Studies in Systems, Decision and Control 89

Calin Belta
Boyan Yordanov
Ebru Aydin Gol

Formal Methods for Discrete-Time Dynamical Systems

Springer

Chapter 3
Model Checking

In this chapter, we introduce model checking, which is the most basic analysis problem in formal verification. As the focus of the book is on LTL, we restrict our attention to LTL specifications. Since we focus on analysis, we consider transition systems with no inputs. Informally, the LTL model checking problem consists of determining whether the language originating at a state of a finite transition system satisfies an LTL formula over its set of observations. The algorithms presented in this chapter will be extended in subsequent chapters to solve more difficult problems, such as finding the largest satisfying region for finite transition systems and infinite transition systems embedding discrete-time dynamical systems.

Given a finite transition system with no inputs $T = (X, \delta, O, o)$ and an LTL formula ϕ over O, checking whether all output words of T from a given subset of its states satisfy ϕ is called *LTL model checking*. Since we only use LTL in this book, LTL model checking is simply referred to as *model checking*.

Definition 3.1 (*LTL satisfaction*) Transition system T satisfies formula ϕ from a given region $X_r \subseteq X$, written as $T(X_r) \vDash \phi$, if and only if all words $w_O \in \mathscr{L}_T(X_r)$ produced by trajectories of T originating in X_r satisfy ϕ. Formally,

$$T(X_r) \vDash \phi \Leftrightarrow \forall w_O \in \mathscr{L}_T(X_r), w_O \vDash \phi, \tag{3.1}$$

which can be expressed equivalently as

$$T(X_r) \vDash \phi \Leftrightarrow \mathscr{L}_T(X_r) \subseteq \mathscr{L}_\phi. \tag{3.2}$$

Problem 3.1 (*Model checking)* Given a finite transition system T, a region $X_r \subseteq X$ of T, and an LTL formula ϕ over its set of observations O, determine if $T(X_r) \vDash \phi$.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_3

Model checking provides a technique for automatically determining if T satisfies ϕ when T is finite. An off-the-shelf model checker (see Sect. 3.1 for a review of such tools) takes as input a finite transition system T and a formula ϕ and returns a positive verification result only if all words produced by runs of the system originating in X_r satisfy the formula (i.e., if Eq. (3.2) is satisfied). Otherwise, the model-checker returns a non-satisfying run of T as a *counterexample* certifying the violation of the formula.

Deciding if the inclusion from Eq. (3.2) holds can be computationally challenging. Therefore, model checking algorithms often rely on computation using the negation of the LTL formula and the following equivalence:

$$\mathscr{L}_T(X_r) \subseteq \mathscr{L}_\phi \Leftrightarrow \mathscr{L}_T(X_r) \cap \mathscr{L}_{\neg\phi} = \emptyset \tag{3.3}$$

In other words, through Eq. (3.3) the model checking problem reduces to finding a word $w_O \in \mathscr{L}_T(X_r) \cap \mathscr{L}_{\neg\phi}$ and, if no such word exists, it is guaranteed that the system satisfies the formula from region X_r. To find a run w_X of T, which produces a word w_O such that $w_O \models \neg\phi$ (i.e., w_X is a counterexample in T), automata-theoretic model checking algorithms rely on the construction of a *product automaton* between the transition system and the Büchi automaton corresponding to the formula $\neg\phi$.

Definition 3.2 (*Uncontrolled Büchi Product Automaton*)[1] The *uncontrolled Büchi product automaton* $P = T \otimes B$ of a finite uncontrolled transition system $T = (X, \delta, O, o)$ and a Büchi automaton $B = (S, S_0, O, \delta_B, F)$ is defined as $P = (S_P, S_{0P}, \delta_P, F_P)$, where

- $S_P = X \times S$ is the set of states,
- $S_{0P} = X \times S_0$ is the set of initial states,
- δ_P is the transition function where, for a state $(x, s) \in S_P$, we have $\delta_P((x, s)) = \{(x', s') \in S_P \mid x' \in \delta(x) \text{ and } s' \in \delta_B(s, o(x))\}$,
- $F_P = X \times F$ is the set of accepting states.

