Chapter 27
Graph Games and Reactive Synthesis

Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann

Abstract Graph-based games are an important tool in computer science. They have applications in synthesis, verification, refinement, and far beyond. We review graph-based games with objectives on infinite plays. We give definitions and algorithms to solve the games and to give a winning strategy. The objectives we consider are mostly Boolean, but we also look at quantitative graph-based games and their objectives. Synthesis aims to turn temporal logic specifications into correct reactive systems. We explain the reduction of synthesis to graph-based games (or equivalently tree automata) using synthesis of LTL specifications as an example. We treat the classical approach that uses determinization of parity automata and more modern approaches.

27.1 Introduction

Reactive synthesis is the problem of automatically constructing a correct reactive system from a given specification [75]. Graph games (or, equivalently, tree automata) are a central tool in solving the synthesis problems [30, 101]. In this chapter, we shall review the theory of games and synthesis. Besides reactive synthesis, Syntax-Guided Synthesis (SYGUS) has become a popular and promising approach to automatically generate (parts of) programs from specifications. SYGUS differs from reactive synthesis in its goals and especially in the techniques it uses: it is typically done inductively instead of deductively, and is often driven by counterexamples (hence the related term Counterexample-Guided Inductive Synthesis or CEGIS). We refer the interested reader to [8] and the references contained in that paper.

R. Bloem (✉)
Graz University of Technology, Graz, Austria
e-mail: roderick.bloem@iaik.tugraz.at

K. Chatterjee
IST Austria, Klosterneuburg, Austria

B. Jobstmann
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

E.M. Clarke et al. (eds.), *Handbook of Model Checking*,
DOI 10.1007/978-3-319-10575-8_27

We consider two-player perfect-information nonterminating games played on graphs, that proceed for an infinite number of rounds. The state of a game is a vertex of a graph. The graph is partitioned into player-1 states and player-2 states: in player-1 states, player 1 chooses the successor vertex; in player-2 states, player 2 chooses the successor vertex. In each round, the state changes along an edge of the graph to a successor vertex. Thus, the outcome of the game being played for an infinite number of rounds is an infinite path through the graph. These games play a central role in several areas of computer science. One important application arises when the vertices and edges of a graph represent the states and transitions of a reactive system, and the two players represent controllable versus uncontrollable decisions during the execution of the system, which corresponds to the *synthesis* problem for reactive systems. Game-theoretic formulations have proved useful not only for synthesis, but also for the modeling [1, 79], refinement [104], verification [6, 9], testing [17], and compatibility checking [3, 4] of reactive systems. The use of ω-regular objectives is natural in these application contexts. This is because the winning conditions of the games arise from requirements specifications for reactive systems, and the ω-regular sets of infinite paths provide an important and robust paradigm for such specifications [124].

Synthesis is the problem of automatically constructing a correct reactive system from a given specification. Thus, synthesis goes beyond verification, in which both a specification and an implementation have to be given, by automatically deriving the latter from the former. Synthesis is thus a fundamental approach that aims at moving the construction of reactive systems from the imperative to the declarative level. If one is convinced that a complete specification should be written before the implementation is constructed, synthesis is a natural and important endeavor.

In this chapter, we employ the term synthesis exclusively in the setting of synchronous reactive systems, which maintain a constant interaction with the environment. We will assume that both the inputs and the outputs of such a system are Boolean. One way to specify the behavior of such systems is as a set of infinite words over the input and output valuations. The distinction between inputs and outputs is crucial: outputs are under direct control of the system whereas inputs are not. Thus, at any point in time, we must find some output that works for all inputs. More precisely, given a finite input sequence, we must find some output that allows a correct execution for any possible future input.

The synthesis question and the corresponding decidability problem, called *realizability*, were originally posed by Church [75], who used the monadic second-order logic of one successor (S1S) as a specification language. The problem was solved by Rabin [146] and by Büchi and Landweber [30] in the late 1960s. In this chapter, we will focus on the more modern Linear Temporal Logic (LTL) [141]. Initial work on synthesizing systems from temporal specifications assumed cooperative environments and reduced the synthesis problem to satisfiability [76, 84, 126]. The problem of synthesizing systems from specifications in LTL (in adversary environments, which are the focus of this chapter) was studied in [142], where it was shown that the problem is complete for 2EXPTIME. Even though the complexity of the synthesis problem from LTL is significantly lower than that from S1S, which

is non-elementary [166], it discouraged researchers and led to few developments for several decades. This relative silence has been followed by a flurry of activity since around 2005. In this chapter, we will give an overview of both the classical approach to LTL synthesis and the relatively practical approaches that have been proposed recently.

The question of synthesis can be generalized to *controller synthesis*: the question of finishing an incompletely specified system. This problem was first studied by Ramadge and Wonham [148] under the name of Synthesis of Discrete Event Systems. The question here is to control a *plant* in such a way that it fulfills its specification. Again, we must distinguish between the inputs of the plant, which are determined by an uncontrollable environment, and its control parameter: the control parameter must be adjusted continually in such a way that the plant works correctly for any input. The original approach aimed at safety properties only, but can be extended to more expressive specification formalisms using the same techniques that are used in synthesis. Note that in synthesis we have only a specification, whereas in controller synthesis we have both a specification and an incomplete implementation. The distinction is somewhat fluid as implementations can be expressed in temporal logic (using extra variables), while specifications can be expressed as automata, which are a form of transition systems.

A standard approach to verification is automata theoretic: we build an automaton corresponding to the negation of the specification and construct the product with the system that we wish to verify. The system can evolve in several ways, depending on the inputs, which means that the product is nondeterministic, and multiple paths must be searched for an incorrect execution. In synthesis, in contrast, we have two types of freedom: the freedom to choose the inputs, which we cannot control, and the freedom to choose the outputs, which are under our control. Thus, instead of a nondeterministic automaton, the natural model here is a game, or more precisely, an infinite zero-sum graph-based game with two players, which is won iff the specification is satisfied. (Equivalently, we can use tree automata.)

In the next section, we will discuss game theory, with a focus on the games that arise in synthesis settings. We will start with qualitative games, i.e., we will look at games with various Boolean winning conditions that occur in practice. Then, we will consider quantitative games, which occur when we consider more subtle specifications. In Sect. 27.3 we will discuss the classical and some more modern algorithms for LTL synthesis. In Sect. 27.4, we will conclude with related work that we cannot discuss in full detail.

27.2 Theory of Graph-Based Games

In this section we present definitions of game graphs, plays, strategies, and objectives. We will define when a game is won and introduce the appropriate decision problems. We will then discuss the basic techniques and algorithms to solve them.

27.2.1 Game Graphs and Strategies

Game Graphs. A *game graph* $G = \langle (S, E), (S_1, S_2) \rangle$ consists of a *finite* set S of states partitioned into player-1 states S_1 and player-2 states S_2 (i.e., $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$), and a set $E \subseteq S \times S$ of edges such that for all $s \in S$, there exists (at least one) $t \in S$ such that $(s, t) \in E$. In other words, every state has at least one outgoing edge. A *player*-1 *game* is a game graph where $S_1 = S$ and $S_2 = \emptyset$, and vice versa for player 2. The sub-graph of G induced by $U \subseteq S$ is the graph $\langle (U, E \cap (U \times U)), (U \cap S_1, U \cap S_2) \rangle$ (which is not a game graph in general); the sub-graph induced by U is a game graph if for all $s \in U$ there exists a $t \in U$ such that $(s, t) \in E$.

Plays and Strategies. A game on G starting from a state $s_0 \in S$ is played in rounds as follows. If the game is in a player-1 state, then player 1 chooses an outgoing edge to determine the successor state; otherwise the game is in a player-2 state, and player 2 chooses the successor state. This way, the game results in a *play* from s_0, i.e., an infinite path $\rho = s_0 s_1 \cdots \in S^\omega$ such that $(s_i, s_{i+1}) \in E$ for all $i \geq 0$. We denote the set of all plays as $\mathsf{Plays}(G)$. The prefix of length n of ρ is denoted by $\rho(n)$. We often identify ρ with the set of states in ρ, and we use expressions such as $s_0 \in \rho$. A *strategy* for player 1 is a recipe that prescribes how to extend the prefix of a play. Formally, a strategy σ for player 1 is a function $\sigma : S^* S_1 \to S$ such that $(s, \sigma(w \cdot s)) \in E$ for all $w \in S^*$ and $s \in S_1$. An *outcome* of σ from s_0 is a play $s_0 s_1 \ldots$ such that $\sigma(s_0 \ldots s_i) = s_{i+1}$ for all $s_i \in S_1$. Strategy and outcome for player 2 are defined analogously. We denote by Σ and Π the set of strategies for player 1 and player 2, respectively. Given strategies σ and π for player 1 and 2, respectively, and a starting state s_0, there is a unique play (or outcome) $s_0 s_1 \ldots$, denoted $\rho(s_0, \sigma, \pi)$, such that for all $i \geq 0$, if $s_i \in S_1$, then $s_{i+1} = \sigma(s_0 s_1 \ldots s_i)$ and if $s_i \in S_2$, then $s_{i+1} = \pi(s_0 s_1 \ldots s_i)$.

Finite-Memory and Memoryless Strategies. A strategy uses *finite-memory* if it can be encoded by a deterministic transducer $\langle M, m_0, \sigma_u, \sigma_n \rangle$ where M is a finite set (the memory of the strategy), $m_0 \in M$ is the initial memory value, $\sigma_u : M \times S \to M$ is an update function, and $\sigma_n : M \times S_1 \to S$ is a next-move function. The *size* of the strategy is the number $|M|$ of memory values. If the game is in a player-1 state s, the strategy chooses $t = \sigma_n(m, s)$ as the next state (where m is the current memory value), and the memory is updated to $\sigma_u(m, s)$. Formally, $\langle M, m_0, \sigma_u, \sigma_n \rangle$ defines the strategy σ such that $\sigma(w \cdot s) = \sigma_n(\hat{\sigma}_u(m_0, w), s)$ for all $w \in S^*$ and $s \in S_1$, where $\hat{\sigma}_u$ extends σ_u to sequences of states in the usual way. A strategy is *memoryless* if it is independent of the history of the play and depends only on the current state. In other words, a memoryless strategy σ has only one memory state, i.e., $|M| = 1$, and hence the strategy is specified as $\sigma : S_1 \to S$. For a finite-memory strategy σ, G_σ is the graph obtained as the product of G with the transducer defining σ, where $(\langle m, s \rangle, \langle m', s' \rangle)$ is a transition in G_σ if $m' = \sigma_u(m, s)$ and either $s \in S_1$ and $s' = \sigma_n(m, s)$, or $s \in S_2$ and $(s, s') \in E$.

27.2.2 Objectives

In this section we will define objectives. An *objective* for G is a set $\varphi \subseteq S^\omega$.

Qualitative Objectives. For an infinite play ρ we denote by $\mathsf{Inf}(\rho)$ the set of states that occur infinitely often in ρ. We consider the following objectives:

- *Reachability Objectives.* A reachability objective is defined by a set $F \subseteq S$ of target states, and the objective requires that a state in F is visited at least once. Formally, $\mathsf{Reach}_G(F) = \{\rho \in \mathsf{Plays}(G) \mid \exists s \in \rho : s \in F\}$. The dual of reachability objectives are safety objectives, and a safety objective is defined by a set $F \subseteq S$ of safe states, and the objective requires that only states in F are visited. Formally, $\mathsf{Safe}_G(F) = \{\rho \in \mathsf{Plays}(G) \mid \forall s \in \rho : s \in F\}$.
- *Büchi Objectives.* A Büchi objective is defined by a set $B \subseteq S$ of target states, and the objective requires that a state in B is visited infinitely often. Formally, $\mathsf{Buchi}_G(B) = \{\rho \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\rho) \cap B \neq \emptyset\}$. Büchi objectives represent liveness specifications, and the dual of a Büchi objective is called a co-Büchi objective. A co-Büchi objective consists of a set $C \subseteq S$ of states and requires states outside C to be visited finitely often, i.e., $\mathsf{coBuchi}_G(C) = \{\rho \in \mathsf{Plays}(G) \mid \mathsf{Inf}(\rho) \subseteq C\}$.
- *Rabin and Streett Objectives.* Rabin and Streett objectives are obtained as Boolean combinations of Büchi and co-Büchi objectives. A *Rabin specification* for the game graph G is a finite set $R = \{(E_1, F_1), \ldots, (E_d, F_d)\}$ of pairs of sets of states, that is, $E_j \subseteq S$ and $F_j \subseteq S$ for all $1 \leq j \leq d$. The pairs in R are called Rabin pairs. We assume without loss of generality that $\bigcup_{1 \leq j \leq d}(E_j \cup F_j) = S$. The Rabin objective requires that for some $1 \leq j \leq d$, all states in the left-hand set E_j are visited finitely often, and some state in the right-hand set F_j is visited infinitely often. Thus, the Rabin objective defined by R is the set $\mathsf{Rabin}_G(R) = \{\rho \in \mathsf{Plays}(G) \mid (\exists 1 \leq j \leq d)(\mathsf{Inf}(\rho) \cap E_j = \emptyset \wedge \mathsf{Inf}(\rho) \cap F_j \neq \emptyset)\}$ of winning paths. Note that the co-Büchi objective $\mathsf{coBuchi}_G(C)$ is equal to the single-pair Rabin objective $\mathsf{Rabin}_G(\{(S \setminus C, S)\})$, and the Büchi objective $\mathsf{Buchi}_G(B)$ is equal to the two-pair Rabin objective $\mathsf{Rabin}_G(\{(\emptyset, B), (S, S)\})$.[1] The complements of Rabin objectives are called Streett objectives. A *Streett specification* for G is likewise a set $Q = \{(E_1, F_1), \ldots, (E_d, F_d)\}$ of pairs of sets of states $E_j \subseteq S$ and $F_j \subseteq S$ such that $\bigcup_{1 \leq j \leq d}(E_j \cup F_j) = S$. The pairs in Q are called Streett pairs. The Streett objective Q requires that for every Streett pair (E_j, F_j), $1 \leq j \leq d$, if some state in the right-hand set F_j is visited infinitely often, then some state in the left-hand set E_j is visited infinitely often. Formally, the Streett objective defined by Q is the set $\mathsf{Streett}_G(Q) = \{\rho \in \mathsf{Plays}(G) \mid (\forall 1 \leq j \leq d)(\mathsf{Inf}(\rho) \cap E_j \neq \emptyset \vee \mathsf{Inf}(\rho) \cap F_j = \emptyset)\}$ of winning paths. Note that $\mathsf{Streett}_G(Q) = \mathsf{Plays}(G) \setminus \mathsf{Rabin}_G(Q)$.

[1] Note that no run can satisfy the condition expressed by the second pair, which is however required by the definition.

- *Parity Objectives.* Let $p : S \to \mathbb{N}_0$ be a *priority function*. The *parity* objective $\mathsf{Parity}_G(p) = \{\rho \in \mathsf{Plays}(G) \mid \min\{p(s) \mid s \in \mathsf{Inf}(\rho)\} \text{ is even}\}$ requires that the minimum of the priorities of the states visited infinitely often be even. The special cases of *Büchi* and *co-Büchi* objectives correspond to the case with two priorities, $p : S \to \{0, 1\}$ and $p : S \to \{1, 2\}$ respectively. (Here, priorities 0 and 2 are for the accepting states, 1 is for the rejecting states.)

We refer to the above objectives as qualitative objectives since they are defined by Boolean combinations of sets that are subsets of S.

