Studies in Systems, Decision and Control 89

Calin Belta
Boyan Yordanov
Ebru Aydin Gol

Formal Methods for Discrete-Time Dynamical Systems

Springer

Example 2.4 Consider an agent moving in the planar environment from Fig. 2.1. The specification is to visit regions $\mathbf{X}_2$ or $\mathbf{X}_9$ and then the target region $\mathbf{X}_7$, while avoiding $\mathbf{X}_{11}$ and $\mathbf{X}_{12}$, and staying inside $\mathbf{X} = [-10, 2]^2$ until the target region is reached. This specification translates to the following scLTL formula:

$$\phi = ((\neg \mathbf{X}_{11} \ \wedge \ \neg \mathbf{X}_{12} \ \wedge \ \neg Out) \ U \ \mathbf{X}_7) \wedge (\neg \mathbf{X}_7 \ U \ (\mathbf{X}_2 \ \vee \ \mathbf{X}_9)), \qquad (2.3)$$

where $Out = \mathbb{R}^2 \setminus \mathbf{X}$.

2.2 Automata

We will use automata that accept languages satisfying LTL and scLTL formulas over the set of observations O. There is, therefore, no coincidence that the input alphabets of the automata defined below is O.

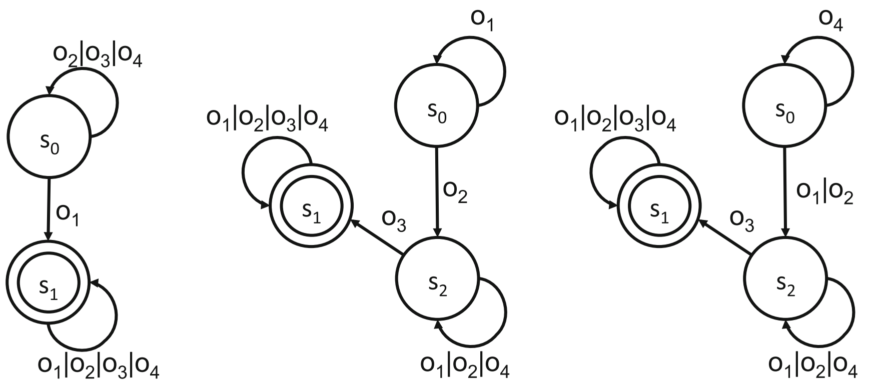
Definition 2.4 (*Finite state automaton*) A finite state automaton (FSA) is a tuple $A = (S, s_0, O, \delta, F)$, where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- O is the input alphabet,
- $\delta : S \times O \rightarrow S$ is a transition function, and
- $F \subseteq S$ is the set of accepting (final) states.

The semantics of a finite state automaton are defined over finite input words in O^*. A run of A over a word $w_O = w_O(1)w_O(2), \ldots, w_O(n) \in O^*$ is a sequence $w_S = w_S(1)w_S(2), \ldots, w_S(n+1) \in S^*$ where $w_S(1) = s_0$ and $w_S(k+1) = \delta(w_S(k), w_O(k))$ for all $k = 1, 2, \ldots, n$. The word w_O is accepted by A if and only if the corresponding run ends in a final automaton state, i.e., $w_S(n+1) \in F$. The language accepted by A is the set of all words accepted by A, and is denoted by $\mathscr{L}_A$.

A finite state automaton with a non-deterministic transition function, i.e., $\delta \colon S \times O \rightarrow 2^S$, and a set of initial states $S_0 \subseteq S$ instead of the singleton s_0 is called a non-deterministic finite state automaton (NFA). Every NFA can be translated to an equivalent FSA. For this reason, we only consider deterministic finite state automata in this book.

An scLTL formula ϕ over a set O can always be translated into an FSA A_ϕ with input alphabet O with $\mathscr{O}(2^{2^{|\phi|}})$ states ($|\phi|$ denotes the length of ϕ, which is defined as the total number of occurrences of observations and operators) that accepts all and only good prefixes of ϕ (i.e., $\mathscr{L}_{A_\phi} = \mathscr{L}_{pref,\phi}$). Some notes on available tools for this translation are given in Sect. 2.3 at the end of the chapter.



