard or not?). Finally, one additional requirement that we might wish to impose is that each computation sequence be "fair" in the sense that each sequential program P_i is executed infinitely often along the sequence.

In 1977 Pnueli [Pn77] proposed the use of Temporal Logic for reasoning about continuously operating concurrent programs. Temporal Logic is a type of modal logic that provides a formalism for describing how the truth values of assertions vary over time. While there are a variety of different systems of Temporal Logic, typical temporal operators or modalities include Fp ("sometime p") which is true now provided there is a future moment where p holds, and Gp ("always p") which is true now provided that p holds at all future moments. As Pnueli argued, Temporal Logic seems particularly well-suited to describing correct behavior of continuously operating concurrent programs. For example, to assert that two concurrently executing programs are not in their respective critical sections at the same time, one can write $\mathrm{G}(\neg inCS_1 \vee \neg inCS_2)$, where $inCS_i$, $i = 1, 2$, is an atomic proposition true exactly when process P_i is in its critical section. This mutual exclusion requirement is quite common in operating systems applications. Similarly, to assert that each request for service is eventually granted, one can write $\mathrm{G}(req \Rightarrow \mathrm{F}grant)$. This type of liveness requirement is typical of what might be expected of a "resource controller" which monitors access to a shared resource.

Following the publication of Pnueli's 1977 paper, Temporal Logic has become an active area of research interest. There is now a consensus among a large body of practicioners as well as theoreticians that the various forms of Temporal Logic comprise a very promising approach for specifying and reasoning about continuously operating concurrent programs. Concommittant with the rising interest in Temporal Logic, there has also been a resurgence of interest in automata on infinite objects. This interests arises from the intimate connection between the two formalisms. Today we find finite state automata on infinite objects used or proposed for use in virtually every capacity for reasoning about ongoing concurrent programs. Automata provide not only important technical machinery, but also a convenient and uniform conceptual framework.

One ambitious application is the mechanical synthesis of a concurrent program from a specification of its correct behavior formulated in a system of propositional Temporal Logic (cf. [EC82]). These propositional logics are surprisingly expressive, and are suffient for a large variety of applications. Synthesis is possible because these logics are decidable, since they possess the small model property: if a formula is satisfiable, then it has a finite model of bounded size. This finite model can be viewed as the global state graph of a concurrent program meeting the specification, from which its constituent sequential subprograms can be obtained. An unsatisfiable formula, on the other hand, indicates that the specification is inconsistent and must be reformulated.

For simple logics, the decision procedure can be obtained using various tableau constructions, similar to those used in classical modal logic. A tableau is a graph, constructed in a systematic way from the syntax of the formula, that essentially encodes all potential models of a formula. A uniform method for developing decision procedures can be obtained by observing that a tableau defines the transition diagram of an automaton with the Buchi acceptance condition. The formula is satisfiable iff the Buchi automaton accepts some input. So the decision procedure is obtained by reduction to the nonemptiness problem for Buchi automata.

For richer logics, the tableau construction is no longer applicable. However, it is still possible to reduce satisfiability to testing nonemptiness of a finite state automaton on infinite strings or trees, by appealing to such delicate automata-theoretic constructions as McNaughton's or Safra's algorithm for determinizing a Buchi automaton on infinite strings (cf. [McN66], [Sa89]) and by allowing a more complicated acceptance condition (essentially boolean combinations of Buchi's acceptance condition). An alternative way to get a decision procedure is by encoding the logics in ,e.g., SnS. Unfortunately, this yields a decision procedure of nonelementary complexity i.e., not bounded by the composition of a fixed number of exponential functions, $exp(n) = c^n$, for some constant $c > 1$ (cf. [Me74]). The only known way to get to elementary time decision procedures is by directly reducing from the logic to the automaton. In fact, it was this application - to improve the complexity of decision procedures for Temporal Logics - which was the primary motivation in Computing Science for resuming the study of automata on infinite objects (cf. [St82]).