Relationship Between Rabin, Streett and Parity Objectives. We have already seen how Büchi and co-Büchi objectives are special cases of Rabin, Streett and parity objectives. We now present the relationship between Rabin, Streett and parity objectives. Parity objectives are also called *Rabin-chain* objectives, as they are a special case of Rabin objectives [169]: if the sets of a Rabin specification $R = \{(E_1, F_1), \ldots, (E_d, F_d)\}$ form a chain $E_1 \subsetneq F_1 \subsetneq E_2 \subsetneq F_2 \subsetneq \cdots \subsetneq E_d \subsetneq F_d$, then $\mathsf{Rabin}_G(R) = \mathsf{Parity}_G(p)$ for the priority function $p\colon S \to \{0, 1, \ldots, 2d\}$ that for every $1 \leq j \leq d$ assigns to each state in $E_j \setminus F_{j-1}$ the priority $2j - 1$, and to each state in $F_j \setminus E_j$ the priority $2j$, where $F_0 = \emptyset$. Conversely, given a priority function $p\colon S \to \{0, 1, \ldots, 2d\}$, we can construct a chain $E_1 \subsetneq F_1 \subsetneq \cdots \subsetneq E_{d+1} \subsetneq F_{d+1}$ of $d + 1$ Rabin pairs such that $\mathsf{Parity}_G(p) = \mathsf{Rabin}_G(\{(E_1, F_1), \ldots, (E_{d+1}, F_{d+1})\})$ as follows: let $E_1 = \emptyset$ and $F_1 = p^{-1}(0)$, and for all $1 \leq j \leq d + 1$, let $E_j = F_{j-1} \cup p^{-1}(2j - 3)$ and $F_j = E_j \cup p^{-1}(2j - 2)$. Hence, the parity objectives are a subclass of the Rabin objectives that is closed under complementation. It follows that every parity objective is both a Rabin objective and a Streett objective. The parity objectives are of special interest, because every ω-regular objective can be turned into a parity objective by modifying the game graph (take the synchronous product of the game graph with a deterministic parity automaton that accepts the ω-regular objective) [133]. Moreover, parity objectives enjoy several attractive computational properties (see discussion of algorithms for parity games in Sect. 27.2.4).

Quantitative Objectives. We consider three classical quantitative objectives defined with weight functions on the edges of the graph. Let $w : E \to \mathbb{Z}$ be a *weight function*, where positive numbers represent rewards. We denote by W the largest weight (in absolute value) according to w.

- *Energy Objectives.* Given a play ρ, the *energy level* of a prefix $\gamma = s_0 s_1 \ldots s_n$ of the play is $\mathsf{EL}(\gamma) = \sum_{i=0}^{n-1} w((s_i, s_{i+1}))$. Given an initial credit $c_0 \in \mathbb{N} \cup \{\infty\}$, the *energy* objective $\mathsf{PosEnergy}_G(c_0) = \{\rho \in \mathsf{Plays}(G) \mid \forall n \geq 0 : c_0 + \mathsf{EL}(\rho(n)) \geq 0\}$ requires that the energy level is always non-negative.
- *Mean-Payoff Objectives.* The *mean-payoff value* of a play $\rho = s_0 s_1 \ldots$ is $\mathsf{MP}(\rho) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{EL}(\rho(n))$. Given a threshold $\theta \in \mathbb{Q}$, the *mean-payoff* objective $\mathsf{MeanPayoff}_G(\theta) = \{\rho \in \mathsf{Plays}(G) \mid \mathsf{MP}(\rho) \geq \theta\}$ requires that the mean-payoff value be at least θ.
- *Discounted Objectives.* Given a discount factor $0 < \lambda < 1$, the *discounted value* of a play $\rho = s_0 s_1 \ldots$ is $\mathsf{Disc}(\lambda, \rho) = \sum_{i=0}^{\infty} \lambda^i \cdot w((s_i, s_{i+1}))$. Given a

threshold $\theta \in \mathbb{Q}$, the *discounted* objective $\mathsf{Discounted}_G(\lambda, \theta) = \{\rho \in \mathsf{Plays}(G) \mid \mathsf{Disc}(\lambda, \rho) \geq \theta\}$ requires that the discounted value be at least θ.

In the sequel, when the game G is clear from the context, we omit the subscript in objective names.

27.2.3 Winning and Optimal Strategies; Decision Problems

We now define the notion of winning in games and decision problems.

Winning Strategies and Sets. Given a game graph G, a starting state s_0 and an objective φ, a strategy σ is *winning* for player 1 from s_0 for φ if for all strategies π for player 2 we have $\rho(s_0, \sigma, \pi) \in \varphi$. The set of winning states $W_1(\varphi) = \{s_0 \mid \exists \sigma \in \Sigma.\ \forall \pi \in \Pi.\ \rho(s_0, \sigma, \pi) \in \varphi\}$ is the set of states s_0 such that player 1 has a winning strategy from s_0 for φ (note that an objective is a set of plays). The winning set $W_2(\varphi) = \{s_0 \mid \exists \pi \in \Pi.\ \forall \sigma \in \Sigma.\ \rho(s_0, \sigma, \pi) \in \varphi\}$ is defined analogously. We will consider the winning sets and strategies for objectives defined in the previous subsection, i.e., reachability, safety, Büchi, co-Büchi, parity, Rabin, Streett, energy, mean-payoff and discounted objectives. For energy objectives, we will also consider the *finite initial credit* problem, where the winning region is the set of states s_0 such that there exists a finite initial credit c_0 such that $s_0 \in W_1(\mathsf{PosEnergy}_G(c_0))$.

Decision Problems. The decision problems that we consider consist of an input game graph G, an objective φ and a state s_0, and the decision problem asks whether $s_0 \in W_1(\varphi)$. We will also consider the decision problem for the finite initial credit problem, which asks whether a given state s_0 is in the winning set for the finite initial credit problem.

27.2.4 Complexity and Algorithms for Graph Games with Qualitative Objectives

In this section we will discuss the results related to graph games with qualitative objectives. We will focus on the strategy complexity, computational complexity, and algorithms. We will mention the basic techniques and relevant pointers to the literature. We will first discuss symbolic algorithms for game solving.

Symbolic Algorithms. The symbolic algorithms for game solving are obtained by characterizing the winning set using μ-calculus formulae (cf. Chap. 26 in this Handbook [24]). A μ-calculus formula is a succinct description of a nested iterative algorithm that uses only set operations and the predecessor operators (described in the next paragraph). All the set operations and predecessor computations are symbolic

steps that are available as primitive operations in, e.g., a BDD library (cf. Chap. 7 in this Handbook [29]) such as CuDD [164]. Thus, a μ-calculus formula for the winning set presents a symbolic algorithm for game solving. We will describe the μ-calculus formula for reachability and Büchi games. The μ-calculus formulas for parity games are presented in [86] and they were later generalized to Rabin and Streett games in [138].

Reachability and Safety Games. We first present the classical algorithm to solve reachability games. Let us first define the *predecessor* operator. Given a set $X \subseteq S$ of states, the predecessor operator $\mathsf{Pre}_1(X)$ is defined as follows

$$\mathsf{Pre}_1(X) = \big\{s \in S_1 \bigm| \exists t \in X.(s,t) \in E\big\} \cup \big\{s \in S_2 \bigm| \forall t \in S.(s,t) \in E \rightarrow t \in X\big\}.$$

In other words, $\mathsf{Pre}_1(X)$ is the set of states such that either the state is a player-1 state and there is a next state in X or the state is a player-2 state and all choices lead to a next state in X. The dual predecessor operator is as follows:

$$\mathsf{Pre}_2(X) = \big\{s \in S_2 \bigm| \exists t \in X.(s,t) \in E\big\} \cup \big\{s \in S_1 \bigm| \forall t \in S.(s,t) \in E \rightarrow t \in X\big\}.$$

The classical algorithm for games with reachability objectives is the fixpoint computation of the $\mathsf{Pre}_1(\cdot)$ operator. Given a target set T, let $T_0 = T$, and for $i \geq 0$, let T_{i+1} be defined inductively as $T_{i+1} := T_i \cup \mathsf{Pre}_1(T_i)$. Let us consider the fixpoint, which is also called the *attractor* of player 1 to T. Let $T_* = \mathsf{Attr}_1(T) = \bigcup_{i \geq 0} T_i$. A memoryless strategy for player 1 for T_* is defined as follows: for a state $s \in (T_{i+1} \setminus T_i) \cap S_1$ choose an edge (s,t) such that $t \in T_i$ (such an edge exists by construction). It is easy to show by induction that for all $s \in T_i$, player 1 can ensure to reach T within i steps against all player-2 strategies. Hence $T_* \subseteq W_1(\mathsf{Reach}(T))$. Let $\overline{T}_* = S \setminus T_*$. For all $s \in \overline{T}_* \cap S_1$, there are no outgoing edges (s,t) with $t \in T_*$ (otherwise, s would have been included in T_*); and for all $s \in \overline{T}_* \cap S_2$, there is an outgoing edge (s,t) with $t \in \overline{T}_*$ (otherwise, s would have been included in T_*). A memoryless strategy for player 2 that for all $s \in \overline{T}_* \cap S_2$ chooses an outgoing edge (s,t) with $t \in \overline{T}_*$ ensures against all player-1 strategies that $\overline{T}_*$ is not left. Thus we have $\overline{T}_* \subseteq W_2(\mathsf{Safe}(S \setminus T))$. A linear-time algorithm to compute $W_1(\mathsf{Reach}(T))$ is given in [13, 106]. We summarize the main results of reachability and safety games in the following theorem.

Theorem 1 *Given a game graph G with n vertices and m edges and a target set T, the following assertions hold*:

1. $W_1(\mathsf{Reach}(T)) = S \setminus W_2(\mathsf{Safe}(S \setminus T))$ *and memoryless winning strategies exist for both players.*
2. *The winning set $W_1(\mathsf{Reach}(T))$ can be computed in $O(m)$ (linear) time.*
3. *The winning set $W_1(\mathsf{Reach}(T))$ can be computed symbolically with the μ-calculus formula $\mu X.[T \cup \mathsf{Pre}_1(X)]$.*

Büchi and co-Büchi Games. The algorithm for Büchi games is obtained by repeatedly applying the attractor computation (reachability game solutions). Informally,

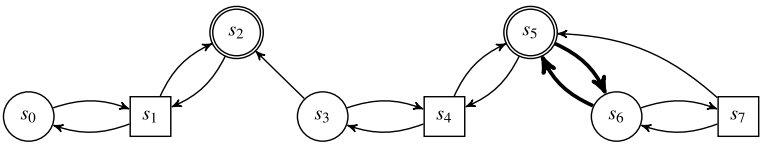


Fig. 1 Büchi game of Example 1

the algorithm is as follows: let B be the set of Büchi states. We first compute the set $A_1 = W_1(\mathsf{Reach}(B)) = \mathsf{Attr}_1(B)$ such that player 1 has a strategy to reach B at least once. In the complement set of A_1, player 1 cannot even reach B once and hence is clearly not winning. The complement set $\overline{A}_1$ and the player-2 attractor $\mathsf{Attr}_2(\overline{A}_1)$ are removed from the graph. The process is iterated unless the set $\overline{A}_1$ is empty. If $\overline{A}_1$ is empty, then from all states in A_1 player 1 can ensure to reach B and stay in A_1 and hence ensure that the set B is visited infinitely often. We now formally describe an iteration j of the algorithm: the set of states at iteration j is denoted by S^j, the game graph by G^j, and the set of Büchi states $B \cap S^j$ by B^j. Given a game graph G and a set U of states, we denote by $G \restriction U$ the game graph induced by U. We have that G^j is the game graph induced by S^j. At iteration j, the algorithm first finds the set of states A_1^j from which player 1 can ensure that the play reaches the set B^j, i.e., computes $\mathsf{Attr}_1(B^j)$ in G^j. The rest of the states $\overline{A}_1^j = S^j \setminus A_1^j$ are winning for player 2. Then the set of states W_{j+1}, from which player 2 can ensure reaching $\overline{A}_1^j$ i.e., $\mathsf{Attr}_2(\overline{A}_1^j)$ in G^j, is computed. The set W_{j+1} is winning for player 2, and *not* for player 1 in G^j and also in G. Thus, it is removed from the vertex set to obtain game graph G^{j+1}. The algorithm then iterates on the reduced game graph, i.e., proceeds to iteration $j+1$ on G^{j+1}. The correctness proof of the algorithm shows that when the algorithm terminates, all the remaining states are winning for player 1. The pseudocode of the algorithm is described in Algorithm 1. For improved algorithms for Büchi games see [52, 53, 61, 64].

Example 1 We illustrate the algorithm for Büchi games on the example game graph shown in Fig. 1. The player-1 states are depicted as circles and player-2 states as boxes; and Büchi states are indicated as double circles. The set $\overline{A}_1^0$ is the set $\{s_0, s_1\}$ and its player-2 attractor is $\{s_0, s_1, s_2\}$. In the following iteration the set $\overline{A}_1^1$ is the set $\{s_3, s_4\}$ and its player-2 attractor is also the set $\{s_3, s_4\}$. In the next iteration the set $\overline{A}_1^2$ is empty, and thus the algorithm returns $(\{s_5, s_6, s_7\}, \{s_0, s_1, s_2, s_3, s_4\})$. The winning strategy for player 1 in the set $\{s_5, s_6, s_7\}$ (indicated by bold arrows in Fig. 1) is as follows: in state s_5 choose the successor s_6 and in s_6 choose the successor s_5.

Characterization of Winning Set by μ-calculus Formula. The μ-calculus formula to characterize the winning set for Büchi objectives is as follows:

$$\nu Y.\mu X.\big[\big(B \cap \mathsf{Pre}_1(Y)\big) \cup \big((S \setminus B) \cap \mathsf{Pre}_1(X)\big)\big].$$

Algorithm 1: Classical algorithm for Büchi Games

Input : A game graph $G = \langle (S, E), (S_1, S_2) \rangle$ and $B \subseteq S$.
Output: $(S \setminus W, W)$: the winning set partition.
1. $G^0 := G$; $S^0 := S$; 2. $W_0 := \emptyset$; 3. $j := 0$
4. **repeat**
  4.1 $W_{j+1} :=$ AvoidSetClassical$(G^j, B \cap S^j)$
  4.2 $S^{j+1} := S^j \setminus W_{j+1}$; $G^{j+1} = G \restriction S^{j+1}$; $j := j + 1$;
until $W_j = \emptyset$
5. $W := \bigcup_{k=1}^{j} W_k$;
6. **return** $(S \setminus W, W)$.

Procedure AvoidSetClassical
Input: Game graph G^j and $B^j \subseteq S^j$.
Output: set $W_{j+1} \subseteq S^j$.
1. $A_1^j :=$ Attr$_1(B^j)$ in G^j; 2. $\overline{A}_1^j := S^j \setminus A_1^j$; 3. $W_{j+1} :=$ Attr$_2(\overline{A}_1^j)$ in G^j;

The argument that the above formula gives the winning set for Büchi objectives is as follows: let $Y_* = \nu Y.\mu X.[(B \cap \mathsf{Pre}_1(Y)) \cup ((S \setminus B) \cap \mathsf{Pre}_1(X))]$. Since Y_* is a fixpoint, if we replace Y by Y_* we would obtain the same result, i.e.,

$$Y_* = \mu X.\big[\big(B \cap \mathsf{Pre}_1(Y_*)\big) \cup \big((S \setminus B) \cap \mathsf{Pre}_1(X)\big)\big].$$

Let $T = B \cap Y_*$; and all states in $B \cap Y_*$ satisfy $\mathsf{Pre}_1(Y_*)$, i.e., in states of $B \cap Y_*$ player 1 can ensure that the next state is in Y_*. Treating the set $T = B \cap Y_* = B \cap \mathsf{Pre}_1(Y_*)$ as the target set for reachability objectives, it follows that for all states in Y_* player 1 can ensure to reach T. Hence player 1 can ensure to reach T from all states in Y_* and Y_* is never left, and thus T is visited infinitely often. Since $T \subseteq B$, it follows that the Büchi objective is satisfied. A similar argument shows that in the complement of the μ-calculus formula, player 2 can ensure the complement co-Büchi objective (we also refer the reader to Chap. 26 in this Handbook [24] for an excellent exposition on μ-calculus). The main results for Büchi games are summarized as follows.