(a) $\phi_1 = \Diamond o_1$ (b) $\phi_2 = \Diamond o_3 \wedge (o_1 U o_2)$ (c) $\phi_3 = (\neg o_3 U (o_1 \vee o_2)) \wedge \Diamond o_3$

Fig. 2.2 Graphical representation of the finite state automata for some scLTL formulas over the set of observations $O = \{o_1, o_2, o_3, o_4\}$. For all automata, s_0 is the initial state and the final state is indicated by a *double circle*. For simplicity of the representation, if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown

Example 2.5 The finite state automata that accept the good prefixes of scLTL formulas $\phi_1 = \Diamond o_1$, $\phi_2 = \Diamond o_3 \wedge (o_1 U o_2)$, and $\phi_3 = (\neg o_3 U (o_1 \vee o_2)) \wedge \Diamond o_3$ over the set of observations $O = \{o_1, o_2, o_3, o_4\}$ are shown in Fig. 2.2.

Definition 2.5 (*Büchi automaton*) A (nondeterministic) Büchi automaton is a tuple $B = (S, S_0, O, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,
- $\delta : S \times O \to 2^S$ is a nondeterministic transition function, and
- $F \subseteq S$ is the set of accepting (final) states.

A Büchi automaton is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton for all $s \in S$ and $o \in O$. The semantics of a Büchi automaton are defined over infinite input words in O^ω. A run of B over a word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$ is a sequence $w_S = w_S(1)w_S(2)w_S(3)\ldots \in S^\omega$ where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 2.6 (*Büchi acceptance*) Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. An input word w_O is accepted by B if and only if there exists at least one run w_S over w_O that visits F infinitely often, i.e., $\inf(w_S) \cap F \neq \emptyset$.

We denote by $\mathscr{L}_B$ the language accepted by B, i.e., the set of all words accepted by B. An LTL formula ϕ over a set O can always be translated into a Büchi automaton

B_ϕ with input alphabet O and $\mathscr{O}(|\phi| \cdot 2^{|\phi|})$ states ($|\phi|$ denotes the length of ϕ, which is defined as the total number of occurrences of observations and operators) that accepts all and only words satisfying ϕ (i.e., $\mathscr{L}_{B_\phi} = \mathscr{L}_\phi$). This translation can be performed using efficient, off-the-shelf software tools, which are reviewed at the end of the chapter in Sect. 2.3.

Note that, in general, a nondeterministic Büchi automaton is obtained by translating an LTL formula. While certain Büchi automata can be determinized, a sound and complete procedure for determinizing general Büchi automata does not exist and, in fact, there exist LTL formulas which cannot be converted to deterministic Büchi automata.

Example 2.6 Examples of Büchi automata for some commonly encountered LTL formulas are shown in Fig. 2.3. Even when a nondeterministic Büchi automaton was obtained through the translation with LTL2BA tool, the automaton was simplified and determinized by hand whenever possible (e.g., for formulas ϕ_1, ϕ_4 and ϕ_5). Even so, some of the automata, such as the ones obtained for LTL formulas ϕ_3, ϕ_6 and ϕ_7 cannot be determinized. In fact, it is known that formulas of the type $\Diamond\Box\phi$ cannot be converted to a deterministic Büchi automaton. For example, the Büchi automaton for LTL formula ϕ_3 in Fig. 2.3c can be naively converted into a deterministic automaton by removing o_1 from the self transition at s_0. However, then word $o_1 o_2 (o_1)^\omega$, which is obviously satisfying, would not be accepted. While, in general, deterministic Büchi automata can be obtained for a class of LTL formulas through alternative approaches other than simply converting non-deterministic to deterministic Büchi automata, no such automaton exists for ϕ_3. To understand why formulas ϕ_6 and ϕ_7 result in nondeterministic Büchi automata, we can rewrite them as

$$\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2 = \Box\Diamond o_1 \wedge \Diamond\Box\neg o_2 \tag{2.4}$$