If we restrict our attention to finite state concurrent programs, the problem of determining whether a given program meets a given specification expressed in Temporal Logic (or a related formalism) is always decidable, and in many cases quite practical and useful. It amounts to a brute force graph reachability analysis. Very efficient algorithms can be developed, which run in time linear in the size of the program and in many cases linear or polynomial in the size of the specification, to check whether the program defines a model of the specification formula. This "model checking" approach to mechanical verification of finite state concurrent programs is still of wide applicability since many solution to concurrent programming problems in the literature are finite state, as are many network communication protocols (cf. [CES86], [CG87]).

Automata have also been proposed as a useful and general specification language themselves. Some would argue that it is more convenient and comprehensible for humans to write specifications as automata rather than in Temporal Logic. In fact, automata provide strictly more expressive power than (ordinary) Temporal Logic. The property G_2p, meaning that at all even moments p holds, is easily described by an automaton, but not by any Temporal Logic formula. Moreover, automata provide a sort of "yardstick" for measuring the expressive power of various system of Temporal Logic and related formalisms for describing sequences. Ordinary Temporal Logic is equivalent to counter-free automata, star-free ω -regular expressions, and the First Order Language of Linear Order. Buchi automata, on the other hand, provide us with the full power of ω -regular expressions, the Second Order Language of Linear Order, and S1S.

Additionally, automata are useful as a representation of concurrent programs, with the global state graph corresponding to the automaton's transition diagram in an obvious way, and any needed fairness requirements captured the automaton's acceptance condition. Since both programs and specifications can be naturally represented using automata, it becomes possible to formulate and compare different approaches to mechanical program synthesis, automatic program verification, and model checking within the uniform framework of automata (cf. [VW86]).

To sum up, we recall the twin origins of automata on infinite objects. First, Buchi introduced them for an essentially theoretical purpose: to give a decision procedure for testing the validity of sentences of S1S. There was also a more practical motivation: describing the behavior of digital circuits. Today, Buchi's and related automata are studied from both a theoretical and a practical viewpoint. But now, the theoretical and practical have blended together to a large extent. A noteworthy point is that the central technical use of automata by Buchi - to provide a decision procedure for a logical theory by reduction to the emptiness problem for the automata - remains today the main use of such automata in connection with logical theories, such as Temporal Logic, for reasoning about program correctness.

References

[Bu60] Buchi, J. R., On a Decision Method in Restricted Second Order Arithmentic, Proc. 1960 International Congerss on Logic, Methodology, and Philosophy of Science, (1962), pp. 1-11.

[CES86] Clarke, E. M., Emerson, E. A., and Sistla, A. P., "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", ACM Transactions on Prog. Lang. and Sys., v. 8, no. 2, p. 244-263, April 1986.

[CG87] Clarke, E. M. and Grumberg, O. Research on Automatic Verification of Finite State Concurrent Systems, Annual Reviews in Computer Science, 2, pp. 269-290, 1987

[EC82] E.A. Emerson, E.M. Clarke, "Using Branching Time Logic to Synthesize Synchronization Skeletons", *Science of Computer Programming*, 2, pp. 241-266, 1982.

[Mc66] R. McNaughton, "Testing and Generating Infinite Sequences by a finite Automaton", *Information and Control*, 9, 521-530, 1966.

[Me74] Meyer, A. R., "Weak Monadic Second Order Theory of Successor is not Elementary Recursive'," Boston Logic Colloquium, Springer-Verlag Lect. Notes. in Math., no. 453, 1974..

[Mu63] Muller, D. E., "Infinite Sequences and Finite Machines," Proc. 4th Annual IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 3-16, 1963.

[Pn77] A. Pnueli, "The Temporal Logic of Programs", *19th IEEE Symp. on Found. of Comp. Science*, pp. 1-14, 1977.

[Ra69] M.O. Rabin, "Decidability of Second Order Theories and Automata on Infinite Trees", *Trans. AMS*, 141(1969), pp. 1-35.

[Sa88] S. Safra, "On Complexity of ω-automata", *29th IEEE Symposium on Found. of Comp. Science*, pp. 319-327, 1988.

[St82] R.S. Streett, " Propositional Dynamic Logic of Looping and Converse is Elementary Decidable", *Information and Control* v. 54, pp. 121-141, 1982.

[VW86] Vardi, M. Y., and Wolper, P. L., "An Automata-Theoretic Approach to Automatic Program Verification," Proc. IEEE Symp. on Logic in Comp. Science, pp. 332-346, 1986.