Theorem 2 *Given a game graph G, with set B of Büchi states, the following assertions hold*:

1. $W_1(\mathsf{Buchi}(B)) = S \setminus W_2(\mathsf{coBuchi}(S \setminus B))$ *and memoryless winning strategies exist for both players.*
2. *The winning set $W_1(\mathsf{Buchi}(B))$ can be computed in $O(n \cdot m)$ (quadratic) time.*
3. *The winning set $W_1(\mathsf{Buchi}(B))$ can be computed symbolically with the μ-calculus formula $\nu Y.\mu X.[(B \cap \mathsf{Pre}_1(Y)) \cup ((S \setminus B) \cap \mathsf{Pre}_1(X))]$.*

Parity Games. Emerson and Jutla [86] established the equivalence of solving 2-player parity games and μ-calculus model checking (see Chap. 26 in this Hand-

book [24]). This intriguing connection led to much research attempting to solve 2-player parity games in polynomial time. Alas, the problem is still open. The classical algorithm for solving parity games proceeds by a recursive decomposition of the problem and repeatedly solving games with reachability objectives [128, 169]. The algorithm generalizes the algorithm presented for Büchi games, and the correctness proof establishes the existence of memoryless winning strategies for both players. The running time of the algorithm for games with n states, m edges, and d priorities is $O(n^{d-1} \cdot m)$. Jurdziński [111] gave an improved algorithm to solve parity games based on a notion of ranking functions and progress measures. This algorithm, called the *small progress measure* algorithm, has a running time of $O((\frac{2n}{d})^{\lfloor \frac{d}{2} \rfloor} \cdot m)$; moreover, there exists a family of games on which the running time of the algorithm is exponential. Another notable algorithm for solving parity games is the *strategy improvement* algorithm [175]. This algorithm iterates local optimizations of memoryless strategies which converge to a globally optimal strategy. Also see [154] for another strategy improvement scheme. Based on the strategy improvement algorithm, a randomized subexponential-time algorithm (with an expected running time of $O(2^{\sqrt{n \cdot \log n}})$) for solving parity games was presented by Björklund et al. [15]. Friedmann [97] showed that there exists a family of games on which the running time of the strategy-improvement algorithms is exponential, and for a more elaborate description of lower bounds for strategy-improvement schemes see [98]. Jurdziński et al. [112] gave a deterministic subexponential-time algorithm for solving 2-player games with parity objectives. By combining the small progress measure algorithm [111] and the deterministic subexponential-time algorithm [112], an improved algorithm was presented in [153] with roughly $O((\frac{3n}{d})^{\lfloor \frac{d}{3} \rfloor} \cdot m)$ running time. We summarize the results in the following theorem.

Theorem 3 *Given a game graph G, with a priority function p with d priorities, the following assertions hold*:

1. $W_1(\mathsf{Parity}(p)) = S \setminus W_2(\mathsf{Plays}(G) \setminus \mathsf{Parity}(p))$ *and memoryless winning strategies exist for both players.*
2. *Given a state s, whether $s \in W_1(\mathsf{Parity}(p))$ can be decided in NP $\cap$ coNP.*
3. *The winning set $W_1(\mathsf{Parity}(p))$ can be computed in $O((\frac{3n}{d})^{\lfloor \frac{d}{3} \rfloor} \cdot m)$, and also in $n^{O(\sqrt{n})}$ time.*
4. *The winning set $W_1(\mathsf{Parity}(p))$ can be computed symbolically with a μ-calculus formula of alternation depth $d - 1$.*

Rabin and Streett Games. Gurevich and Harrington [101] showed that for 2-player games with ω-regular objectives, finite-memory strategies suffice for winning. The construction of finite-memory winning strategies is based on a data structure, called a *latest appearance record* (LAR), which remembers the order of the latest appearances of the states in a play. Emerson and Jutla [85] established that for 2-player games with Rabin objectives, memoryless strategies suffice for winning. The results of Dziembowski et al. [80] give precise memory requirements for strategies in 2-player games with ω-regular objectives: their construction of strategies is based on

a tree representation of a Muller objective, called the *Zielonka tree*, which was introduced in [180] (for details of Muller objectives see [169, 180]). It follows from these results that for Streett objectives with d-pairs, $d!$ memory is both necessary and sufficient. Emerson and Jutla [85] showed that the solution problem for Rabin objectives is NP-complete, and dually, coNP-complete for Streett objectives. The notable algorithms for games with Rabin and Streett objectives include the adaptation of the classical algorithm of Zielonka [180] for Muller games specialized to Rabin and Streett games (see [105] for an exposition); an algorithm that is based on a reduction to the emptiness problem for weak alternating automata [119]; a generalization of the small progress measure algorithm for parity games to Rabin and Streett games [138]; and a generalization of the subexponential-time algorithm for parity games [112] to Rabin and Streett games [62]. Symbolic algorithms for Rabin and Streett games are presented in [138].

Theorem 4 *Given a game graph G, with a set $P = \{(E_1, F_1), \ldots, (E_d, F_d)\}$ of d pairs of sets of states, the following assertions hold*:

1. *We have $W_1(\mathsf{Rabin}(P)) = S \setminus W_2(\mathsf{Plays}(G) \setminus \mathsf{Rabin}(P))$, and $W_1(\mathsf{Streett}(P)) = S \setminus W_2(\mathsf{Plays}(G) \setminus \mathsf{Streett}(P))$. Memoryless winning strategies exist for Rabin objectives, and for Streett objectives $d!$ memory is necessary and sufficient.*
2. *Given a state s, the decision problem whether $s \in W_1(\mathsf{Rabin}(P))$ is NP-complete, and the decision problem whether $s \in W_1(\mathsf{Streett}(P))$ is coNP-complete.*
3. *The winning sets $W_1(\mathsf{Rabin}(P))$ and $W_1(\mathsf{Streett}(P))$ can be computed in $O(d! \cdot n^d \cdot m)$ time.*

Boolean Combinations. We now discuss some Boolean combinations of the above objectives that have been used in synthesis. The class of GR(1) (Generalized Reactivity(1)) conditions was introduced in [140]. A GR(1) objective is specified as an implication between a conjunction of k_1 Büchi objectives (the assumptions) and a conjunction of k_2 Büchi objectives (the guarantees). A large class of objectives in synthesis can be specified as GR(1) specifications [140], and games with GR(1) conditions can be solved in time $O(n^2 \cdot m \cdot k_1 \cdot k_2)$ [140] and also in time $O(n \cdot m \cdot (k_1 \cdot k_2)^2)$ [18]. Games with generalized parity objectives (conjunction and disjunction of parity objectives) have been studied in [62].

27.2.5 Complexity and Algorithms for Graph Games with Quantitative Objectives

In this section we will discuss the results related to solving graph games with quantitative objectives. Again we will focus on the strategy complexity, computational complexity, and algorithms. We will mention the basic techniques, and the relevant pointers to literature.

Mean-Payoff and Energy Games. The existence of memoryless winning strategies in mean-payoff games was established in [83], and the proof was based on induction on the number of edges and establishing the equivalence of the mean-payoff game played for finitely many steps and the mean-payoff game played forever. The algorithmic solution for mean-payoff games was given in [181], using a *value iteration* algorithm. Consider a sequence of *valuations* $(v_i)_{i\geq 0}$, where each valuation v_i is a function $v_i : S \to \mathbb{Z}$ defined as follows: (1) $v_0(s) = 0$ for all $s \in S$; and (2) for $i \geq 0$ we have

$$v_{i+1}(s) = \begin{cases} \max_{(s,t)\in E}\{w(s,t) + v_i(t)\} & \text{for } s \in S_1 \\ \min_{(s,t)\in E}\{w(s,t) + v_i(t)\} & \text{for } s \in S_2. \end{cases}$$

Observe that v_k can be computed in time $O(k \cdot m)$, where m is the number of edges. Let v_* be the optimal valuation of the mean-payoff game. The results of [181] show that

$$\frac{v_k}{k} - \frac{2 \cdot n \cdot W}{k} \leq v_* \leq \frac{v_k}{k} + \frac{2 \cdot n \cdot W}{k},$$

where W is the maximum absolute value of the weights. Furthermore, it was shown in [181] that by computing v_k for $k = 4 \cdot n^3 \cdot W$, the optimal value vector v_* can be computed. The result for energy games is similar: existence of memoryless winning strategies was established in [33], and a value iteration algorithm was also given. The running time of the value iteration algorithm is $O(n^3 \cdot m \cdot W)$. Recently, the value iteration algorithm has been improved by [28] to obtain an algorithm that runs in $O(n^2 \cdot m \cdot W)$ time. A strategy improvement algorithm for mean-payoff games is presented in [16]. We summarize the result in the following theorem (we present the result for mean-payoff objectives, but the result for energy objectives is similar).

Theorem 5 *Given a game graph G, with a weight function w,*

1. *For all $\theta \in \mathbb{N}$, we have $W_1(\mathsf{MeanPayoff}(\theta)) = S \setminus W_2(\mathsf{Plays}(G) \setminus \mathsf{MeanPayoff}(\theta))$ and memoryless winning strategies exist for both players.*
2. *Given a state s and $\theta \in \mathbb{N}$, whether $s \in W_1(\mathsf{MeanPayoff}(\theta))$ belongs to NP $\cap$ coNP.*
3. *For $\theta \in \mathbb{N}$, the winning set $W_1(\mathsf{MeanPayoff}(\theta))$ can be computed in $O(n^2 \cdot m \cdot W)$ time.*

Discounted Games. The existence of memoryless strategies in discounted games can be obtained as a special case of the result of Shapley [159]. The algorithm to solve discounted games is similar to the value iteration for mean-payoff games, and in discounted games the valuations need to be computed for $O(n^3 \cdot \frac{1}{1-\lambda})$ steps (see [92, 181] for details). The results for discounted games are as follows.

Theorem 6 *Given a game graph G, with a weight function w,*

1. *For all $\theta \in \mathbb{N}$ and rational $0 < \lambda < 1$, we have $W_1(\mathsf{Discounted}(\lambda, \theta)) = S \setminus W_2(\mathsf{Plays}(G) \setminus \mathsf{Discounted}(\lambda, \theta))$ and memoryless winning strategies exist for both players.*

2. *Given a state* s, *rational* $0 < \lambda < 1$, *and* $\theta \in \mathbb{N}$, *whether* $s \in W_1(\mathsf{Discounted}(\lambda, \theta))$ *can be decided in NP* $\cap$ *coNP.*
3. *For* $\theta \in \mathbb{N}$ *and rational* $0 < \lambda < 1$, *the winning set* $W_1(\mathsf{Discounted}(\lambda, \theta))$ *can be computed in* $O(n^3 \cdot m \cdot \frac{1}{1-\lambda})$ *time.*

27.2.6 Reducibility Between Graph Games

We now discuss the reducibility between various classes of games.

Parity to Mean-Payoff Games. A reduction of parity games to mean-payoff games was presented in [110]. The reduction is defined on the same game graph, and the reduction function is as follows: for a state with priority i, the reward is $(-1)^i \cdot n^i$, where n is the number of states. Then we have $W_1(\mathsf{Parity}(p)) = W_1(\mathsf{MeanPayoff}(0))$, i.e., the winning sets for parity and mean-payoff objectives coincide. The question of whether the decision problem for mean-payoff objectives can be reduced to parity objectives is open.

Mean-Payoff to Discounted Games. The reduction of mean-payoff games to discounted games was presented in [181]. The reduction was defined on the same game graph, with the same reward function, and the discount factor of λ defined as $1 - \frac{1}{4 \cdot n^3 \cdot W}$, where n is the number of states and W is the maximum absolute value of the weights. The question of whether the decision problem for discounted objectives can be reduced to mean-payoff objectives is open.

Energy Games and Mean-Payoff Games. The equivalence of the decision problem for finite initial credit for energy objectives and the mean-payoff objectives was established in [22]. The main argument is as follows: by the existence of memoryless strategies it follows that if the answer to the mean-payoff objectives with threshold $\theta = 0$ is true, then player 1 can fix a memoryless strategy such that in all cycles the sum of the rewards is non-negative, and this exactly coincides with the finite initial credit problem (where after a prefix, the sum of the rewards in cycles is non-negative). A similar argument holds for the reduction in the other direction.

27.2.7 Extensions

We briefly discuss several extensions of such games which have been studied in the literature, and give a few relevant references (there is no attempt to be exhaustive).

Stochastic and Concurrent Games. In this chapter we focused on games where the transitions are deterministic, and the games were turn-based (in each round one of the players makes a move). The class of *turn-based stochastic games* (games with

a probabilistic transition function) has been widely studied, for example in [34–36, 65, 77, 78]. The class of *concurrent* games where both players make their move simultaneously has also been studied in depth [5, 7, 37–39, 67, 91, 102, 127, 159]. For a survey of stochastic and concurrent games see [56].

Partial-Information Games. In *partial-information* games, the players choose their moves based on incomplete information about the state of the game. Such games are harder to solve than the corresponding perfect-information games. For example, turn-based deterministic (2-player) games with partial information and zero-sum reachability/safety objectives are EXPTIME-complete [149]. In the presence of more than two players, turn-based deterministic games with partial information and reachability objectives (for one of the players) are even undecidable [149]. A key technique to solve partial-information games (when possible) is reduction to perfect-information games, using a subset construction on the state space similar to the determinization of finite automata. The results in [54] present a close connection between a subclass of partial-information turn-based games and perfect-information concurrent games. The algorithmic analysis of partial-information stochastic games with ω-regular objectives has been studied in [44, 48, 49, 135]; the complexity of partial-information Markov decision processes has been studied in [40, 46, 136]. The more general class of partial-information stochastic games where both players have partial information has been studied in [14, 43]. Another interesting variety of partial-information games is the class of games where the starting state is unknown [103]. See [41, 47] for surveys related to partial-observation games.

Infinite-State Games. There are several extensions of games played on finite state spaces to games played on infinite state spaces. Notable examples are *pushdown* games and *timed* games. In the case of pushdown games, the state of a game encodes an unbounded amount of information in the form of the contents of a stack. Deterministic pushdown games are solved in [176] (see [178] for a survey); probabilistic pushdown games in [89, 90]; and pushdown games with quantitative objectives in [51, 68, 72]. In the case of timed games, the state of a game encodes an unbounded amount of information in the form of real-numbered values for finitely many clocks. Timed games are studied in [2, 123].

Quantitative and Qualitative Objectives. The problem of solving turn-based games with a conjunction of quantitative and qualitative objectives has been studied in [42, 60]; and multi-dimensional quantitative objectives in [27, 45, 71, 73, 174]. The problem of multi-dimensional objectives has also been widely studied for stochastic models [25, 26, 50, 66, 74, 87, 96].

Logical Framework for Games. Logical frameworks where properties for games can be described concisely with precise semantics for reasoning about games have also been studied in the literature. Some prominent examples of logical frameworks for reasoning about games are alternating-time temporal logic (ATL) and game logic [9]; strategy logic and various fragments [63, 130, 131]; and coordination logic [95].