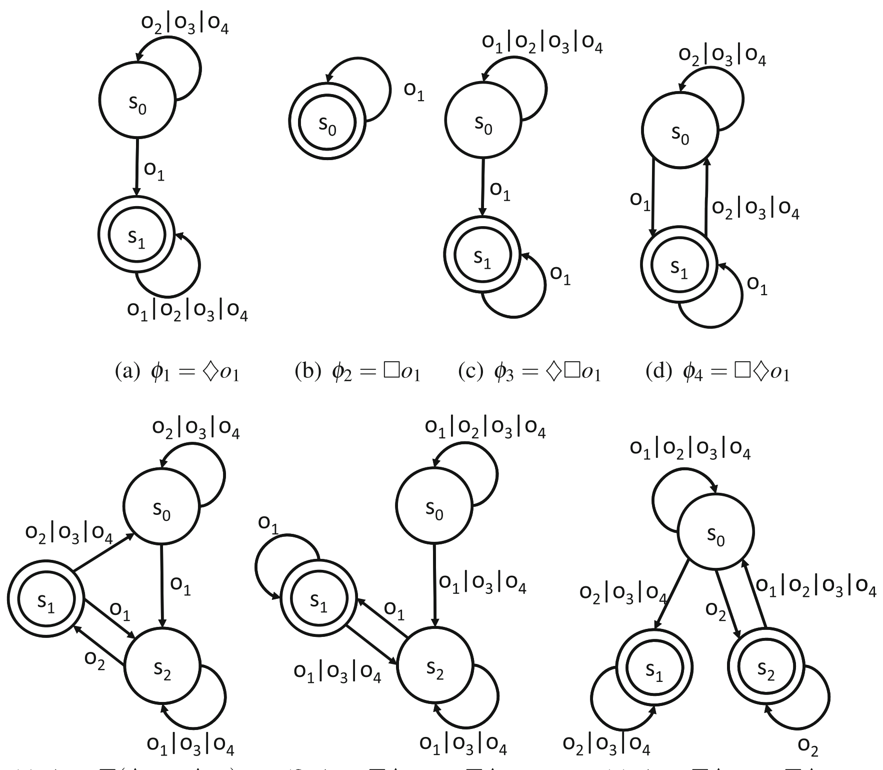
$$\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2 = (\Box\Diamond o_1 \wedge \Box\Diamond o_2) \vee \neg\Box\Diamond o_1 = \tag{2.5}$$

$$= (\Box\Diamond o_1 \wedge \Box\Diamond o_2) \vee \Diamond\Box\neg o_1 \tag{2.6}$$

to reveal that both contain a $\Diamond\Box$ sub-formula.

Definition 2.7 (*Rabin automaton*) A (nondeterministic) Rabin automaton is a tuple $R = (S, S_0, O, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,



(a) $\phi_1 = \Diamond o_1$ (b) $\phi_2 = \Box o_1$ (c) $\phi_3 = \Diamond\Box o_1$ (d) $\phi_4 = \Box\Diamond o_1$

(e) $\phi_5 = \Box(\Diamond o_1 \wedge \Diamond o_2)$ (f) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$ (g) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 2.3 Graphical representation of the Büchi automata for some commonly used LTL formulas over the set of observations $O = \{o_1, \ldots, o_4\}$. For all automata, s_0 is the initial state and the final states are indicated by *double circles*. As in Fig. 2.2, for simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 2.6