This product automaton is a nondeterministic Büchi automaton with input alphabet containing only one element, which is therefore omitted. In addition, P is finite since both T and B are finite. A state $(x, s) \in S_P$ of P can be projected into a state $x \in X$ of T. We denote this projection by $\alpha : S_P \to X$, where $\alpha(x, s) = x$. A run $w_{S_P} = (x_1, s_1)(x_2, s_2)\ldots$ that is accepted by P can be projected into a run $w_X = \alpha(w_{S_P}) = \alpha(x_1, s_1)\alpha(x_2, s_2)\ldots = x_1 x_2 \ldots$ of T, such that $o(x_1)o(x_2)\ldots$ is accepted by B. If the product automaton $P_\phi = T \otimes B_\phi$ of a transition system T and the Büchi automaton B_ϕ is constructed (where B_ϕ accepts the language $\mathscr{L}_\phi$ for some

[1] There will be several versions of product automata throughout the book, e.g., controlled Büchi product automaton, controlled Rabin product automaton, controlled finite state product automaton. We will have formal definitions for all these product automata. However, whenever possible, we will call them simply product automata in the text, as their meaning will be clear from the context.

LTL formula ϕ as described in Definition 2.5), a run accepted by P_ϕ is projected into a run of T that produces a word w_O over O accepted by B_ϕ (i.e., w_O satisfies ϕ).

Note that a product automaton $P = T \otimes R$ analogous to the one from Definition 3.2 can be constructed with a Rabin automaton R (see Definition 5.3). In that case, the acceptance condition is inherited from the Rabin acceptance condition (see Definition 2.8).

In model checking a finite transition system T from region X_r, we construct the product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$ and restrict the set of initial states of $P_{\neg\phi}$ to $S_{0P_{\neg\phi}} = X_r \times S_0$. A run $w_{S_{P_{\neg\phi}}} = (x_1, s_1)(x_2, s_2)\ldots$ that is accepted by $P_{\neg\phi}$ can be projected into a run $w_X = x_1 x_2 \ldots$ of T. If w_O is the word produced by w_X, then we have $w_O \vDash \neg\phi$ or, equivalently, $w_O \nvDash \phi$. Therefore, w_X is a counterexample proving that $T(X_r) \nvDash \phi$ and the model-checking problem reduces to finding the run $w_{S_{P_{\neg\phi}}}$ that is accepted by $P_{\neg\phi}$.

Finding an accepting run in $P_{\neg\phi}$ can be accomplished efficiently by treating it as a directed graph and decomposing it into *maximal strongly connected components* (SCCs)—maximal regions of $P_{\neg\phi}$ for which each state is reachable from every other one. Considering states of $P_{\neg\phi}$ that belong to the same SCC as equivalent induces a *quotient system*, where each state is a SCC (an equivalence class of states from $S_{P_{\neg\phi}}$) and a transition between states (SCCs) C_1 and C_2 exists if and only if there was a transition between some states $s_1 \in C_1$ and $s_2 \in C_2$ in $P_{\neg\phi}$. The corresponding SCC quotient is a directed acyclic graph. Initial and accepting states of the quotient system are SCCs that contain at least one initial or accepting state from $S_{P_{\neg\phi}}$, respectively. The existence of an accepting state (SCC) in the quotient system that is reachable from an initial state (SCC) is equivalent with the existence of a run that is accepted by $P_{\neg\phi}$. The former can be checked efficiently with basic graph algorithms. This computation provides the core of an LTL model checking algorithm and is summarized as Algorithm 2. The overall complexity of LTL model checking is $\mathcal{O}(|X| \cdot 2^{|\phi|})$.