27.3 Reactive Synthesis

27.3.1 Introduction

In this section, we summarize techniques to automatically construct reactive systems from specifications. A reactive system [124, 125] is a system that maintains an ongoing interaction with its environment. Examples of reactive systems are concurrent programs, air traffic control systems, controllers for mechanical devices, and digital hardware designs. We will use LTL as our representative specification language for concurrent systems (see Chap. 2 in this Handbook [139]). Many interesting properties such as mutual exclusion, deadlock freedom, fairness, and termination can be expressed in LTL.

We will limit ourselves to synchronous systems. This limitation implies that we consider games in which the players strictly alternate turns. The theory needed for asynchronous systems is somewhat different [115, 143, 145, 155, 170], as for such systems the interleaving of processes is not under the control of (or even known to) the system. Thus, it is not possible, for instance, to guarantee that the value of an output changes before a given input changes [1, 5]. (See also [79], where realizability is used to define the concept of "receptiveness" for asynchronous systems.)

The classical approach to synthesis (presented in Sect. 27.3.4) reduces the LTL synthesis problem to the problem of synthesizing a system that realizes a language defined by a Büchi automaton. Thus, this approach can be seen as a solution to the more general problem of deriving a system from a specification given as an ω-regular language.

Every synthesis problem consists of two inputs: (1) a specification that defines the desired behavior of the system and (2) a partition of variables used in the specification into input and output variables. Let us fix a set $\mathscr{I}$ of Boolean input signals and a set $\mathscr{O}$ of Boolean output signals. Thus, the input and output alphabet are $\Sigma_{\mathscr{I}} = 2^I$ and $\Sigma_{\mathscr{O}} = 2^O$, respectively, where a letter is a subset of $\Sigma_{\mathscr{I}} \cup \Sigma_{\mathscr{O}}$ that consists of those signals that are true (cf. Chap. 2 in this Handbook [139]). In LTL synthesis the specification is an LTL formula over a set of atomic propositions $\mathscr{I} \cup \mathscr{O}$.

The synthesis problem can be described as a turn-based game between two players: the environment and the system. In each round, the system picks an output from $\Sigma_{\mathscr{O}}$ and then the environment picks an input from $\Sigma_{\mathscr{I}}$, and the next round starts. (This order corresponds to a Moore machine, we will consider Mealy machines below.)

Example 2 Figure 2(a) gives a safety automaton for the specification $\square(r \rightarrow g \vee \bigcirc g)$—every request r must be followed by a grant g in the current or in the next step. States are represented by circles and transitions by arrows. Each transition is labeled with one or more conjuncts of atomic propositions or their negations (denoted by a bar), which indicate that a transition can only be taken with a letter that satisfies one of these conjuncts. A word is accepted by this safety automaton if its run stays within the accepting states (denoted by a double circle).

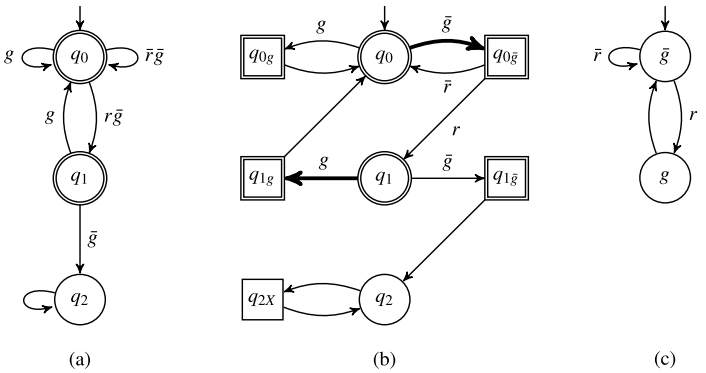


Fig. 2 A safety automaton, a labeled safety game, and a system that wins the game

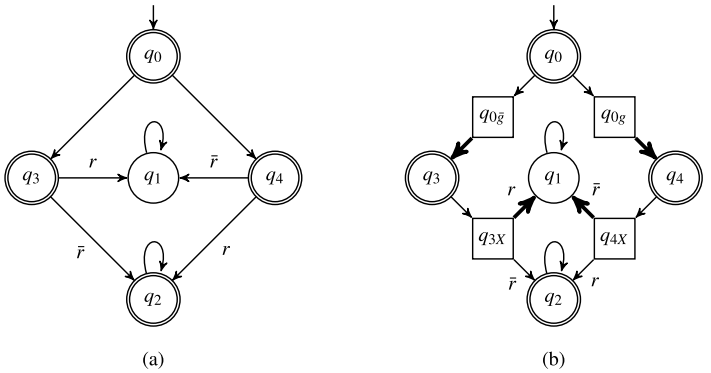


Fig. 3 A nondeterministic safety automaton, and a safety game that does NOT correspond to it

Figure 2(b) shows the safety game corresponding to this specification. The game is between the system (player 1, owner of the states depicted as circles), which controls output g, and the environment (player 2, owner of the boxes), which controls input r. The game is created by splitting every transition of the automaton into two parts: (i) one part controlled by the system and (ii) one part controlled by the environment. For instance, the self-loop on state q_0 with the label $\bar{r}\bar{g}$ in Fig. 2(a) is split into (i) the transition from state q_0 to $q_{0\bar{g}}$ in Fig. 2(b), which indicates that the system has chosen to set g to 0, and the transition from $q_{0\bar{g}}$ back to q_0, which indicates that the environment chooses to set r to 0. The winning condition for player 1 mirrors the acceptance condition of the automaton: stay within the accepting states.

Any player-1 strategy that follows the specification is winning for this game. The bold arrows in Fig. 2(b) indicate one such winning strategy. Since strategies on safety games are memoryless (see Theorem 1), a correct system can be implemented by keeping track of the state of the game and always playing the proper response, as shown in Fig. 2(c).

For the acceptance conditions that we consider, this simple transformation works as long as the specification is given by a *deterministic* automaton. For nondeterministic automata it does not work: the nondeterministic automaton in Fig. 3 accepts any word, but the game in the same figure on the right is lost for player 1. (The bold arrays indicate a winning strategy for player 2, which shows that player 1 cannot win this game from the initial state.) The need for determinization has a significant impact on complexity, as we will see later.

In order to define the LTL synthesis problem formally, we first give a formal definition of transducers, which describe the desired systems. We refer the reader to Chap. 2 in this Handbook [139] for a detailed description of LTL.

27.3.2 Games, Transducers, Trees, and Automata

In the following, we define labeled games as deterministic tree automata. The two formalisms are equivalent. We will need universal tree automata as an intermediate step between a logical specification and the resulting transducer. The relation between (nondeterministic) tree automata and games without labels is formalized in [101].

Note that the complete behavior of a transducer is a tree, where nodes are labeled with outputs and edges with inputs: the output of the transducer after input word w is the label of the node at the end of the path labeled w. Thus, a tree automaton defines a set of transducers. In the following, we will formalize this notion.

Definition 1 (Tree, Labeled Tree) Given a finite alphabet D of directions, a *D-tree* $T \subseteq D^*$ is a prefix-closed set of words over D. The nodes $v \cdot d$ for $d \in D$ are the *children* of v; v is their *parent*. The empty word ε is called the *root* of T.

A *path* ρ of T is a prefix-closed subset of T such that (1) the root is in ρ (i.e., $\varepsilon \in \rho$), and (2) every node has at most one child (i.e., $\forall v \in T\, \forall d_1, d_2 \in D$, if $d_1 \neq d_2$ and $v \cdot d_1 \in \rho$, then $v \cdot d_2 \notin \rho$). A tree T is *complete* if $T = D^*$.

A *Σ-labeled D-tree* is a pair (T, τ), where T is a tree and $\tau : T \to \Sigma$ is a *labeling function* mapping every node in T to a letter from a finite *alphabet* Σ.

Definition 2 (Transducer) A *(finite-state Moore) transducer* is a tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{O}, M, m_0, \alpha, \lambda \rangle$, where M is a (finite) set of *states*, $m_0 \in M$ is the *initial state*, $\alpha : M \times \Sigma_{\mathcal{I}} \to M$ is a *transition function* mapping a state and an input to a successor state, and $\lambda : M \to \Sigma_{\mathcal{O}}$ is a *labeling function* that maps every state to an output.

We extend α from input letters to input words in the usual way, i.e., $\alpha(m, \varepsilon) = m$ and $\alpha(m, v_0 \dots v_n) = \alpha(\alpha(m, v_0 \dots v_{n-1}), v_n)$. An *execution* of $\mathcal{M}$ on input $x_0, x_1, \dots \in \Sigma_{\mathcal{I}}$ is a word $m_0, m_1, \dots$, where $m_i = \alpha(m_{i-1}, x_{i-1})$ for all $i > 0$. The associated *I/O word* is $\lambda(m_0) \cup x_0, \lambda(m_1) \cup x_1, \dots$, and $\mathcal{L}(\mathcal{M})$ is the set of all I/O words of $\mathcal{M}$.

Every transducer $\mathscr{M} = \langle \mathscr{I}, \mathscr{O}, M, m_0, \alpha, \lambda \rangle$ generates a complete $\Sigma_{\mathscr{O}}$-labeled $\Sigma_{\mathscr{I}}$-tree (T, τ) with $\tau(\varepsilon) = \lambda(m_0)$ and $\tau(v_0 \dots v_n) = \lambda(\alpha(m_0, v_0 \dots v_{n-1}))$. We denote by $\mathscr{L}_T(\mathscr{M})$ the singleton set that contains this tree. A complete labeled tree (T, τ) is called *regular* if there exists a finite-state transducer that generates (T, τ).

Definition 3 (Deterministic Tree Automaton) A *deterministic tree automaton (on infinite labeled trees)* is a tuple $\mathscr{A} = \langle D, \Sigma, Q, q_0, \delta, \phi \rangle$, where D and Σ are a finite set of *directions* and *letters*, respectively, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to D \to Q$ is the *transition function*, and ϕ is an *acceptance condition* that specifies a subset of Q^ω.

Given a $\Sigma_{\mathscr{O}}$-labeled $\Sigma_{\mathscr{I}}$-tree (T, τ), a *run* of a deterministic tree automaton $\mathscr{A}$ on (T, τ) is an isomorphic Q-labeled tree (T, τ_r) in which (1) $\tau_r(\varepsilon) = q_0$ and (2) if $\tau_r(v) = q$ and $\tau(v) = \sigma$, then $\tau_r(v \cdot d) = \delta(q, \sigma, d)$. (The acceptance conditions correspond to the winning objectives defined in Sect. 27.2.2. We will discuss them more below.)

Example 3 The labeled game in Fig. 2(b) is formally defined by a deterministic tree automaton with $D = \{\emptyset, \{r\}\}$, $\Sigma = \{\emptyset, \{g\}\}$, $Q = \{q_0, q_1, q_2\}$, $\delta(q_0, \emptyset) = \{(\emptyset, q_0), (\{r\}, q_1)\}$,[2] $\delta(q_0, \{g\}) = \{(\emptyset, q_0), (\{r\}, q_0)\}$, $\delta(q_1, \emptyset) = \cdots$, and $\phi = \{q_0, q_1\}^\omega$. The player-1 states (circles) correspond to states of the automaton; the player-2 states (boxes) can be seen as the different transitions of the automaton, i.e., pairs of states and letters. Note that the automaton is deterministic, because in every state for every pair of letters and directions, there exists exactly one successor state.

As an intermediate step in some synthesis procedures, we will use universal tree automata. They differ from deterministic tree automata by being able to send multiple copies of the automaton, in different states, to a child of a tree node.

Definition 4 (Universal Tree Automaton) A *universal tree automaton* is a tuple $\mathscr{A} = \langle D, \Sigma, Q, q_0, \delta, \phi \rangle$, where $\delta : Q \times \Sigma \to 2^{D \times Q}$ is the *transition function*, and everything else is defined as for deterministic tree automata.

Deterministic tree automata are a special case of universal tree automata. Runs of universal automata, however, are more complicated: they are not isomorphic to the input tree. Thus, we label each node of the run tree with the node of the input tree to which it pertains.

Note that the relation between universal and deterministic automata is more complicated for infinitary acceptance conditions than in the finitary case, even for word automata. By symmetry, the same holds for the relation between nondeterministic and deterministic automata. For instance, not every Nondeterministic Büchi Word

[2] We use a set of tuples $D \times Q$ to represent a function $D \to Q$.

automaton has an equivalent Deterministic Büchi Word automaton [167] (and symmetrically, a Universal co-Büchi automaton cannot always be translated to Deterministic co-Büchi Word automaton). In other cases, the construction may be possible but very complicated, as with the determinization of parity automata [137, 151], or it may be close to the well-known subset construction (as for the translation of Universal Büchi automata to Deterministic Büchi automata) [129]. As we will see later, the complexity of the determinization procedure for parity automata is an important reason that the standard approach to synthesis is quite expensive.

Given a $\Sigma_{\mathscr{O}}$-labeled $\Sigma_{\mathscr{I}}$-tree (T, τ), a run of a universal tree automaton $\mathscr{A}$ on (T, τ) is a $T \times Q$-labeled tree (T_r, τ_r) in which (1) $\tau_r(\varepsilon) = (\varepsilon, q_0)$ and (2) for any $v \in T_r$, if $\tau_r(v) = (n, q)$, $\tau(n) = \sigma$, and $\delta(q, \sigma) = \{(d_1, q_1), \ldots, (d_n, q_n)\}$, then v has children $v_1, \ldots, v_n$ labeled $(n \cdot d_1, q_1), \ldots, (n \cdot d_n, q_n)$. Note that branches of the run tree can be finite if $\delta(q, \sigma) = \emptyset$.

Acceptance Condition. A run (T_r, τ_r) is *accepting* if all its *infinite* paths ρ satisfy the acceptance condition. The acceptance condition ϕ on paths is defined in the same way as winning objectives on plays (see Sect. 27.2.2). For example, a *Büchi condition* is given by a set $B \subseteq Q$ of target states and we define ϕ to be all the paths on which we see infinitely often a state from B, i.e., $\phi = \{q_0 q_1 \cdots \in Q^\omega \mid \forall i \geq 0 \exists j > i, q_j \in B\}$. The language $\mathscr{L}_T(\mathscr{A})$ of $\mathscr{A}$ is the set of all trees t such that the run of $\mathscr{A}$ on t is accepting. Note that finite paths can never be a reason for rejection.

If $|D| = 1$, then $\mathscr{A}$ is called a *word automaton*. In this chapter, we are not interested in nondeterministic and alternating tree automata [86, 134]. Automata types are typically denoted by three letter acronyms, where the first letter denotes the branching (A for alternating, U for universal, N for nondeterministic, or D for deterministic), the second letter describes the acceptance condition (B for Büchi, C for co-Büchi, or P for parity), and the third letter is T for tree automata or W for word automata.

Example 4 We will ignore Fig. 4 for now. Figure 5 shows a universal co-Büchi tree automaton (UCT) with letters $\Sigma = 2^{\{g_1, g_2\}}$ (two grant signals) and directions $D = 2^{\{r_1, r_2\}}$ (two request signals). Recall that a UCT accepts a tree if none of its paths visits a co-Büchi state infinitely often (cf. Sect. 27.2.2). We show part of an input tree and the corresponding run in Fig. 6. Consider the infinite path indicated with dashed bold lines Fig. 6. The sequence of directions along this path is $\{r_1 r_2\}, \emptyset, \emptyset, \{r_1 r_2\}, \emptyset, \emptyset, \ldots$, which captures the behavior in which the environment sends two requests in every third step.

This path of the input tree is labeled with the following output letter sequence $\{g_1\}, \emptyset, \{g_2\}, \{g_1\}, \emptyset, \{g_2\}$; the sequence states that the system responds to this input behavior by the following three-step pattern: first it sets g_1 to high, then it lowers both grants, and finally it sets g_2 to high.