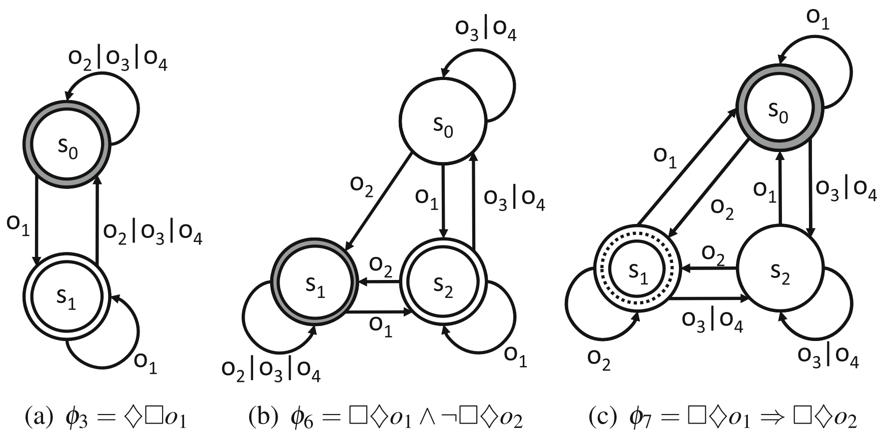
- $\delta : S \times O \rightarrow 2^S$ is a transition map, and
- $F = \{(G_1, B_1), \ldots, (G_n, B_n)\}$, where $G_i, B_i \subseteq S$, $i = 1, 2, \ldots, n$ is the acceptance condition.

A Rabin automaton R is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton, for all $s \in S$ and $o \in O$. The semantics of a Rabin automaton are defined over infinite input words in O^ω. A run of R over a word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$ is a sequence $w_S = w_S(1)w_S(2)w_S(3)\ldots \in S^\omega$, where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 2.8 (*Rabin acceptance*) Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. A run w_S is accepted by R if $\inf(w_S) \cap G_i \neq \emptyset \wedge \inf(w_S) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. An input word w_O is accepted by a Rabin automaton R if some run over w_O is accepted by R.

We denote by $\mathscr{L}_R$ the language accepted by R, i.e., the set of all words accepted by R. Given an LTL formula ϕ, one can build a deterministic Rabin automaton R with input alphabet O, $2^{2^{\mathcal{O}(|\phi|\cdot\log|\phi|)}}$ states, and $2^{\mathcal{O}(|\phi|)}$ pairs in its acceptance condition, such that $\mathscr{L}_R = \mathscr{L}_\phi$. The translation can be done using off-the-shelf software tools reviewed in Sect. 2.3. Note that a Büchi automaton B is a Rabin automaton R with one pair in its acceptance condition $F_R = \{(G, B)\}$ where $G = F_B$ and $B = \emptyset$.

Example 2.7 Even though LTL formulas ϕ_3, ϕ_6 and ϕ_7 could only be translated into nondeterministic Büchi automata in Example 2.6, we can translate them instead into the deterministic Rabin automata shown in Fig. 2.4. The Rabin automata for formulas ϕ_3 and ϕ_6 contain only a single pair in their acceptance conditions, while the one for ϕ_7 contains two pairs.



o2|o3|o4
s0
o1
o2|o3|o4
s1
o1
(a) $\phi_3 = \Diamond\Box o_1$
o3|o4
s0
o2
o1
o3|o4
s1
o2
s2
o1
o2|o3|o4
o1
(b) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$
o1
s0
o1
o2
o1
o3|o4
s1
o2
s2
o3|o4
o2
o3|o4
(c) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 2.4 Graphical representation of the Rabin automata for the LTL formulas from Example 2.6 resulting in nondeterministic Büchi automata. For each automaton, s_0 is the initial state. For the automata accepting formulas ϕ_3 and ϕ_6 the acceptance condition F is defined by one pair of singletons (G, B) where $G = \{s_1\}$, $B = \{s_0\}$ in (**a**) and $G = \{s_2\}$, $B = \{s_1\}$ in (**b**) (good and bad states are denoted by *unshaded* or *shaded double circles*, respectively). For the automaton accepting ϕ_7 the acceptance condition includes two pairs where $G_1 = \{s_1, s_2\}$, $B_1 = \{s_0\}$ and $G_2 = \{s_1\}$, $B_2 = \emptyset$ (the single bad state is denoted by a *shaded circle* and the good state that is common for both pairs of the acceptance conditions is denoted by a *solid* and *dashed circle* in (**c**)). As in Fig. 2.3, for simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 2.7

