17

AUTOMATA-THEORETIC TECHNIQUES FOR TEMPORAL REASONING

Moshe Y. Vardi

1 INTRODUCTION

This chapter describes an automata-theoretic approach to temporal reasoning. The basic idea underlying this approach is that for any temporal formula we can construct a finite-state automaton that accepts the computations that satisfy the formula. For linear temporal logics the automaton runs on infinite words while for branching temporal logics the automaton runs on infinite trees. The simple combinatorial structures that emerge from the automata-theoretic approach decouple the logical and algorithmic components of temporal reasoning and yield clear and asymptotically optimal algorithms.

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying behavior concurrent programs and for verifying that such programs meet their specifications [47, 57]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [15, 44, 60, 81].

(A state of a program is a complete description of its status, including the assignment of values to variables, the value of the program counter, which points to the instruction currently being executed, and the like. Finite-state programs have finitely many possible states, which means that the variables range over finite domains and recursion, if allowed, is bounded in depth.) This derives its significance from the fact that many synchronization, coordination and communication protocols can be modeled as finite-state programs [46, 63]. Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of Boolean atomic propositions. This means that a finite-state program can be viewed as a finite *propositional Kripke structure* and that its properties can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, is a model of (satisfies) the propositional temporal logic formula that specifies that behavior. Hence the name *model checking* for the verification methods derived from this viewpoint. An extensive survey of model checking can be found in [16, 37]. Model checking is an *algorithmic* approach to program verification. It is different from the *deductive approach*, in which in which a person, perhaps aided by computers, use deductive techniques to prove that a program satisfy its specification [48].

We distinguish between two types of temporal logics: linear and branching (see [43] and general discussion of temporal structures in Chapter 11 of this handbook). In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. (For an extensive discussion of various temporal logics, see [22].) For both types of temporal logics, a close and fruitful connection with the theory of automata on infinite structures has been developed. The basic idea is to associate with each temporal logic formula a finite automaton on infinite structures that accepts the computations that satisfy the formula. For linear temporal logic the structures are infinite words [66, 45, 68, 83], while for branching temporal logic the structures are infinite trees [30, 69, 21, 26, 82]. This enables the reduction of temporal logic decision problems, such as satisfiability and model checking, to known automata-theoretic problems, such as nonemptiness, yielding clean and asymptotically optimal algorithms. This reduction is the subject matter of this chapter. (See also Chapter 11 for a general discussion of temporal reasoning and Chapters 4 and 7 for discussions of modal reasoning.)

Initially, the translations in the literature from temporal logic formulas to automata used *nondeterministic* automata (cf. [34, 82, 83]). These translations have two disadvantages. First, the translation itself is rather nontrivial; For example, in [82, 83] the translations go through a series of ad-hoc intermediate representations in an attempt to simplify the translation. Second, for both linear and branching temporal logics there is an exponential blow-up involved in going from formulas to automata. This suggests that any algorithm that uses these translations as one of its steps is going to be an exponential-time algorithm. Thus, the automata-theoretic approach did not seem to be applicable to branching-time model checking, which in many cases can be done in linear execution time [15, 17, 60].

In the mid 1990s, it was shown that if one uses *alternating* automata rather than *nondeterministic* automata, then these problems can be addressed [42, 74]. Alternating automata generalize the standard notion of nondeterministic automata by allowing several successor states to go down along the same word or the same branch of the tree. In