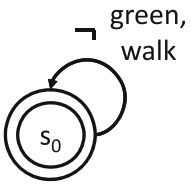
Algorithm 2 MODEL- CHECK(T, X_r, ϕ): Determine if finite transition system T satisfies LTL formula ϕ from region X_r

1: Translate the negation of the formula $\neg\phi$ to Büchi automaton $B_{\neg\phi}$
2: Construct the product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$
3: Restrict the initial states of $P_{\neg\phi}$ to $S_{0P_{\neg\phi}} = X_r \times S_0$
4: Decompose $P_{\neg\phi}$ into SCCs
5: **if** there exists a run from an SCC containing an initial state (a state in $S_{0P_{\neg\phi}}$) to an SCC containing an accepting state (a state from $F_{P_{\neg\phi}}$) **then**
6: return *false* and the corresponding counterexample $(T(X_r) \nvDash \phi)$
7: **else**
8: return *true* $(T(X_r) \vDash \phi)$
9: **end if**

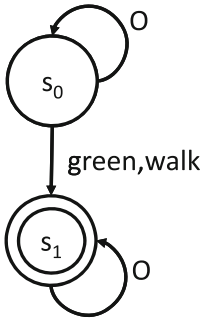
Example 3.1 In order to illustrate the model checking procedure, we consider the pedestrian crossing traffic light system introduced in Example 1.6. For this system, the crucial safety property requires that pedestrians and cars are not allowed through the intersection at the same time—a mutual exclusion specification. Formally, we can express this requirement with the LTL formula $\phi = \square\neg$ "green, walk" (shown as a Büchi automaton B_ϕ in Fig. 3.1a), specifying that the system should never visit a state where the car traffic light displays the "green" signal, while the pedestrian light displays the "walk" signal. To guarantee that the crossing operates safely, we are interested in model checking system T from Example 1.6 against specification ϕ from region $X_r = \{(x_1^c, x_1^p) \ldots (x_3^c, x_2^p)\}$, which includes all states of the system.

For transition system T shown in Fig. 1.6c it is clear that, regardless of the initial state, every trajectory of the system visits state (x_1^c, x_1^p), where "green, walk" is satisfied, and therefore specification ϕ is violated. To illustrate the process, in the following we go through the individual steps of the model checking procedure summarized in Algorithm 2.

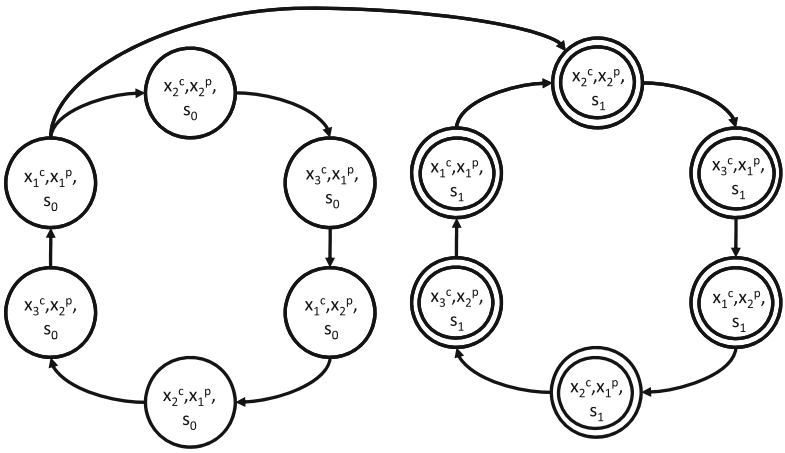
i. The negation $\neg\phi$ of specification $\phi = \square\neg$ "green, walk" (shown as a Büchi automaton in Fig. 3.1a) is translated into the Büchi automaton $B_{\neg\phi}$ shown in Fig. 3.1b (s_0 is the initial state of $B_{\neg\phi}$). Transitions of $B_{\neg\phi}$ are enabled by observations from the set $O =$ {"yellow, don't walk", "red, walk", "green, don't walk", "yellow, walk", "red, don't walk", "green, walk" }.
ii. The product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$ shown in Fig. 3.1c is constructed. Since all states are initial in T and only state s_0 is initial in $B_{\neg\phi}$, all states $(x_1^c, x_1^p, s_0) \ldots (x_3^c, x_2^p, s_0)$ are initial in $P_{\neg\phi}$. Similarly, since state s_1 is accepting in $B_{\neg\phi}$, all states $(x_1^c, x_1^p, s_1) \ldots (x_3^c, x_2^p, s_1)$ are accepting in $P_{\neg\phi}$.
iii. The product automaton $P_{\neg\phi}$ is decomposed into strongly connected components (SCCs) and the corresponding quotient system is shown in Fig. 3.1d, where state S_0 is initial and state S_1 is accepting.
iv. Since a run from state S_0 to state S_1 exists in the SCC quotient of $P_{\neg\phi}$, then there exists a trajectory of T that violates the specification. In other words, there exists a sequence of signals displayed at the intersection that eventually leads to the unsafe "green, walk" signal being displayed. In fact, for this particular system, all trajectories eventually lead to such unsafe behavior and can therefore serve as counterexamples.