2.3 Notes

Temporal logics were originally developed by philosophers to reason about how truth and knowledge change over time. They were later adapted in computer science and used to specify the correctness of digital circuits and computer programs. Besides several more expressive temporal logics, Linear Temporal Logic (LTL), Computation Tree Logic (CTL) and the CTL* framework [45, 56], which is a superset of LTL and CTL, are the most commonly encountered. For the applications we consider, the computational expense involved in model checking (see Chap. 3) CTL* outweighs the gains in expressivity and therefore this logic is not considered. LTL and CTL are incomparable in the sense that there exist LTL formulas that cannot be expressed in CTL and vice versa. CTL is a branching time logic that allows for the quantification of specifications over the executions of the system. In other words, a CTL property can be satisfied by the system if it is satisfied over all paths (universal quantification) or if there exists a path that satisfies it (existential quantification). However, the additional semantics of CTL might make the formulation of specifications prone to errors [130, 155], as one must consider all possible executions of a system at the same time. On the other hand, expressing specifications in LTL is more natural because executions are considered one at a time. In the worst case, model checking CTL and LTL specifications respectively requires polynomial and exponential time in the size of the formula. While CTL model checking is computationally cheaper, empirical results suggest that performance is similar [172] for formulas expressible in both logics, since formulas in CTL can be larger than their equivalent LTL representation. Because of its resemblance to natural language, we adopt LTL as a specification formalism.

Fragments of LTL, such as GR(1) [143] and syntactically co-safe LTL (scLTL) [111, 156], have also been proposed as specification languages for verification and control. With particular relevance to this book, scLTL has been primarily used to verify safety of a system [111, 156]. As we will discuss in the next chapter, analysis of a system from an LTL formula ϕ involves constructing an automaton from the negation of the formula, i.e., $B_{\neg\phi}$. A safety property asserts that nothing bad happens to the system, e.g., $\Box$ "*safe*", and negation of a safety formula is called a co-safe formula, e.g., $\Diamond\neg$ "*safe*". As we presented in this chapter, an FSA is sufficient to recognize the words that satisfy a syntactically co-safe LTL formula, which reduces the computational complexity associated with the analysis of the corresponding safety property due to the simple acceptance condition of an FSA. In addition to analysis of safety properties, scLTL formulas are also used to express finite horizon specifications [30].

There are also some differences between the terms used here and elsewhere. The symbols appearing in an LTL formula are usually called *atomic propositions* in the formal methods community [45]. However, we call them *observations* as in this book we use LTL formulas to specify properties of words over observations O produced by transition systems $T = (X, \Sigma, \delta, O, o)$ (see Definition 1.1). This is consistent with control theoretic nomenclatures, where the term *output* is also used [162].

The semantics of LTL formulas are usually given over infinite words in the power set of the set of observations 2^O, as they are normally used to specify properties of transition systems with possibly several observations (atomic propositions satisfied) at each state (see Sect. 1.4). The available off-the-shelf tools for construction of FSA from scLTL formulas (SCHECK2 [117], based on the algorithm from [111]), Büchi automata from LTL formulas (LTL2BA [65, 66], based on algorithms from [173]), and Rabin automata from LTL formulas (LTL2DSTAR [102]) produce automata with input alphabet 2^O, i.e., which accept words over 2^O. This is commonly denoted by labeling transitions of the automaton with Boolean formulas over the observations from O (see Example 2.8). A transition is enabled by the set of subsets of O (i.e., the elements of 2^O) that satisfies the corresponding Boolean formula. However, as the transition systems that we consider in this book have exactly one observation at each state (see Definition 1.1), we simplify the automata produced by SCHECK2, LTL2BA, and LTL2DSTAR to only accept satisfying words over O. For example, Boolean terms or formulas that cannot be satisfied by any individual element of O (e.g., the conjunction $o_1 \wedge o_2$ of any two observations $o_1, o_2 \in O$) are not relevant for the applications we consider and can therefore be simplified. Such a simplification for a Büchi automaton is shown in Example 2.8 and similar simplifications apply to FSA and Rabin automata.