On the right of Fig. 6 we depict the corresponding part of the run of the UCT on the input tree. Initially, the automaton is in state q_0 and it reads the label of the root of the tree (i.e., $\{g_1\}$), which enables transition t_1. From t_1 we have to move according to the direction that we consider. In our example this direction is $\{r_1 r_2\}$,

Fig. 4 NBW for $\neg\varphi$

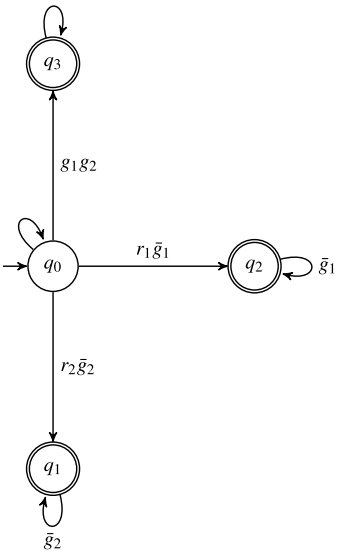
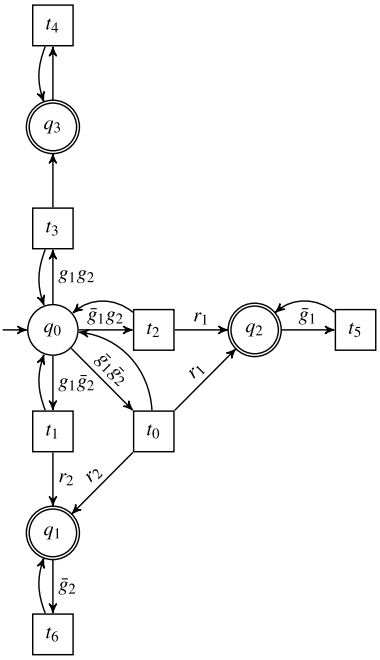


Fig. 5 UCT for φ



which enables the edge from t_1 to q_0 *and* the edge from t_1 to q_1. Since the automaton has universal branching mode, we have to follow both edges and the run continues in both states. From state q_0 with label $\emptyset$ (input tree node: 3), the automaton has to select transition t_0 and the direction $\emptyset$ leads again to q_0. From state q_1 the label $\emptyset$

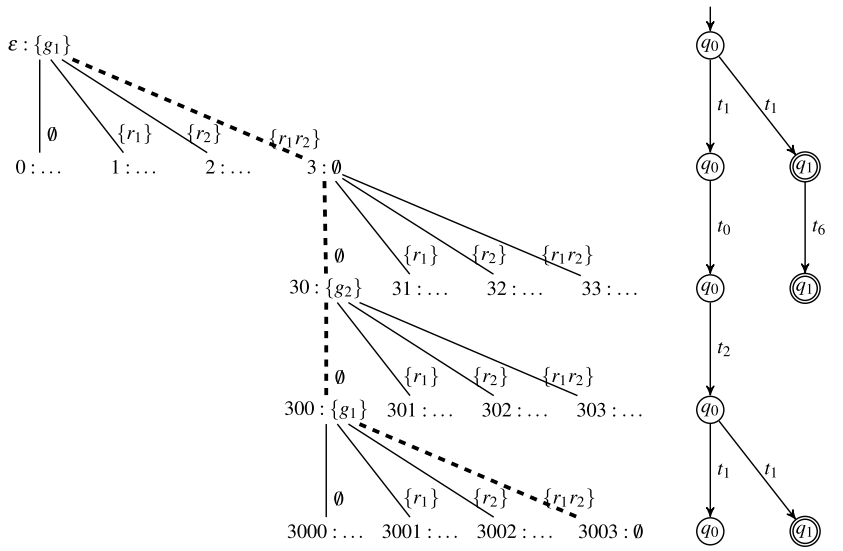


Fig. 6 (*Left*) Part of an input tree. (*Right*) UCT-run on the dashed path of the input tree

enables transition t_6, which brings us back to q_1. When the automaton is in state q_1 and it reaches node 30 labeled $\{g_2\}$ no transition is enabled, therefore this path of the run ends here. Recall that a run is accepting if none of the paths visits a co-Büchi state (indicated by a double circle) infinitely often, and a finite path is always accepting. In Fig. 6, we show only the first part of the input tree and part of the corresponding run. The depicted part of the run will repeat as the input repeats.

27.3.3 Realizability and Synthesis Problem

Given an LTL formula φ over $\mathscr{I} \cup \mathscr{O}$, and a transducer $\mathscr{M} = \langle \mathscr{I}, \mathscr{O}, M, m_0, \alpha, \lambda \rangle$, we say that $\mathscr{M}$ *realizes (or implements)* φ if $\mathscr{L}(\mathscr{M}) \subseteq \mathscr{L}(\varphi)$.

Definition 5 (LTL Realizability and Synthesis Problem) Given an LTL formula φ over the atomic propositions $\mathscr{I} \cup \mathscr{O}$, the *realizability problem* asks whether there exists a transducer $\mathscr{M}$ that realizes φ. If the answer to the realizability problem is yes, then we call the specification φ *realizable*. The *synthesis problem* is to construct $\mathscr{M}$.

Let us make two simple observations: First, constructing a Mealy machine is equally easy (or hard) as constructing a Moore machine. It can be achieved by shifting inputs by one time step. Taking LTL as an example, ϕ is Mealy-realizable iff ϕ' is Moore-realizable, where ϕ' is obtained from ϕ by replacing every occurrence of an output signal y by $\bigcirc y$. Equivalently, one can switch the order of the players in the corresponding game. Second, when negation is possible and the game is

determined (as with LTL), then ϕ is Mealy-realizable with inputs $\mathscr{I}$ and outputs $\mathscr{O}$ iff $\neg\phi$ is Moore-realizable with inputs $\mathscr{O}$ and outputs $\mathscr{I}$. In other words, there is a system that fulfills the specification iff there is no environment that guarantees violation (and vice versa).

27.3.4 Classical Approach to LTL Synthesis

In this section we summarize the classical approach to LTL synthesis [30, 142, 146]. The approach consists of the following steps:

1. Translate the LTL formula φ into a nondeterministic Büchi word automaton A.
2. Translate A into a deterministic parity word automaton (DPW) $\mathscr{B}$.
3. Construct a deterministic parity tree automaton (DPT) $\mathscr{A}_T$ from $\mathscr{B}$.
4. Check language emptiness of $\mathscr{A}_T$ (i.e., solve the parity game).
5. If $\mathscr{A}_T$ is non-empty, construct a finite-state transducer $\mathscr{M}$, otherwise report that φ is not realizable.

Example 5 (Arbiter Example) We use a specification for a simple arbiter to show the approach. The arbiter controls the access of two clients, C_1 and C_2, to a shared resource. It has two input variables r_1 and r_2 and two output variables g_1 and g_2. Client i can request the resource by setting the input variable r_i to true. The arbiter grants the resource to Client i by setting the corresponding output variable g_i to true.

We require the arbiter to ensure (i) *mutually exclusive* and (ii) *fair* access to the resource. Formally, the specification $\varphi_A = \psi \wedge \varphi_1 \wedge \varphi_2$ is the conjunction of the following three properties:

$$\begin{aligned}
\psi &= \square(\neg g_1 \vee \neg g_2) && \text{mutually exclusive access of the clients,}\\
\varphi_1 &= \square(r_1 \rightarrow \Diamond g_1) && \text{fair access for Client 1, and}\\
\varphi_2 &= \square(r_2 \rightarrow \Diamond g_2) && \text{fair access for Client 2.}
\end{aligned}$$

Step 1: LTL to NBW

Given an LTL formula φ, we first construct a nondeterministic Büchi word automaton $\mathscr{A}_\varphi$ such that $\mathscr{A}_\varphi$ accepts all the words that satisfy φ, i.e., $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A}_\varphi)$ with $|A_\varphi| = 2^{O(|\varphi|)}$ [171]. (Several people have worked on improving this translation, e.g., [88, 99, 157, 158, 165].)

Step 2: NBW to DPW

Using Piterman's determinization construction [137] (an improved version of Safra's construction [151]), we translate $\mathscr{A}$ into a deterministic Parity word automaton $\mathscr{B}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B})$. This automaton has $2^{2^{O(|\varphi|)}}$ states and $2^{O(|\varphi|)}$ priorities.

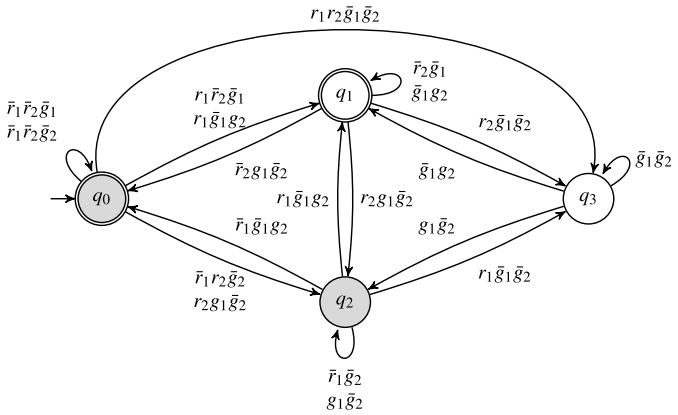


Fig. 7 Deterministic Streett word automaton for φ_A with the Streett pairs $R = \{(Q, \{q_0, q_1\}), (Q, \{q_0, q_2\})\}$

Example 6 (Arbiter Example (Cont.)) For simplicity, we show in Fig. 7 a deterministic Streett (instead of parity) word automaton for the specification φ_A. The winning condition is that sets $\{q_0, q_1\}$ and $\{q_0, q_2\}$ (marked with double circles and filled circles, respectively) must be visited infinitely often (a *generalized Büchi condition*). Formally, the automaton $\mathscr{A}_S$ has two Streett pairs $(Q, \{q_0, q_1\})$ and $(Q, \{q_0, q_2\})$. The intuitive meaning of the four states q_0, q_1, q_2 and q_3 is as follows: there are no outstanding requests in state q_0. There is an outstanding request from Client 1 (or Client 2) in state q_1 (or q_2, respectively). In state q_3 both requests are outstanding.

Step 3: DPW to DPT

It is easy to convert the DPW $\mathscr{B}$ obtained in Step 2 to a DPT $\mathscr{A}_\varphi^T$ such that $\mathscr{A}_\varphi^T$ accepts a tree iff all its paths satisfy φ. Intuitively, we split the transitions into two parts: the first part refers to the output variables (the alphabet of the tree automaton), the second part to the input variables (the directions of the tree automaton).

Example 7 (Arbiter Example (Cont.)) Figure 8 shows the tree automaton generated from the automaton in Fig. 7. The construction was described in Example 2, and will not be formalized.

Step 4: DPT Emptiness Check

The language of $\mathscr{A}_\varphi^T$ is non-empty iff it contains a $\Sigma_{\mathscr{O}}$-labeled $\Sigma_{\mathscr{I}}$-tree, i.e., iff player 1 (the system) has a winning strategy in the corresponding game. (See Example 3 for the correspondence between DPTs and games.) We can use the techniques describes in Sect. 27.2 to check whether $\mathscr{A}_\varphi^T$ (a parity game) is empty.

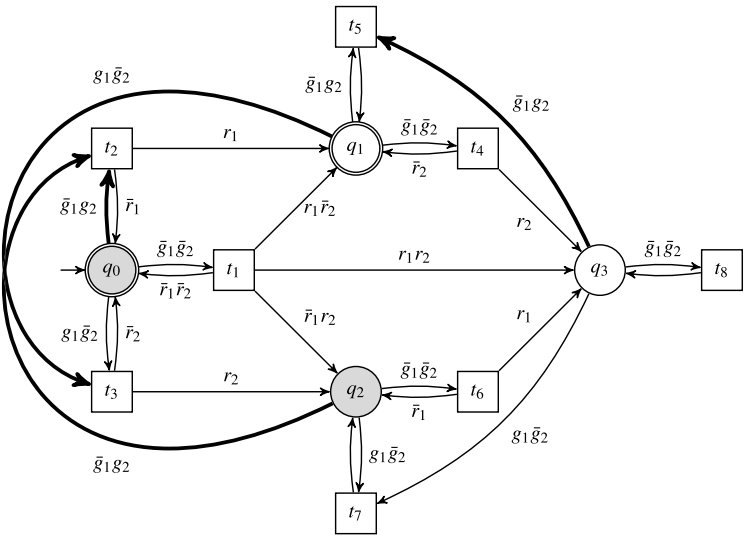


Fig. 8 Streett tree automaton for φ_A (generated from the automaton in Fig. 7). The *bold arrows* denote a winning (memoryless) strategy for player 1 in the corresponding Streett game

In general, the LTL formula is translated into a parity game with $2^{2^{O(|\varphi|)}}$ states and $2^{O(|\varphi|)}$ priorities, which can be solved in polynomial time in the number of states and exponential time in the number of priorities (see Theorem 3). A corresponding doubly exponential lower bound for LTL synthesis was shown by Rosner [150].

Step 5: Construction of Finite-State Transducer

If the language of the deterministic parity tree automaton $\mathcal{A}_\varphi^T$ is non-empty, then there exists a winning memoryless strategy for player 1 in the corresponding parity game. The strategy corresponds to a regular tree, and regular trees coincide with finite-state transducers.

Example 8 (Arbiter Example (Cont.)) The winning objective of player 1 is to visit the set $\{q_0, q_2\}$ and $\{q_0, q_1\}$ infinitely often. The bold arrows in Fig. 8 denote a memoryless winning strategy for player 1 in the corresponding game, where states $q_0, \ldots, q_3$ correspond to player-1 states of the game and the transitions $t_1, \ldots, t_8$ are the player-2 states. Note that memoryless strategies suffice for parity games. (For Streett games, which we use in the example, memoryless strategies are not sufficient, see Sect. 27.2.2.)

The strategy corresponds to the finite-state transducer shown in Fig. 9 with the three states q_0, q_1 and q_2 labeled with $\{g_2\}$, $\{g_1\bar{g}_2\}$, and $\{g_2\}$, respectively. Following the strategy, the transducer initially outputs g_2, and then moves to q_0 or q_1 depending on the input. If Client 1 sends a request (i.e., r_1 is high), then the transducer

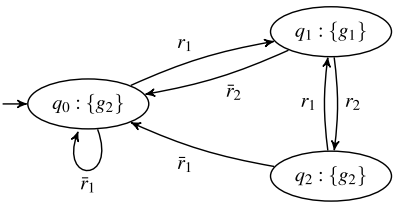


Fig. 9 Finite-state transducer implementing the specification φ_A

moves to state q_1 and sends a grant to Client 1 (i.e., it outputs g_1). Otherwise, the arbiter grants the shared resource to Client 2 by raising g_2. State q_3 is not reachable with the given strategy.

Theorem 7 ([30, 142, 146]) *Given an LTL formula φ, we can decide in $2^{2^{O(|\varphi|)}}$ time whether φ is realizable. If an LTL formula φ is realizable, then there exists a finite-state transducer with at most $2^{2^{O(|\varphi|)}}$ states that satisfies it. Both these bounds are tight.*

27.3.5 Recent Approaches to LTL Synthesis

We describe two main ideas to cope with the complexity of the LTL synthesis problem: (i) **bounding** the size of the **generated systems** and (ii) specialized procedures for **specification with restricted expressiveness**, and combinations thereof. Recent approaches are based on one or both ideas.

27.3.5.1 Bounded (or Safraless) Approaches

The idea of bounded synthesis approaches [93, 120, 156] is to restrict the size of the generated system. These algorithms are also called *Safraless*, because they avoid Safra's determinization construction. Bounded synthesis algorithms are based on the following two key insights, which were first presented by Kupferman and Vardi [120].