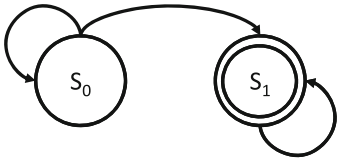
(a) Büchi automaton of the specification $\phi = \Box\neg$ "green, walk" (B_ϕ)



(b) Büchi automaton of the negation of the specification $\neg\phi = \Diamond$ "green, walk" ($B_{\neg\phi}$)



(c) Product automaton $P = T \otimes B_{\neg\phi}$



(d) SCC quotient graph of P

Fig. 3.1 Automata involved in the model checking problem described in Example 3.1 for the pedestrian intersection traffic light system. For the Büchi automaton shown in (**a**), the transition labeled by ¬ "green, walk" signifies that there exists a transition enabled by each observation from the set O, except for one enabled by observation "green, walk". For the Büchi automaton shown in (**b**), a transition labeled by the set of observations O signifies that there exists a transition enabled by each observation from O. See Example 3.1 for additional details

3.1 Notes

An in-depth discussion on model checking can be found in [45], where model checking algorithms for branching-time logics such as CTL are also included. For a more recent and comprehensive treatment, the reader is referred to [15], which also includes probabilistic verification, in which a Markov Chain is checked against a probabilistic CTL or LTL formula. Model checking algorithms based on nested depth-first search [48] have been developed as more memory-efficient alternatives to the SCC-based approach discussed in this chapter. Popular off-the-shelf implementations of model checking algorithms include tools such as SPIN [89], NuSMV [43], PRISM [114], and DiVinE [18].

As the complexity of model checking algorithms increases very fast with the sizes of systems, various symbolic representations have been developed to describe and manipulate finite transition systems. The most popular symbolic approach is based on binary decision diagrams (BDDs) [129]. The main limitation of the BDD-based representation is that its size depends on variable ordering and manual adjustments are usually necessary. Methods based on encodings such as Boolean satisfiability (SAT) problems, which take advantage of modern SAT solvers, were shown to perform well on large systems despite theoretical hardness results [31]. To deal with the difficulty of framing model checking problems as Boolean satisfiability problems, encodings based on Satisfiability Modulo Theories (SMT) have also been proposed [10].

In this book, model checking is performed against finite models or finite (simulation or bisimulation) abstractions of infinite models. There exist, however, model checking algorithms specifically adapted to apply directly to some classes of infinite systems, such as timed automata [7], pushdown automata [34, 58], or, more generally, well structured systems [2]. Even though the abstraction is not explicitly constructed in these methods, they are based on notions of decidability and computability similar to the ones used in this book.