Example 2.8 Consider the LTL formula $\phi = (o_1 U o_2) \wedge \Diamond o_3$, defined over observations $O = \{o_1, o_2, o_3, o_4\}$. The Büchi automaton representation of the formula is obtained using LTL2BA and is given in Fig. 2.5 (states s_0 and s_3 are respectively the initial and final (accepting) state). A transition labeled by an observation is enabled by any subset of O that includes the observations (e.g., the self loop at state s_0 is enabled by observations (o_1),(o_1, o_2),(o_1, o_3),...,(o_1, o_2, o_3),...). Similarly, a transition labeled by a conjunction of observations is enabled under any subset of observations that includes both observations (e.g., the transition between states s_0 and s_1 labeled by the conjunction $o_1 \wedge o_3$, is enabled by observations (o_1, o_3), (o_1, o_2, o_3),(o_1, o_3, o_4),...). Finally, a transition labeled by "true" is enabled by any subset from 2^O.

In this book, we consider only observation from the set O and not the set of subsets 2^O. Therefore, a transition under any conjunction of inputs can never be enabled (i.e., the set of observations satisfying such conjunctions is always empty) and transitions that are never enabled can be safely ignored. In Fig. 2.5, we ignore the transitions from state s_0 to state s_1 and from s_0 to s_3. As a result, state s_1 becomes unreachable and can be ignored as well. This simplification reduces the number of states and transitions in the Büchi automaton, which improves the complexity of the methods that will be discussed subsequently.

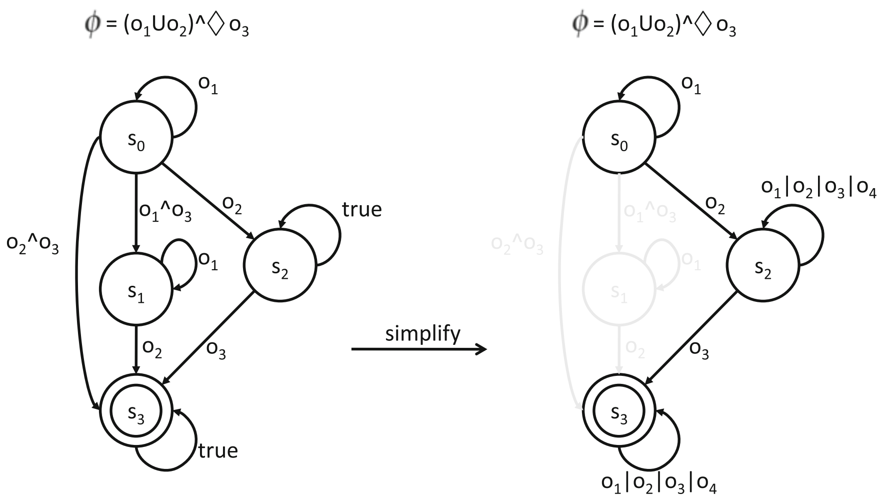


Fig. 2.5 Büchi automata used for our applications can be simplified as described in Example 2.8

Most temporal logics, including LTL, have probabilistic versions. In particular, the probabilistic version of LTL, called Probabilistic LTL (PLTL), is simply defined by adding a probability operator that quantifies the satisfaction probability in front of the formula. Its semantics is defined over a Markov decision process (MDP), the probabilistic version of the transition system defined in Chap. 1 (see Sect. 1.4). Probabilistic temporal logics go beyond the scope of this book, and the interested reader is referred to [4, 14, 15].

There also exist logics, such as Bounded Linear Temporal Logic (BLTL) [188], Signal Temporal Logic (STL) [126], and Metric Temporal Logic (MTL) [108], in which the temporal operators have specific time intervals. In such logics, one can specify eventuality with deadlines (e.g., $\Diamond_{[2,4]} o_1$—"o_1 will happen in between times 2 and 4"), persistence with time bounds (e.g., $\Box_{[3,7]} o_2$—"o_2 will be true for all times between 3 and 7"), etc. In particular, MTL and STL also have quantitative semantics, which allow to quantify how far a system execution is from satisfying a given formula. Recent works [3, 12, 19–21, 54, 59, 91, 94, 95, 107, 151, 176] showed that logics with quantitative semantics can be used to formulate machine learning and control problems as optimization problems with costs induced by quantitative semantics.