Synthesizing Dominant Strategies for Liveness

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Noemi Passing
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Reactive synthesis automatically derives a strategy that satisfies a given specification. However, requiring a strategy to meet the specification in every situation is, in many cases, too hard of a requirement. Particularly in compositional synthesis of distributed systems, individual winning strategies for the processes often do not exist. Remorsefree dominance, a weaker notion than winning, accounts for such situations: dominant strategies are only required to be as good as any alternative strategy, i.e., they are allowed to violate the specification if no other strategy would have satisfied it in the same situation. The composition of dominant strategies is only guaranteed to be dominant for safety properties, though; preventing the use of dominance in compositional synthesis for liveness specifications. Yet, safety properties are often not expressive enough. In this paper, we thus introduce a new winning condition for strategies, called delay-dominance, that overcomes this weakness of remorsefree dominance: we show that it is compositional for both safety and liveness specifications, enabling a compositional synthesis algorithm based on delay-dominance for general specifications. Furthermore, we introduce an automaton construction for recognizing delay-dominant strategies and prove its soundness and completeness. The resulting automaton is of single-exponential size in the squared length of the specification and can immediately be used for safraless synthesis procedures. Thus, synthesis of delay-dominant strategies is, as synthesis of winning strategies, in 2EXPTIME.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Dominant Strategies, Compositional Synthesis, Reactive Synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.15

Related Version *Full Version*: `https://arxiv.org/abs/2210.01660` [19]

Funding Supported by DFG grant 389792660 as part of TRR 248 (CPEC) and by ERC grant 683300 (OSARES). N. Passing carried out this work as PhD candidate at Saarland University, Germany.

1 Introduction

Reactive synthesis is the task of automatically deriving a strategy that satisfies a formal specification, e.g., given in LTL [31], in *every* situation. Such strategies are called winning. In many cases, however, requiring the strategy to satisfy the specification in every situation is too hard of a requirement. A prominent example is the compositional synthesis of distributed systems consisting of several processes. Compositional approaches for distributed synthesis [27, 13, 14, 15, 18] break down the synthesis task for the whole system into several smaller ones for the individual processes. This is necessary due to the general undecidability [33] of distributed synthesis and the non-elementary complexity [20] for decidable cases: non-compositional distributed synthesis approaches [22, 21] suffer from a severe state space explosion problem and are thus not feasible for larger systems. However, winning strategies rarely exist when considering the processes individually in the smaller subtasks of compositional synthesis since usually the processes need to collaborate in order to achieve the overall system's correctness. For instance, a particular input sequence may prevent the satisfaction of the specification no matter how a single process reacts, yet, the other processes of the system ensure in the interplay of the whole system that this input sequence will never be produced.

Remorsefree dominance [9], a weaker notion than winning, accounts for such situations. A dominant strategy is allowed to violate the specification as long as no other strategy would have satisfied it in the same situation. Hence, a dominant strategy is a best-effort strategy as we do not blame it for violating the specification if the violation is not its fault. Searching for dominant rather than winning strategies allows us to find strategies that do not necessarily satisfy the specification in all situations but in all that are *realistic* in the sense that they occur in the interplay of the processes if all of them play best-effort strategies.

The parallel composition of dominant strategies, however, is only guaranteed to be dominant for safety properties [10]. For liveness specifications, in contrast, dominance is not a compositional notion and thus not suitable for compositional synthesis. Consider, for example, a system with two processes p_1 and p_2 sending messages to each other, denoted by atomic propositions m_1 and m_2, respectively. Both processes are required to send their message eventually, i.e., $\varphi = \diamondsuit m_1 \land \diamondsuit m_2$. For p_i, it is dominant to wait for the other process to send the message m_{3-i} before sending its own message m_i: if p_{3-i} sends its message eventually, p_i does so as well, satisfying φ. If p_{3-i} never sends its message, φ is violated, no matter how p_i reacts, and thus the violation of φ is not p_i's fault. Combining these strategies for p_1 and p_2, however, yields a system that never sends any message since both processes wait indefinitely on each other, while there clearly exist strategies for the whole system that satisfy φ.

Bounded dominance [10] is a variant of remorsefree dominance that ensures compositionality of general properties. Intuitively, it reduces every specification φ to a safety property by introducing a measure of the strategy's progress with respect to φ, and by bounding the number of non-progress steps, i.e., steps in which no progress is made. Yet, bounded dominance has two major disadvantages: (i) it requires a concrete bound on the number of non-progress steps, and (ii) not every bounded dominant strategy is dominant: if the bound n is chosen too small, every strategy, also a non-dominant one, is trivially n-dominant.