1. The LTL synthesis problem can be reduced to the language emptiness check of a universal co-Büchi tree automaton (UCT).
2. The language emptiness problem of UCTs can be reduced to a parametric emptiness check, where the parameter restricts the size of the trees of interest (and hence the size of the generated system).

In [120], the emptiness problem of a universal co-Büchi tree automaton with parameter k (k-UCT) is reduced to the emptiness problem of a nondeterministic Büchi tree automaton. Checking emptiness of a nondeterministic Büchi tree automaton in turn corresponds to solving a Büchi game [101]. For the purpose of this paper, we can assume that k limits the number of times that the automaton can visit a co-Büchi

state [156]. In [156], Schewe and Finkbeiner present a reduction of the k-UCT emptiness problem to emptiness of deterministic safety tree automata. The exact meaning of k is different in [120], but the general idea is the same. These bounded approaches are well suited for symbolic implementations. Schewe and Finkbeiner propose an encoding as an SMT formula, while Filiot, Jin, and Raskin [93] provide a symbolic algorithm for checking k-UCT emptiness using anti-chains. (see also [82]). Bounded approaches can also be used as a semi-decision procedure for distributed synthesis [156].

A bounded synthesis algorithm consists of the following steps:

1. Translate the LTL formula φ into a UCT $\mathscr{A}$.
2. Transform $\mathscr{A}$ into a k-UCT $\mathscr{A}_k$ for a given parameter k.
3. Check emptiness of $\mathscr{A}_k$ (solving a Büchi or Safety game).
4. If $\mathscr{A}_k$ is non-empty, construct an FSM $\mathscr{M}$, otherwise increase k and go to Step 2 or abort.

For realizable specifications the parameter k turns out to be small in practice [93, 108], leading to an efficient LTL synthesis procedure if the specification is realizable. In order to conclude that a specification is unrealizable, we must show that $\mathscr{A}_k$ is empty for a very large k (doubly exponential in the size of the formula φ, see Theorems 8 and 9). However, it follows from the remarks in Sect. 27.3.3 that in case of unrealizability there is a Mealy machine for the environment that realizes $\neg\phi$ which can be found in much the same manner as the Moore machine for the system. Again, in practice this machine can be quite small and is then found quickly.

Step 1: LTL to UCT

Given an LTL formula and a partitioning of the atomic propositions, we can construct a universal co-Büchi tree automaton that accepts all state machines that realize the formula. Intuitively, we construct an NBW for the negated formula (see Step 1 in Sect. 27.3.4). We then dualize the acceptance condition and the branching condition and transform the automaton into a tree automaton as described above.

Theorem 8 ([120], Theorem 5.1) *The realizability problem for an LTL formula can be reduced to the non-emptiness problem for a UCT with exponentially many states.*

Example 9 (Arbiter Example (Cont.)) Recall the specification of the arbiter from Example 5: $\varphi = \Box(\neg g_1 \vee \neg g_2) \wedge \Box(r_1 \rightarrow \Diamond g_1) \wedge \Box(r_2 \rightarrow \Diamond g_2)$. We can construct a UCT for φ by first constructing an NBW for the negation of the specification, i.e., $\neg\varphi = \Diamond(g_1 \wedge g_2) \vee \Diamond(r_1 \wedge \Box\neg g_1) \vee \Diamond(r_2 \wedge \Box\neg g_2)$. Note that $\neg\varphi$ simplifies to $\Diamond((g_1 \wedge g_2) \vee (r_1 \wedge \Box\neg g_1) \vee (r_2 \wedge \Box\neg g_2))$ for which we show an NBW in Fig. 4. The automaton accepts a word in one of the following cases. Each case corresponds to violating the specification in a particular way: (i) Any word that includes simultaneous grants at some point will be accepted. In this case the automaton stays in state

q_0 until $g_1 g_2$ occurs and then moves to state q_3, which is accepting and can be revisited independently of the values of the grant and request signals. (ii) A word that includes a request 1 but no subsequent grant 1 is accepting. In order to see this, note that the automaton again can stay in state q_0 until the unanswered request 1 arrives, then it moves to state q_2, where it can stay as long as it does not observe a grant 1 (i.e., as long as the request is unanswered). (iii) The situation for an outstanding request 2 is analogous. In this case the automaton will move to state q_1.

Given this NBW and a splitting of the atomic propositions into inputs r_1, r_2 and output propositions g_1, g_2, we construct the UCT shown in Fig. 5. The alphabet of the tree automaton is $2^{\{g_1,g_2\}}$; each tree has four directions corresponding to the letters in $2^{\{r_1,r_2\}}$. The dualization of the branching mode means that nondeterministic edges are now viewed as universal edges. Dualizing the Büchi condition results in a co-Büchi acceptance condition.

Step 2: UCT to k-UCT

We will present the reduction from UCT emptiness to emptiness of deterministic safety tree automata, which corresponds to finding the winning player in games with safety objectives. The reduction to Büchi games can be found in [120], Theorem 3.3. Note that in the reduction to Büchi games the parameter k is used in a different way than in the reduction presented here.

The reduction from UCT emptiness to safety games is best explained in two steps. In the first step, we reduce the UCT emptiness problem to the emptiness problem of a tree automaton with a simpler *universal k-co-Büchi* acceptance condition, which asks that every path of an accepting run visits a co-Büchi state at most k times [156]. In the second step, we show how to check whether the language of a universal k-co-Büchi tree automaton is empty by constructing an equivalent safety game and solving it.

Given a UCT $\mathscr{A}$, we write $\mathscr{A}_k$ to denote the tree automaton with the same structure as $\mathscr{A}$ and the universal k-co-Büchi acceptance condition. For any UCT $\mathscr{A}$ and any parameter k, $\mathscr{L}_T(\mathscr{A}_k) \subseteq \mathscr{L}_T(\mathscr{A})$ holds, so if the language of $\mathscr{A}_k$ is non-empty, then so is the language of $\mathscr{A}$.

For the other direction, we will use the following two lemmas. The first one shows that if the language of the UCT $\mathscr{A}$ is non-empty, then there exists a finite-state machine of bounded size that is accepted by $\mathscr{A}$. The second states that a given finite-state machine is accepted by automaton $\mathscr{A}$ if and only if it is accepted by the automaton $\mathscr{A}_k$, where $k = |\mathscr{M}| \cdot |\mathscr{A}|$.

Lemma 1 ([120], Theorem 4.3) *Given a UCT $\mathscr{A}$ with n states, if the language of $\mathscr{A}$ is not empty, i.e., $\mathscr{L}_T(\mathscr{A}) \neq \emptyset$, there exists a (non-empty) finite-state machine $\mathscr{M}$ with at most $n^{n+1} + 1$ states such that $\mathscr{L}_T(\mathscr{M}) \subseteq \mathscr{L}_T(\mathscr{A})$ (meaning that the tree generated by $\mathscr{M}$ is accepted by $\mathscr{A}$).*

Lemma 2 ([156]) *Given a finite-state machine $\mathscr{M}$ and a UCT $\mathscr{A}$, then $\mathscr{A}$ accepts $\mathscr{M}$ if and only if the k-UCT $\mathscr{A}_k$ with $k = |\mathscr{M}| \cdot |\mathscr{A}|$ accepts $\mathscr{M}$, i.e., $\mathscr{L}_T(\mathscr{M}) \subseteq \mathscr{L}_T(\mathscr{A})$ iff $\mathscr{L}_T(\mathscr{M}) \subseteq \mathscr{L}_T(\mathscr{A}_{|\mathscr{M}|\cdot|\mathscr{A}|})$ holds.*

From Lemmas 1 and 2, it follows that if the language of a UCT $\mathscr{A}$ of size n is non-empty, then so is the language of the k-UCT $\mathscr{A}_k$ with $k = (n^{n+1}+1) \cdot n$ and we can obtain the following theorem.

Theorem 9 ([156]) *For any UCT $\mathscr{A}$ of size n, there exists a k-UCT $\mathscr{A}_k$ with $k = (n^{n+1}+1) \cdot n$ such that $\mathscr{L}_T(\mathscr{A}) = \emptyset$ iff $\mathscr{L}_T(\mathscr{A}_k) = \emptyset$.*

Note that a UCT and a k-UCT differ only in the interpretation of the acceptance condition. A UCT accepts all trees that allow a run on which none of the paths visits rejecting states infinitely often. A k-UCT is more restricted; it allows at most k visits to the rejecting states. So, Fig. 5 can be seen as a UCT or as a k-UCT.

Step 3: k-UCT Emptiness Check

Given a k-UCT, we can construct a safety game with labels (i.e., a deterministic safety tree automaton) such that the safety game is winning if and only if the language of the k-UCT is not empty.

Lemma 3 ([156]) *Given a k-UCT $\mathscr{A}$ with n states, there exists a deterministic safety tree automaton $\mathscr{B}$ with $(k+2)^n$ states such that $\mathscr{L}_T(\mathscr{A}) = \mathscr{L}_T(B)$.*

This lemma can be proven as follows. Intuitively, $\mathscr{B}$ is constructed from $\mathscr{A}$ by applying an extended subset construction that keeps track of how many rejecting states (i.e., states in F) have been visited so far along all the paths ending in a state of $\mathscr{A}$. To do that, $\mathscr{B}$ has a counter for every state in $\mathscr{A}$ that counts from -1 up to $k+1$. A state of $\mathscr{B}$ is an evaluation of all the counters. Counter value -1 of a state q means that state q is not reached in the current step of the subset construction. Formally, let $\mathscr{A} = \langle 2^{\mathscr{I}}, 2^{\mathscr{O}}, Q, q_0, \delta, F\rangle$, then $\mathscr{B} = \langle 2^{\mathscr{I}}, 2^{\mathscr{O}}, S, s_0, \delta_{\mathscr{B}}, F_{\mathscr{B}}\rangle$ is given by

$$S = Q \to \{-1, 0, \ldots, k+1\}$$

$$s_0(q) = \begin{cases} 0 & \text{if } q = q_0 \\ -1 & \text{otherwise.} \end{cases}$$

$$\delta_{\mathscr{B}}(s, o) = \bigwedge_{i \in 2^{\mathscr{I}}} (i, s'), \quad \text{where}$$

$$s'(q) = \begin{cases} k+1 & \text{if } \exists p : (i,q) \in \delta(p,o) \wedge s(p) = k+1, \\ \max_{p \in Q:(i,q)\in\delta(p,o)}\{s(p)+1\} & \text{if } \exists p : (i,q) \in \delta(p,o) \wedge -1 < s(p) < k+1 \wedge \\ & \quad q \in F, \\ \max_{p \in Q:(i,q)\in\delta(p,o)}\{s(p)\} & \text{if } \exists p : (i,q) \in \delta(p,o) \wedge -1 < s(p) < k+1 \wedge \\ & \quad q \notin F, \\ -1 & \text{otherwise } (\forall p : (i,q) \notin \delta(p,o) \vee s(p) = -1). \end{cases}$$

$$F_{\mathscr{B}} = \{s \in S \mid \forall q \in Q : s(q) < k+1\}$$

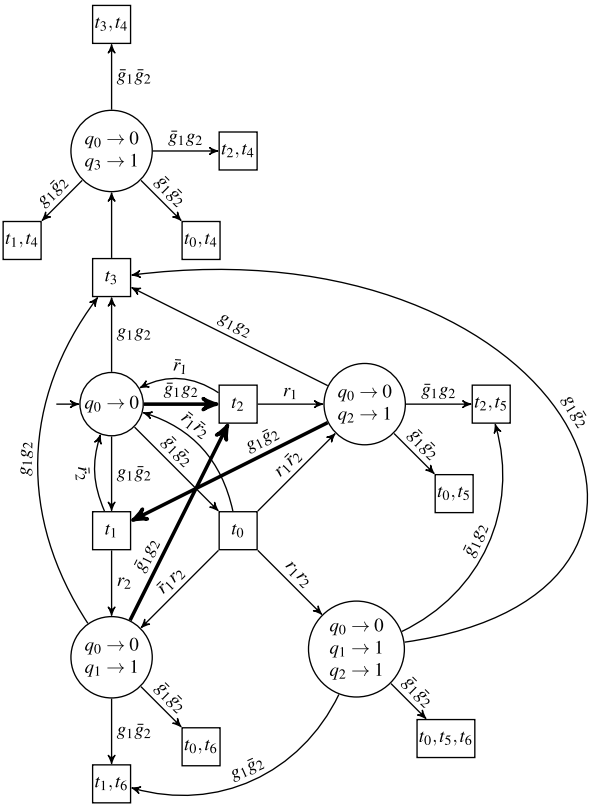


Fig. 10 Safety game (with labels) for the UCT shown in Fig. 5, with $k = 1$. The *bold arrows* represent a winning strategy from all winning states for player circle

Note that for each label $o \in 2^{\mathscr{O}}$, the automaton $\mathscr{B}$ has exactly one successor for each direction $i \in 2^{\mathscr{I}}$.

Example 10 (Arbiter Example (Cont.)) Recall the UCT from Fig. 5. Assume we fix $k = 1$, i.e., for every path we allow at most one visit to a rejecting state. In each player-1 state of the safety game we store two pieces of information: (i) the set of active states (of the UCT) in the current run and (ii) the number of rejecting states we have seen along the way. As shown after Lemma 3 we can represent this information by a function mapping from the UCT states to a number between -1 and k, i.e., the state $(q_0 \to 0, q_1 \to 2, q_2 \to -1, q_3 \to -1)$ indicates that the run is currently in state q_0 and q_1 and that on the path to q_1 we have seen two rejecting states (see the third level of the run on the right of Fig. 6). Figure 10 shows the safety game for the UCT shown in Fig. 5 with $k = 1$. In order to save space we omit UCT-states that are assigned to -1 in the description of the player-1 states. For simplicity, we label player-2 states with transitions of the UCT. In order to keep the game graph small, we also omit "unsafe" states, which are states in which a UCT-state is mapped to a number higher than k. This leads to several player-2 states without outgoing edges, called dead-end states. In all these states, player 2 wins because he could move to an

unsafe state. So, the (safety) winning condition for player 1 in this game is to avoid these dead-end states.

The game starts in state $(q_0 \to 0)$. The system (player 1) has four options corresponding to the four output letters. If she chooses the letter $\bar{g}_1 g_2$ the play continues in state t_2, from which player 2 can choose one of the four directions. If he chooses a direction that includes a request to Client 1 (r_1), then the play moves to state $(q_0 \to 0, q_2 \to 1)$. Note that on the path to state q_2, we have seen one rejecting state (namely q_2 itself), therefore q_2 is mapped to 1. Intuitively, q_2 indicates that there is an outstanding request from Client 1, so staying in q_2 forever violates the specification. Since we have chosen $k = 1$, we are only allowed to visit q_2 for one step (which corresponds to delaying the grant by one step). So, from state $(q_0 \to 0, q_2 \to 1)$ the automaton has to choose $g_1 \bar{g}_2$ in order to avoid one of the losing dead-end states. If we solve this game, we conclude that only the following five states are winning: $(q_0 \to 0)$, (t_1), (t_2), $(q_0 \to 0, q_2 \to 1)$, and $(q_0 \to 0, q_1 \to 1)$. The bold arrows in Fig. 10 show a winning strategy, which corresponds to the system shown in Fig. 9. The system sends by default a grant to Client 2. If it receives a request from Client 1, it responds to it in the next step. If it receives a request from Client 2 while sending a grant to Client 1, it will respond to it in the next step. If no request is outstanding it moves back to the default behavior (i.e., sending a grant to Client 2).

Step 4: System Construction

Once a winning strategy is found, we can construct the desired system following Step 5 of the classical approach.

27.3.5.2 Approaches for Fragments of LTL

We will now consider another approach to making synthesis more efficient by considering specification with restricted expressiveness.