In this paper, we introduce a new winning condition for strategies, called *delay-dominance*, that builds upon the ideas of bounded dominance but circumvents the aforementioned weaknesses. Similar to bounded dominance, it introduces a progress measure on strategies. However, it does not require a concrete bound on the number of non-progress steps but relates such steps in the potentially delay-dominant strategy s to non-progress steps in an alternative strategy t: intuitively, s delay-dominates t if, whenever s makes a non-progress step, t makes a non-progress step *eventually* as well. A strategy s is then delay-dominant if it delay-dominates all other strategies t. In this way, we ensure that a delay-dominant strategy satisfies the specification "faster" than all other strategies in all situations in which the specification can be satisfied. Delay-dominance considers specifications given as alternating co-Büchi automata. Non-progress steps with respect to the automaton are those that enforce a visit of a rejecting state in all run trees. We introduce a two-player game, the so-called *delay-dominance game*, which is vaguely leaned on the delayed simulation game for alternating Büchi automata [24], to formally define delay-dominance: the winner of the game determines whether or not a strategy s delay-dominates a strategy t on a given input sequence.

We show that (i) every delay-dominant strategy is also remorsefree dominant, and (ii) delay-dominance is compositional for both safety and liveness properties, thus overcoming the weaknesses of both remorsefree and bounded dominance. Note that since delay-dominance relies, as bounded dominance, on the automaton structure, there are realizable specifications for which no delay-dominant strategy exists. Yet, we experienced that this rarely occurs in practice when constructing the automaton from an LTL formula with standard algorithms. Moreover, if a delay-dominant strategy exists, it is guaranteed to be winning if the specification is realizable. Hence, the parallel composition of delay-dominant strategies for all processes in a distributed system is winning for the whole system as long as the specification is realizable. Therefore, delay-dominance is a suitable notion for compositional synthesis.

We thus introduce a synthesis approach for delay-dominant strategies that immediately enables a compositional synthesis algorithm for distributed systems, namely synthesizing delay-dominant strategies for the processes separately. We present the construction of a universal co-Büchi automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ from an LTL formula φ that recognizes delay-dominant strategies. $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ can immediately be used for safraless synthesis [28] approaches such as bounded synthesis [22] to synthesize delay-dominant strategies. We show that the size of $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ is single-exponential in the squared length of φ. Thus, synthesis of delay-dominant strategies is, similar to synthesis of winning or remorsefree dominant strategies, in 2EXPTIME.

Related Work. *Remorsefree dominance* has first been introduced for reactive synthesis in [9]. Dominant strategies have been utilized for compositional synthesis of safety properties [10]. Building up on this work, a compositional synthesis algorithm, that finds solutions in more cases by incrementally synthesizing individual dominant strategies, has been developed [16]. Both algorithms suffer from the non-compositionality of dominant strategies for liveness properties. *Bounded dominance* [10], a variant of dominance that introduces a bound on the number of steps in which a strategy does not make progress with respect to the specification, solves this problem. However, it requires a concrete bound on the number of non-progress steps. Moreover, a bounded dominant strategy is not necessarily dominant.

Good-enough synthesis [1, 29] follows a similar idea as dominance. It is thus not compositional for liveness properties either. In good-enough synthesis, conjuncts of the specification can be marked as *strong*. If the specification is unrealizable, a good-enough strategy needs to satisfy the strong conjuncts while it may violate the other ones. Thus, dominance can be seen as the special case of good-enough synthesis in which no conjuncts are marked as strong. Good-enough synthesis can be extended to a multi-valued correctness notion [1].

Synthesis under environment assumptions is a well-studied problem that also aims at relaxing the requirements on a strategy. There, explicit assumptions on the environment are added to the specification. These assumptions can be LTL formulas restricting the possible input sequences (see, e.g., [7, 5]) or environment strategies (see, e.g., [2, 3, 17, 18]). The assumptions can also be conceptual such as assuming that the environment is rational (see, e.g., [23, 26, 6, 8]). Synthesis under environment assumptions is orthogonal to the synthesis of dominant strategies and good-enough synthesis since it requires an explicit assumption on the environment, while the latter two approaches rely on implicit assumptions.