The four simplest LTL fragments are (i) invariants ($\Box p$), (ii) reachability properties ($\Diamond p$), (iii) recurrence properties ($\Box \Diamond p$), and (iv) persistence properties ($\Diamond \Box p$). These fragments can be translated directly into the corresponding synthesis games: invariants translate into safety games, reachability properties into reachability games, recurrence properties into Büchi games, and persistence properties into co-Büchi games. Each of these fragments by itself is not expressive enough to specify a complete system. However, they are very useful in the context of controller synthesis [147, 148]. For example, given a system with deadlocks, we can ask for a controlled system that is deadlock-free.

Alur and La Torre [10] provide a comprehensive study of generators for deterministic automata and complexity analysis of various LTL fragments. Here, we will focus on a recent approach by Piterman, Pnueli, and Sa'ar [140] for LTL formulas in the Generalized Reactivity-1 (GR(1)) fragment, because this fragment lends itself to an efficient symbolic implementation.

Specifications in GR-(1) are of the form

$$\mathsf{env}_1 \wedge \cdots \wedge \mathsf{env}_n \rightarrow \mathsf{sys}_1 \wedge \cdots \wedge \mathsf{sys}_m,$$

where every sub-formula env_i and sys_i can be represented by a deterministic Büchi automaton.[3] The intuition is that $\mathsf{env}_1, \ldots, \mathsf{env}_n$ are formulas describing assumptions on the environment and $\mathsf{sys}_1, \ldots, \mathsf{sys}_m$ specify the desired behavior of the system if all the environment assumptions are satisfied.

The approach proceeds as follows: first, every sub-formula env_i (sys_i) is translated into a deterministic Büchi automaton. Each automaton is represented symbolically by (i) an initial predicate, (ii) a transition predicate, and (iii) a predicate describing the Büchi states. The initial and Büchi predicate refer to the atomic propositions and the set of state variables. As usual, the transition predicate may refer to the current and next values of the atomic propositions and the state variables. Then, the initial and transition predicates obtained from the different sub-formulas are conjoined to make a single initial predicate and a single transition system. On this transition system, we define the following acceptance condition using the set of Büchi predicates $\phi_1^e, \ldots, \phi_n^e$ obtained from environment assumptions and the predicates $\phi_1^s, \ldots, \phi_m^s$ obtained from the system guarantees. A path ρ through the transition system is accepting iff all system predicates ϕ_i^s are true infinitely often along the path or if some environment predicate ϕ_i^e is true only finitely often along the path. This is a generalized Streett condition with a single Streett pair (see Sect. 27.2.4). Finally, this transition system is transformed into a game by splitting the transition predicate into two parts: one part that modifies the input variables and one part that modifies the output variables. In this way, we obtain a symbolic representation of a generalized Streett-1 game, which can be solved symbolically using a triply nested fix-point (see [140] and Sect. 27.2.4).

There are several further approaches whose strength is based on a decomposition of the specification (according to the top-level Boolean structure, for instance) together with appropriate handling of the parts. Notable examples are [82, 117, 132, 160]. The GR(1) synthesis algorithm was extended to specifications in the intersection of LTL and ACTL by Ehlers [81].

[3] One way to syntactically characterize such sub-formulas is to require them to be in the set $\mathrm{LTL}^{\mathrm{det}}$ [122], which is the set of formulas defined as follows:

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid (p \wedge \varphi) \vee (\neg p \wedge \varphi) \mid (p \wedge \varphi)\,\mathsf{U}(\neg p \wedge \varphi) \mid (p \wedge \varphi)\,\mathsf{W}(\neg p \wedge \varphi),$$

where p is an arbitrary atomic proposition. Note that this set includes invariants ($\Box\, p$) and the formula $\neg p\,\mathsf{U}\, p$, which is equivalent to $\Diamond\, p$. In [140], the authors provide a different set of syntactic restrictions.

27.4 Related Topics

In this chapter we have considered perfect-information turn-based zero-sum games for synthesis from linear-time logical specifications. Perfect-information zero-sum games are used in several applications other than temporal logic synthesis.

In *controller synthesis*, we are given a nondeterministic system and we aim to restrict (*control*) the nondeterministic choices such that the controlled system satisfies the given specification. Synthesis from logical specifications and controller synthesis focus on different aspects of the synthesis problem. Research in the area of synthesis from logical specifications initially concentrated on developing new synthesis algorithms for more expressive logics, while controller synthesis focused on how to efficiently compute restrictions for a system composed of several sub-components. For an introduction to control theory of discrete-event systems we refer the reader to [31]. Discrete-event control theory has been used, for instance, to automatically avoid deadlocks in multi-threaded programs [179] or for synthesis of fault-tolerant systems [100].

Several approaches use controller synthesis to combine synthesis with imperative programming, thus avoiding the need to fully specify the system. Such approaches include program sketching [162, 163], program repair [11, 109], and synthesis of concurrent data structures [161] and synchronizations [172, 173]. It should be noted that some of these approaches use very different theory from what is presented in this chapter and that automatic programming has a much richer background than can be covered in this chapter [113, 114, 116]. See [21] for an overview of recent program synthesis techniques.

Other classes of games are also relevant in synthesis. While standard LTL synthesis reduces to perfect-information games, synthesis in the distributed setting reduces to multi-player partial-information games and distributed games [94, 118, 121, 144]. Distributed synthesis is undecidable in general but semi-decision procedures exist [156]. For the related case of parameterized systems, see [107].

The extension of synthesis to an assume-guarantee setting requires solving non-zero-sum games (namely secure equilibria) [55, 59]; and has been applied to protocol synthesis [69, 70]. Synthesis problems for resource constraints [32, 33], for performance guarantees [19], and for synthesis of robust systems [20] entail non-Boolean properties and reduce to games with quantitative objectives. Synthesis problems in probabilistic environments [12, 58, 152] or for synthesizing environment assumptions for synthesis reduce to stochastic games [57].

The games we have discussed are generalizations of finite automata. It is also possible to generalize other automata models, for instance to pushdown games [168, 177] or timed games. Timed games and controller synthesis for timed automata are discussed in Chap. 29 in this Handbook [23]. Finally, the close connection between games and verification is the subject of Chap. 26 in this Handbook [24].

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Della Rocca, S.R. (eds.) Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: The element of surprise in timed games. In: Amadio, R.M., Lugiez, D. (eds.) Intl. Conf. on Concurrency Theory (CONCUR). LNCS, vol. 2761, pp. 144–158. Springer, Heidelberg (2003)
3. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Tjoa, A.M., Gruhn, V. (eds.) Intl. Symp. on Foundations of Software Engineering (FSE), pp. 109–120. ACM, New York (2001)
4. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) Intl. Conf. on Embedded Software (EMSOFT). LNCS, vol. 2211, pp. 148–165. Springer, Heidelberg (2001)
5. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theor. Comput. Sci. **386**(3), 188–217 (2007)
6. de Alfaro, L., Henzinger, T.A., Mang, F.: Detecting errors before reaching them. In: Emerson, E.A., Sistla, A.P. (eds.) Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 1855, pp. 186–201. Springer, Heidelberg (2000)
7. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. In: Vitter, J.S., Spirakis, P.G., Yannakakis, M. (eds.) Annual ACM Symposium on Theory of Computing (STOC), pp. 675–683. ACM, New York (2001)
8. Alur, R., Bodík, R., Dallal, E., Fisman, D., Garg, P., Juniwal, G., Kress-Gazit, H., Madhusudan, P., Martin, M.M.K., Raghothaman, M., Saha, S., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Irlbeck, M., Peled, D.A., Pretschner, A. (eds.) Dependable Software Systems Engineering, pp. 1–25. IOS Press, Amsterdam (2015)
9. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**, 672–713 (2002)
10. Alur, R., Torre, S.L.: Deterministic generators and games for LTL fragments. In: Symp. on Logic in Computer Science (LICS), pp. 291–302. IEEE, Piscataway (2001)
11. Antoniotti, M.: Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system. Ph.D. thesis, New York University (1995)
12. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: IFIP Intl. Conf. on Theoretical Computer Science, pp. 493–506 (2004)
13. Beeri, C.: On the membership problem for functional and multivalued dependencies in relational databases. ACM Trans. Database Syst. **5**, 241–259 (1980)
14. Bertrand, N., Genest, B., Gimbert, H.: Qualitative determinacy and decidability of stochastic games with signals. In: Symp. on Logic in Computer Science (LICS), pp. 319–328. IEEE, Piscataway (2009)
15. Björklund, H., Sandberg, S., Vorobyov, S.: A discrete subexponential algorithms for parity games. In: Annual Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 2607, pp. 663–674. Springer, Heidelberg (2003)
16. Bjorklund, H., Sandberg, S., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. In: Intl. Symp. on Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 3153, pp. 673–685. Springer, Heidelberg (2004)
17. Blass, A., Gurevich, Y., Nachmanson, L., Veanes, M.: Play to test. In: Intl. Conf. on Formal Approaches to Software Testing (FATES). LNCS, vol. 3997. Springer, Heidelberg (2005)
18. Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T.A., Jobstmann, B.: Robustness in the presence of liveness. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 6174, pp. 410–424. Springer, Heidelberg (2010)

19. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
20. Bloem, R., Greimel, K., Henzinger, T.A., Jobstmann, B.: Synthesizing robust systems. In: Formal Methods in Computer Aided Design (FMCAD), pp. 85–92. IEEE, Piscataway (2009)
21. Bodík, R., Jobstmann, B.: Algorithmic program synthesis: introduction. Int. J. Softw. Tools Technol. Transf. **15**(5–6), 397–411 (2013)
22. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: Intl. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS). LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)
23. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Ouaknine, J., Worrell, J., Verification of real-time systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
24. Bradfield, J., Walukiewicz, I.: The mu-calculus. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
25. Brázdil, T., Brozek, V., Chatterjee, K., Forejt, V., Kucera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. In: Symp. on Logic in Computer Science (LICS), pp. 33–42. IEEE, Piscataway (2011)
26. Brázdil, T., Chatterjee, K., Forejt, V., Kucera, A.: Trading performance for stability in Markov decision processes. In: Symp. on Logic in Computer Science (LICS), pp. 331–340. IEEE, Piscataway (2013)
27. Brázdil, T., Chatterjee, K., Kucera, A., Novotný, P.: Efficient controller synthesis for consumption games with multiple resource types. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 7358, pp. 23–38. Springer, Heidelberg (2012)
28. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.F.: Faster algorithms for mean-payoff games. Form. Methods Syst. Des. **38**(2), 97–118 (2011)
29. Bryant, R.E.: Binary decision diagrams: an algorithmic basis for symbolic model checking. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
30. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. Am. Math. Soc. **138**, 295–311 (1969)
31. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer, Heidelberg (2008)
32. Cerný, P., Chatterjee, K., Henzinger, T.A., Radhakrishna, A., Singh, R.: Quantitative synthesis for concurrent programs. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 6806, pp. 243–259. Springer, Heidelberg (2011)
33. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Intl. Conf. on Embedded Software (EMSOFT). LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
34. Chatterjee, K.: Stochastic ω-regular games. Ph.D. thesis, UC Berkeley (2007)
35. Chatterjee, K.: The complexity of stochastic Müller games. Inf. Comput. **211**, 29–48 (2012)
36. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: The complexity of stochastic Rabin and Streett games. In: Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 3580, pp. 878–890. Springer, Heidelberg (2005)
37. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: The complexity of quantitative concurrent parity games. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 678–687. ACM/SIAM, New York/Philadelphia (2006)
38. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Strategy improvement in concurrent reachability games. In: Intl. Conf. on Quantitative Evaluation of Systems (QEST), pp. 291–300. IEEE, Piscataway (2006)
39. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Qualitative concurrent parity games. ACM Trans. Comput. Log. **12**(4), 28 (2011)

40. Chatterjee, K., Chmelik, M., Tracol, M.: What is decidable about partially observable Markov decision processes with omega-regular objectives. In: Annual Conf. on Computer Science Logic (CSL). LIPIcs, vol. 23 (2013)
41. Chatterjee, K., Doyen, L.: The complexity of partial-observation parity games. In: Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). LNCS, vol. 6397, pp. 1–14. Springer, Heidelberg (2010)
42. Chatterjee, K., Doyen, L.: Energy parity games. Theor. Comput. Sci. **458**, 49–60 (2012)
43. Chatterjee, K., Doyen, L.: Partial-observation stochastic games: how to win when belief fails. In: Symp. on Logic in Computer Science (LICS), pp. 175–184. IEEE, Piscataway (2012)
44. Chatterjee, K., Doyen, L., Gimbert, H., Henzinger, T.A.: Randomness for free. In: Intl. Symp. on Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 6281, pp. 246–257. Springer, Heidelberg (2010)
45. Chatterjee, K., Doyen, L., Henzinger, T., Raskin, J.F.: Generalized mean-payoff and energy games. In: Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LIPIcs, vol. 8, pp. 505–516. Schloss Dagstuhl—LZI, Dagstuhl (2010)
46. Chatterjee, K., Doyen, L., Henzinger, T.A.: Qualitative analysis of partially-observable Markov decision processes. In: Intl. Symp. on Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 6281, pp. 258–269. Springer, Heidelberg (2010)
47. Chatterjee, K., Doyen, L., Henzinger, T.A.: A survey of partial-observation stochastic parity games. Form. Methods Syst. Des. **43**(2), 268–284 (2013)
48. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.: Algorithms for omega-regular games with imperfect information. In: Annual Conf. on Computer Science Logic (CSL). LNCS, vol. 4207, pp. 287–302. Springer, Heidelberg (2006)
49. Chatterjee, K., Doyen, L., Nain, S., Vardi, M.Y.: The complexity of partial-observation stochastic parity games with finite-memory strategies. In: Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS). LNCS, vol. 8412. Springer, Heidelberg (2014)
50. Chatterjee, K., Forejt, V., Wojtczak, D.: Multi-objective discounted reward verification in graphs and MDPs. In: Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). LNCS, vol. 8312. Springer, Heidelberg (2013)
51. Chatterjee, K., Goyal, P., Ibsen-Jensen, R., Pavlogiannis, A.: Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In: Symp. on Principles of Programming Languages (POPL). ACM, New York (2015)
52. Chatterjee, K., Henzinger, M.: An $O(n^2)$ time algorithm for alternating Büchi games. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1386–1399 (2012)
53. Chatterjee, K., Henzinger, M.: Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition. J. ACM **61**(3), 15:1–15:40 (2014)
54. Chatterjee, K., Henzinger, T.A.: Semiperfect-information games. In: Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS, vol. 3821, pp. 1–18. Springer, Heidelberg (2005)
55. Chatterjee, K., Henzinger, T.A.: Assume-guarantee synthesis. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 4424, pp. 261–275. Springer, Heidelberg (2007)
56. Chatterjee, K., Henzinger, T.A.: A survey of stochastic omega-regular games. J. Comput. Syst. Sci. **78**(2), 394–413 (2012)
57. Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Environment assumptions for synthesis. In: Intl. Conf. on Concurrency Theory (CONCUR). CONCUR, vol. 2008, pp. 147–161. Springer, Heidelberg (2008)
58. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Singh, R.: Measuring and synthesizing systems in probabilistic environments. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 6174, pp. 380–395. Springer, Heidelberg (2010)
59. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. In: Symp. on Logic in Computer Science (LICS), pp. 160–169. IEEE, Piscataway (2004)

60. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Mean-payoff parity games. In: Symp. on Logic in Computer Science (LICS), pp. 178–187 (2005)
61. Chatterjee, K., Henzinger, T.A., Piterman, N.: Algorithms for Büchi games. In: Workshop on Games in Design and Verification (GDV) (2006)
62. Chatterjee, K., Henzinger, T.A., Piterman, N.: Generalized parity games. In: Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS). LNCS, vol. 4423, pp. 153–167. Springer, Heidelberg (2007)
63. Chatterjee, K., Henzinger, T.A., Piterman, N.: Strategy logic. Inf. Comput. **208**(6), 677–693 (2010)
64. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Annual Conf. on Computer Science Logic (CSL). LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
65. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 121–130. SIAM, Philadelphia (2004)
66. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Markov decision processes with multiple objectives. In: Annual Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 3884, pp. 325–336. Springer, Heidelberg (2006)
67. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Stochastic limit-average games are in EXPTIME. Int. J. Game Theory **37**(2), 219–234 (2008)
68. Chatterjee, K., Pavlogiannis, A., Velner, Y.: Quantitative interprocedural analysis. In: Symp. on Principles of Programming Languages (POPL). ACM, New York (2015)
69. Chatterjee, K., Raman, V.: Synthesizing protocols for digital contract signing. In: Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI). LNCS, vol. 7148, pp. 152–168. Springer, Heidelberg (2012)
70. Chatterjee, K., Raman, V.: Assume-guarantee synthesis for digital contract signing. Form. Asp. Comput. **26**(4), 825–859 (2014)
71. Chatterjee, K., Randour, M., Raskin, J.F.: Strategy synthesis for multi-dimensional quantitative objectives. In: Intl. Conf. on Concurrency Theory (CONCUR). LNCS, vol. 7454, pp. 115–131. Springer, Heidelberg (2012)
72. Chatterjee, K., Velner, Y.: Mean-payoff pushdown games. In: Symp. on Logic in Computer Science (LICS), pp. 195–204. IEEE, Piscataway (2012)
73. Chatterjee, K., Velner, Y.: Hyperplane separation technique for multidimensional mean-payoff games. In: Intl. Conf. on Concurrency Theory (CONCUR). LNCS, vol. 8052, pp. 500–515. Springer, Heidelberg (2013)
74. Chen, T., Forejt, V., Kwiatkowska, M.Z., Simaitis, A., Wiltsche, C.: On stochastic games with multiple objectives. In: Intl. Symp. on Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 8087, pp. 266–277. Springer, Heidelberg (2013)
75. Church, A.: Logic, arithmetic, and automata. In: Proceedings of the International Congress of Mathematicians, pp. 23–35. Institut Mittag-Leffler, Djursholm (1962)
76. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1981)
77. Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992)
78. Condon, A.: On algorithms for simple stochastic games. In: Advances in Computational Complexity Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, pp. 51–73. AMS, Providence (1993)
79. Dill, D.L.: Trace theory for automatic hierarchical verification of speed-independent circuits. Ph.D. thesis, ACM Distinguished Dissertation Series. MIT Press (1989)
80. Dziembowski, S., Jurdziński, M., Walukiewicz, I.: How much memory is needed to win infinite games? In: Symp. on Logic in Computer Science (LICS), pp. 99–110. IEEE, Piscataway (1997)
81. Ehlers, R.: ACTL ∩ LTL synthesis. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 7358, pp. 39–54. Springer, Heidelberg (2012)

82. Ehlers, R.: Symbolic bounded synthesis. Form. Methods Syst. Des. **40**(2), 232–262 (2012)
83. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. Int. J. Game Theory **8**(2), 109–113 (1979)
84. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
85. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 328–337. IEEE, Piscataway (1988)
86. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 368–377. IEEE, Piscataway (1991)
87. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Log. Methods Comput. Sci. **4**(4) (2008)
88. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005)
89. Etessami, K., Yannakakis, M.: Recursive Markov decision processes and recursive stochastic games. In: Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 3580, pp. 891–903. Springer, Heidelberg (2005)
90. Etessami, K., Yannakakis, M.: Recursive concurrent stochastic games. In: Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 4052, pp. 324–335. Springer, Heidelberg (2006)
91. Etessami, K., Yannakakis, M.: On the complexity of Nash equilibria and other fixed points. SIAM J. Comput. **39**(6), 2531–2597 (2010)
92. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
93. Filiot, E., Jin, N., Raskin, J.F.: An antichain algorithm for LTL realizability. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
94. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Symp. on Logic in Computer Science (LICS), pp. 321–330. IEEE, Piscataway (2005)
95. Finkbeiner, B., Schewe, S.: Coordination logic. In: Annual Conf. on Computer Science Logic (CSL). LNCS, vol. 6247, pp. 305–319. Springer, Heidelberg (2010)
96. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 6065, pp. 112–127. Springer, Heidelberg (2011)
97. Friedmann, O.: An exponential lower bound for the parity game strategy improvement algorithm as we know it. In: Symp. on Logic in Computer Science (LICS), pp. 145–156. IEEE, Piscataway (2009)
98. Friedmann, O.: Exponential lower bounds for solving infinitary payoff games and linear programs. Ph.D. thesis, University of Munich (2011)
99. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001)
100. Girault, A., Rutten, É.: Automating the addition of fault tolerance with discrete controller synthesis. Form. Methods Syst. Des. **35**(2), 190–225 (2009)
101. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Annual ACM Symposium on Theory of Computing (STOC), pp. 60–65. ACM, New York (1982)
102. Hansen, K.A., Koucký, M., Lauritzen, N., Miltersen, P.B., Tsigaridas, E.P.: Exact algorithms for solving stochastic games: extended abstract. In: Annual ACM Symposium on Theory of Computing (STOC), pp. 205–214 (2011)
103. Henzinger, T.A., Krishnan, S., Kupferman, O., Mang, F.: Synthesis of uninitialized systems. In: Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 2380, pp. 644–656. Springer, Heidelberg (2002)
104. Henzinger, T.A., Kupferman, O., Rajamani, S.: Fair simulation. Inf. Comput. **173**, 64–81 (2002)

105. Horn, F.: Streett games on finite graphs. In: Workshop on Games in Design and Verification (GDV) (2005)
106. Immerman, N.: Number of quantifiers is better than number of tape cells. J. Comput. Syst. Sci. **22**, 384–406 (1981)
107. Jacobs, S., Bloem, R.: Parameterized synthesis. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 7214, pp. 362–376. Springer, Heidelberg (2012)
108. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Formal Methods in Computer Aided Design (FMCAD), pp. 117–124. IEEE, Piscataway (2006)
109. Jobstmann, B., Staber, S., Griesmayer, A., Bloem, R.: Finding and fixing faults. J. Comput. Syst. Sci. **78**(2), 441–460 (2012)
110. Jurdziński, M.: Deciding the winner in parity games is in UP ∩ co-UP. Inf. Process. Lett. **68**(3), 119–124 (1998)
111. Jurdziński, M.: Small progress measures for solving parity games. In: Annual Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
112. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 117–123. ACM/SIAM, New York/Philadelphia (2006)
113. Katz, G., Peled, D.: Synthesizing solutions to the leader election problem using model checking and genetic programming. In: Intl. Haifa Verification Conference (HVC). LNCS, vol. 6405, pp. 117–132. Springer, Heidelberg (2009)
114. Kitzelmann, E.: Inductive programming: a survey of program synthesis techniques. In: Intl. Workshop on Approaches and Applications of Inductive Programming (AAIP), pp. 50–73 (2009)
115. Klein, U., Piterman, N., Pnueli, A.: Effective synthesis of asynchronous systems from GR(1) specifications. In: Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI). LNCS, vol. 7148, pp. 283–298. Springer, Heidelberg (2012)
116. Kuncak, V., Mayer, M., Piskac, R., Suter, P.: Complete functional synthesis. In: Conf. on Programming Language Design and Implementation (PLDI), pp. 316–329. ACM, New York (2010)
117. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
118. Kupferman, O., Vardi, M.: Synthesis with incomplete information. In: 2nd International Conference on Temporal Logic, Manchester, pp. 91–106 (1997)
119. Kupferman, O., Vardi, M.: Weak alternating automata and tree automata emptiness. In: Annual ACM Symposium on Theory of Computing (STOC), pp. 224–233. ACM, New York (1998)
120. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 531–542. IEEE, Piscataway (2005)
121. Madusudan, P.: Control and synthesis of open reactive systems. Ph.D. thesis, Theoretical Computer Science Group, Institute of Mathematical Sciences, University of Madras (2001)
122. Maidl, M.: The common fragment of CTL and LTL. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 643–652. IEEE, Piscataway (2000)
123. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems. In: Annual Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
124. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
125. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems. Springer, Heidelberg (1995)
126. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. Trans. Program. Lang. Syst. **6**(1), 68–93 (1984)

127. Martin, D.: The determinacy of Blackwell games. J. Symb. Log. **63**(4), 1565–1581 (1998)
128. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Log. **65**, 149–184 (1993)
129. Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. **32**, 321–330 (1984)
130. Mogavero, F., Murano, A., Sauro, L.: On the boundary of behavioral strategies. In: Symp. on Logic in Computer Science (LICS), pp. 263–272 (2013)
131. Mogavero, F., Murano, A., Vardi, M.Y.: Reasoning about strategies. In: Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LIPIcs, vol. 8, pp. 133–144 (2010)
132. Morgenstern, A.: Symbolic controller synthesis for LTL specifications. Ph.D. thesis, TU Kaiserslautern (2010)
133. Mostowski, A.: Regular expressions for infinite trees and a standard form of automata. In: Symposium on Computation Theory. LNCS, vol. 208, pp. 157–168. Springer, Heidelberg (1984)
134. Muller, D.E., Schupp, P.E.: Alternating automata on infinite objects, determinacy and Rabin's theorem. In: Automata on Infinite Words. LNCS, vol. 192, pp. 100–107. Springer, Heidelberg (1984)
135. Nain, S., Vardi, M.Y.: Solving partial-information stochastic parity games. In: Symp. on Logic in Computer Science (LICS), pp. 341–348. IEEE, Piscataway (2013)
136. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Math. Oper. Res. **12**, 441–450 (1987)
137. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Symp. on Logic in Computer Science (LICS), pp. 255–264. IEEE, Seattle (2006)
138. Piterman, N., Pnueli, A.: Faster solution of Rabin and Streett games. In: Symp. on Logic in Computer Science (LICS), pp. 275–284. IEEE, Piscataway (2006)
139. Piterman, N., Pnueli, A.: Temporal logic and fair discrete systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking. Springer, Heidelberg (2018)
140. Piterman, N., Pnueli, A., Sa´ar, Y.: Synthesis of reactive(1) designs. In: Intl. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI). LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2006)
141. Pnueli, A.: The temporal logic of programs. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 46–57. IEEE, Piscataway (1977)
142. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Symp. on Principles of Programming Languages (POPL), pp. 179–190. ACM, New York (1989)
143. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Intl. Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 372, pp. 652–671. Springer, Heidelberg (1989)
144. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 746–757. IEEE, Piscataway (1990)
145. Puchala, B.: Asynchronous omega-regular games with partial information. In: Intl. Symp. on Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 6281, pp. 592–603. Springer, Heidelberg (2010)
146. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Am. Math. Soc. **141**, 1–35 (1969)
147. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**, 206–230 (1987)
148. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE **77**, 81–98 (1989)
149. Reif, J.H.: Universal games of incomplete information. In: Annual ACM Symposium on Theory of Computing (STOC), pp. 288–308. ACM, New York (1979)
150. Rosner, R.: Modular synthesis of reactive systems. Ph.D. thesis, Weizmann Institute of Science (1992)

151. Safra, S.: On the complexity of ω-automata. In: Annual Symp. on Foundations of Computer Science (FOCS), pp. 319–327. IEEE, Piscataway (1988)
152. Schewe, S.: Synthesis for probabilistic environments. In: Intl. Symp. Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 4218, pp. 245–259. Springer, Heidelberg (2006)
153. Schewe, S.: Solving parity games in big steps. In: Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). LNCS, vol. 4855, pp. 449–460. Springer, Heidelberg (2007)
154. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Annual Conf. on Computer Science Logic (CSL). LNCS, vol. 5213, pp. 369–384. Springer, Heidelberg (2008)
155. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: Intl. Sym. on Logic-Based Program Synthesis and Transformation, 16th International (LOPSTR). LNCS, vol. 4407, pp. 127–142. Springer, Heidelberg (2006)
156. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Intl. Symp. Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)
157. Schneider, K.: Improving automata generation for linear temporal logic by considering the automaton hierarchy. In: Intl. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). LNCS, vol. 2250, pp. 39–54. Springer, Heidelberg (2001)
158. Sebastiani, R., Tonetta, S.: "More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In: Correct Hardware Design and Verification Methods (CHARME). LNCS, vol. 2860, pp. 126–140. Springer, Heidelberg (2003)
159. Shapley, L.: Stochastic games. Proc. Natl. Acad. Sci. USA **39**, 1095–1100 (1953)
160. Sohail, S., Somenzi, F.: Safety first: a two-stage algorithm for LTL games. In: Formal Methods in Computer Aided Design (FMCAD), pp. 77–84. IEEE, Piscataway (2009)
161. Solar-Lezama, A., Jones, C.G., Bodík, R.: Sketching concurrent data structures. In: Conf. on Programming Language Design and Implementation (PLDI), pp. 136–148. ACM, New York (2008)
162. Solar-Lezama, A., Rabbah, R.M., Bodík, R., Ebcioglu, K.: Programming by sketching for bit-streaming programs. In: Conf. on Programming Language Design and Implementation (PLDI), pp. 281–294. ACM, New York (2005)
163. Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S.A., Saraswat, V.A.: Combinatorial sketching for finite programs. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 404–415. ACM, New York (2006)
164. Somenzi, F.: Colorado University Decision Diagram Package (1998). http://vlsi.colorado.edu/~fabio/CUDD/
165. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)
166. Stockmeyer, L.: The complexity of decision problems in automata theory and logic. Ph.D. thesis, Massachusetts Institute of Technology (1974)
167. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 133–191. Elsevier, Amsterdam (1990)
168. Thomas, W.: On the synthesis of strategies in infinite games. In: Annual Symposium on Theoretical Aspects of Computer Science (STACS). LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
169. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Heidelberg (1997). Beyond Words, Chap. 7
170. Vardi, M.Y.: An automata-theoretic approach to fair realizability and synthesis. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 939, pp. 267–278. Springer, Heidelberg (1995)

171. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. **115**(1), 1–37 (1994)
172. Vechev, M.T., Yahav, E., Yorsh, G.: Inferring synchronization under limited observability. In: Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 5505, pp. 139–154. Springer, Heidelberg (2009)
173. Vechev, M.T., Yahav, E., Yorsh, G.: Abstraction-guided synthesis of synchronization. In: Symp. on Principles of Programming Languages (POPL), pp. 327–338. ACM, New York (2010)
174. Velner, Y., Rabinovich, A.: Church synthesis problem for noisy input. In: Intl. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS). LNCS, vol. 6604, pp. 275–289. Springer, Heidelberg (2011)
175. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
176. Walukiewicz, I.: Pushdown processes: games and model checking. In: Intl. Conf. on Computer-Aided Verification (CAV). LNCS, vol. 1102, pp. 62–74. Springer, Heidelberg (1996)
177. Walukiewicz, I.: Pushdown processes: games and model-checking. Inf. Comput. **164**(2), 234–263 (2001)
178. Walukiewicz, I.: A landscape with games in the background. In: Symp. on Logic in Computer Science (LICS), pp. 356–366. IEEE, Piscataway (2004)
179. Wang, Y., Lafortune, S., Kelly, T., Kudlur, M., Mahlke, S.A.: The theory of deadlock avoidance via discrete control. In: Symp. on Principles of Programming Languages (POPL), pp. 252–263. ACM, New York (2009)
180. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1–2), 135–183 (1998)
181. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. **158**, 343–359 (1996)