2 Preliminaries

Notation. Given an infinite word $\sigma = \sigma_0\sigma_1 \ldots \in (2^\Sigma)^\omega$, we denote the prefix of length $t+1$ of σ with $\sigma_{|t} := \sigma_0 \ldots \sigma_t$. For σ and a set $X \subseteq \Sigma$, let $\sigma \cap X := (\sigma_0 \cap X)(\sigma_1 \cap X) \ldots \in (2^X)^\omega$. For $\sigma \in (2^\Sigma)^\omega$, $\sigma' \in (2^{\Sigma'})^\omega$ with $\Sigma \cap \Sigma' = \emptyset$, we define $\sigma \cup \sigma' := (\sigma_0 \cup \sigma'_0)(\sigma_1 \cup \sigma'_1) \ldots \in (2^{\Sigma \cup \Sigma'})^\omega$. For a k-tuple a, we denote the j-th component of a with $j(a)$. We represent a Boolean formula $\bigvee_i \bigwedge_j c_{i,j}$ in disjunctive normal form (DNF) also in its set notation $\bigcup_i \{\bigcup_j \{c_{i,j}\}\}$.

LTL. Linear-time temporal logic (LTL) [31] is a standard specification language for linear-time properties. Let Σ be a finite set of atomic propositions and let $a \in \Sigma$. The syntax of LTL is given by $\varphi, \psi ::= a \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc \varphi \mid \varphi \mathcal{U} \psi$. We define $true = a \vee \neg a$, $false = \neg true$, $\Diamond \varphi = true \mathcal{U} \varphi$, and $\Box \varphi = \neg \Diamond \neg \varphi$ as usual. We use the standard semantics. The language $\mathcal{L}(\varphi)$ of an LTL formula φ is the set of infinite words that satisfy φ.

Non-Alternating ω-Automata. Given a finite alphabet Σ, a Büchi (resp. co-Büchi) automaton $\mathcal{A} = (Q, Q_0, \delta, F)$ over Σ consists of a finite set of states Q, an initial state $q_0 \in Q$, a transition relation $\delta : Q \times 2^\Sigma \times Q$, and a set of accepting (resp. rejecting) states $F \subseteq Q$. For an infinite word $\sigma = \sigma_0\sigma_1 \ldots \in (2^\Sigma)^\omega$, a run of $\mathcal{A}$ induced by σ is an infinite sequence $q_0q_1 \ldots \in Q^\omega$ of states with $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i \geq 0$. A run is accepting if it contains infinitely many accepting states (resp. only finitely many rejecting states). A nondeterministic (resp. universal) automaton $\mathcal{A}$ accepts a word σ if some run is accepting (resp. all runs are accepting). The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all accepted words. We consider nondeterministic Büchi automata (NBAs) and universal co-Büchi automata (UCAs).

Alternating ω-Automata. An alternating Büchi (resp. co-Büchi) automaton (ABA resp. ACA) $\mathcal{A} = (Q, q_0, \delta, F)$ over a finite alphabet Σ consists of a finite set of states Q, an initial state $q_0 \subseteq Q$, a transition function $\delta : Q \times 2^\Sigma \to \mathbb{B}^+(Q)$, where $\mathbb{B}^+(Q)$ is the set of positive Boolean formulas over Q, and a set of accepting (resp. rejecting) states $F \subseteq Q$. We assume that the elements of $\mathbb{B}^+(Q)$ are given in DNF. Runs of $\mathcal{A}$ are Q-labeled trees: a tree $\mathcal{T}$ is a prefix-closed subset of $\mathbb{N}^*$. The children of a node $x \in \mathcal{T}$ are $\mathrm{c}(x) = \{x \cdot d \in \mathcal{T} \mid d \in \mathbb{N}\}$. An X-labeled tree $(\mathcal{T}, \ell)$ consists of a tree $\mathcal{T}$ and a labeling function $\ell : \mathcal{T} \to X$. A branch of $(\mathcal{T}, \ell)$ is a maximal sequence $\ell(x_0)\ell(x_1)\ldots$ with $x_0 = \varepsilon$ and $x_{i+1} \in \mathrm{c}(x_i)$ for $i \geq 0$. A run tree of $\mathcal{A}$ induced by $\sigma \in (2^\Sigma)^\omega$ is a Q-labeled tree $(\mathcal{T}, \ell)$ with $\ell(\varepsilon) = q_0$ and, for all $x \in \mathcal{T}$, $\{\ell(x') \mid x' \in \mathrm{c}(x)\} \in \delta(\ell(x), \sigma_{|x|})$. A run tree is accepting if every infinite branch contains infinitely many accepting states (resp. only finitely many rejecting states). $\mathcal{A}$ accepts σ if there is some accepting run tree. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all accepted words.

Two-Player Games. An arena is a tuple $\mathbb{A} = (V, V_0, V_1, v_0, E)$, where V, V_0, V_1 are finite sets of positions with $V = V_0 \cup V_1$ and $V_0 \cap V_1 = \emptyset$, $v_0 \in V$ is the initial position, $E \subseteq V \times V$ is a set of edges such that $\forall v \in V.\ \exists v' \in V.\ (v, v') \in E$. Player i controls positions in V_i. A game $\mathcal{G} = (\mathbb{A}, W)$ consists of an arena $\mathbb{A}$ and a winning condition $W \subseteq V^\omega$. A play is an infinite sequence $\rho \in V^\omega$ such that $(\rho_i, \rho_{i+1}) \in E$ for all $i \in \mathbb{N}$. The player owning a position chooses the edge on which the play is continued. A play ρ is initial if $\rho_0 = v_0$ holds. It is winning for Player 0 if $\rho \in W$ and winning for Player 1 otherwise. A strategy for Player i is a function $\tau : V^*V_i \to V$ such that $(v, v') \in E$ whenever $\tau(w, v) = v'$ for some $w \in V^*$, $v \in V_i$. A play ρ is consistent with a strategy τ if, for all $j \in \mathbb{N}$, $\rho_j \in V_i$ implies $\rho_{j+1} = \tau(\rho_{|j})$. A strategy for Player i is winning if all initial and consistent plays are winning for Player i.

System Architectures. An architecture is a tuple $A = (P, \Sigma, \mathit{inp}, \mathit{out})$, where P is a set of processes consisting of the environment env and a set $P^- = P \setminus \{\mathit{env}\}$ of n system processes, Σ is a set of Boolean variables, $\mathit{inp} = \langle I_1, \ldots, I_n \rangle$ assigns a set $I_j \subseteq \Sigma$ of input variables to each $p_j \in P^-$, and $\mathit{out} = \langle O_{\mathit{env}}, O_1, \ldots O_n \rangle$ assigns a set $O_j \subseteq \Sigma$ of output variables to each $p_j \in P$. For all $p_j, p_k \in P^-$ with $j \neq k$, $I_j \cap O_j = \emptyset$ and $O_j \cap O_k = \emptyset$ hold. The variables Σ_j of $p_j \in P^-$ are given by $\Sigma_j = I_j \cup O_j$. The inputs I, outputs O, and variables Σ of the whole system are defined by $X = \bigcup_{p_j \in P^-} X_j$ for $X \in \{I, O, \Sigma\}$. A is called distributed if $|P^-| \geq 2$. In the remainder of this paper, we assume that a distributed architecture is given.

Process Strategies. A strategy for process p_i is a function $s_i : (2^{I_i})^* \to 2^{O_i}$ mapping a history of inputs to outputs. We model s_i as a Moore machine $\mathcal{M}_i = (T, t_0, \tau, o)$ consisting of a finite set of states T, an initial state $t_0 \in T$, a transition function $\tau : T \times 2^{I_i} \to T$, and a labeling function $o : T \to 2^{O_i}$. For a sequence $\gamma = \gamma_0\gamma_1 \ldots \in (2^{I_i})^\omega$, $\mathcal{M}_i$ produces a path $(t_0, \gamma_0 \cup o(t_0))(t_1, \gamma_1 \cup o(t_1)) \ldots \in (T \times 2^{I_i \cup O_i})^\omega$, where $\tau(t_j, \gamma_j) = t_{j+1}$. The projection of a path to the variables is called a trace. The trace produced by $\mathcal{M}_i$ on $\gamma \in (2^{I_i})^\omega$ is called the computation of s_i on γ, denoted $comp(s_i, \gamma)$. We say that s_i is winning for an LTL formula φ, denoted $s_i \models \varphi$, if $comp(s_i, \gamma) \models \varphi$ holds for all input sequences $\gamma \in (2^{I_i})^\omega$. Overloading notation with two-player games, we call a process strategy simply a strategy whenever the context is clear. The parallel composition $\mathcal{M}_i \,||\, \mathcal{M}_j$ of two Moore machines $\mathcal{M}_i = (T_i, t_0^i, \tau_i, o_i)$, $\mathcal{M}_j = (T_j, t_0^j, \tau_j, o_j)$ for $p_i, p_j \in P^-$ is the Moore machine (T, t_0, τ, o) with inputs $(I_i \cup I_j) \setminus (O_i \cup O_j)$ and outputs $O_i \cup O_j$ as well as $T = T_i \times T_j$, $t_0 = (t_0^i, t_0^j)$, $\tau((t, t'), \iota) = (\tau_i(t, (\iota \cup o_j(t')) \cap I_i), \tau_j(t', (\iota \cup o_i(t)) \cap I_j))$, and $o((t, t')) = o_i(t) \cup o_j(t')$.

Synthesis. Given a specification φ, synthesis derives strategies $s_1, \dots, s_n$ for the system processes such that $s_1 \,||\, \cdots \,||\, s_n \models \varphi$, i.e., such that the parallel composition of the strategies satisfies φ for all input sequences generated by the environment. If such strategies exist, φ is called realizable. Bounded synthesis [22] additionally bounds the size of the strategies. The search for strategies is encoded into a constraint system that is satisfiable if, and only if, φ is realizable for the size bound. There are SMT, SAT, QBF, and DQBF encodings [22, 11, 4]. We consider a compositional synthesis approach that synthesizes strategies for the processes separately. Thus, outputs produced by the other system processes are treated similar to the environment outputs, namely as part of the input sequence of the considered process. Nevertheless, compositional synthesis derives strategies such that $s_1 \,||\, \cdots \,||\, s_n \models \varphi$ holds.

3 Dominant Strategies and Liveness Properties

Given a specification φ, the naïve compositional synthesis approach is to synthesize strategies $s_1, \dots, s_n$ for the system processes such that $s_i \models \varphi$ holds for all $p_i \in P^-$. Then, it follows immediately that $s_1 \,||\, \cdots \,||\, s_n \models \varphi$ holds as well. However, since winning strategies are required to satisfy φ for every input sequence, usually no such individual winning strategies exist due to complex interconnections in the system. Therefore, the naïve approach does not succeed in many cases. The notion of *remorsefree dominance* [9], in contrast, has been successfully used in compositional synthesis [10, 16]. The main idea is to synthesize *dominant* strategies for the system processes separately instead of winning ones. Dominant strategies are, in contrast to winning strategies, allowed to violate the specification for some input sequence if no other strategy would have satisfied it in the same situation. Thus, remorsefree dominance is a weaker requirement than winning and therefore individual dominant strategies exist for more systems. Formally, remorsefree dominant strategies are defined as follows:

▶ **Definition 1** (Dominant Strategy [10]). *Let φ be an LTL formula. Let s and t be strategies for process p_i. Then, t* is dominated by *s, denoted $t \preceq s$, if for all input sequences $\gamma \in (2^{I_i})^\omega$ either $comp(s, \gamma) \models \varphi$ or $comp(t, \gamma) \not\models \varphi$ holds. Strategy s is called* dominant *for φ if $t \preceq s$ holds for all strategies t for process p_i.*

Intuitively, a strategy s dominates a strategy t if it is *at least as good* as t. It is dominant for φ if it is at least as good as *every other possible strategy* and thus if it is *as good as possible*. As an example, reconsider the message sending system. Let s_i be a strategy for process p_i that outputs m_i in the very first step. It satisfies $\varphi = \Diamond m_1 \land \Diamond m_2$ on all input sequences containing at least one m_{3-i}. On all other input sequences, it violates φ. Let t_i be some alternative strategy. Since no strategy for p_i can influence m_{3-i}, t_i satisfies φ only on input sequences containing at least one m_{3-i}. Yet, s_i satisfies φ for such sequences as well. Hence, s_i dominates t_i and since we chose t_i arbitrarily, s_i is dominant for φ.

Synthesizing dominant strategies rather than winning ones allows us to synthesize strategies for the processes of a distributed system compositionally, although no winning strategies for the individual processes exist. Dominant strategies for the individual processes can then be recomposed to obtain a strategy for the whole system. For safety specifications, the composed strategy is guaranteed to be dominant for the specification as well:

▶ **Theorem 2** (Compositionality of Dominance for Safety Properties [10]). *Let φ be an LTL formula. Let s_1 and s_2 be dominant strategies for processes p_1 and p_2, respectively, as well as for φ. If φ is a safety property, then $s_1 \,||\, s_2$ is dominant for $p_1 \,||\, p_2$ and φ.*

Compositionality is a crucial property for compositional synthesis: it allows for concluding that the parallel composition of the separately synthesized process strategies is indeed a useful strategy for the whole system. Thus, Theorem 2 enables compositional synthesis with dominant strategies for safety properties. For liveness properties, however, the parallel composition of two dominant strategies is not necessarily dominant: consider strategy t_i for p_i in the message sending system that waits for m_{3-i} before sending its own message. This strategy is dominant for φ: for input sequences in which m_{3-i} occurs eventually, t_i sends m_i in the next step, satisfying φ. For all other input sequences, no strategy for p_i can satisfy φ. Yet, the parallel composition of t_1 and t_2 does not send any message; violating φ, while there exist strategies that satisfy φ, e.g., a strategy sending both m_1 and m_2 in the first step.

Bounded dominance [10] is a variant of dominance that is compositional for both safety and liveness properties. Intuitively, it reduces the specification φ to a safety property by introducing a bound on the number of steps in which the strategy *does not make progress* with respect to φ. The progress measure is defined on an equivalent UCA $\mathcal{A}$ for φ. The measure $m_\mathcal{A}$ of a strategy s on an input sequence $\gamma \in (2^{I_i})^\omega$ is then the supremum of the number of rejecting states of the runs of $\mathcal{A}$ induced by $comp(s,\gamma)$. Thus, a strategy s n-dominates a strategy t for $\mathcal{A}$ and $n \in \mathbb{N}$ if for every $\gamma \in (2^{I_i})^\omega$, either $m_\mathcal{A}(comp(s,\gamma)) \leq n$ or $m_\mathcal{A}(comp(t,\gamma)) > n$ holds. If $\mathcal{A}$ is a safety automaton, then remorsefree dominance and bounded dominance coincide. For liveness specifications, however, they differ.

Yet, bounded dominance does not imply dominance: there are specifications φ with a minimal measure m, i.e., all strategies have a measure of at least m [10]. When choosing a bound $n < m$, every strategy is trivially n-dominant for φ, even non-dominant ones. Hence, the choice of the bound is crucial for bounded dominance. It is not obvious how to determine a *good* bound, though: it needs to be large enough to avoid non-dominant strategies. As the bound has a huge impact on the synthesis time, however, it cannot be chosen too large as otherwise synthesis becomes infeasible. Especially for specifications with several complex dependencies between processes, it is hard to determine a proper bound. Therefore, bounded dominance is not a suitable notion for compositional synthesis for liveness properties. In the remainder of this paper, we introduce a different variant of dominance that implies remorsefree dominance and that ensures compositionality also for liveness properties.

4 Delay-Dominance

In this section, we introduce a new winning condition for strategies, *delay-dominance*, which resembles remorsefree dominance but ensures compositionality also for liveness properties. It builds on the idea of bounded dominance to not only consider the satisfaction of the LTL formula φ but to measure progress based on an automaton representation of φ. Similar to bounded dominance, we utilize visits of rejecting states in a co-Büchi automaton. Yet, we use an *alternating* automaton instead of a universal one. Note that delay-dominance can be equivalently formulated on UCAs, yet, using ACAs allows for more efficient synthesis of delay-dominant strategies (see Section 5). Moreover, we do not require a fixed bound on the number of visits to rejecting states; rather, we relate visits of rejecting states induced by the delay-dominant strategy to visits of rejecting states induced by the alternative strategy.

Intuitively, delay-dominance requires that every visit to a rejecting state in the ACA $\mathcal{A}$ caused by the delay-dominant strategy *is matched* by a visit to a rejecting state caused by the alternative strategy *eventually*. The rejecting states of the ACA $\mathcal{A}$ are closely related to the satisfaction of the LTL specification φ: if infinitely many rejecting states are visited, then φ is not satisfied. Thus, delay-dominance allows a strategy to violate the specification if all alternative strategies violate it as well. Defining delay-dominance on the rejecting states of $\mathcal{A}$ instead of the satisfaction of φ allows for measuring the progress on satisfying the specification. Thus, we can distinguish strategies that wait indefinitely for another process from those that do not: intuitively, a strategy s that waits will visit a rejecting state later than a strategy t that does not. This visit to a rejecting state is then not matched eventually by a visit to a rejecting state in t, preventing delay-dominance of s.

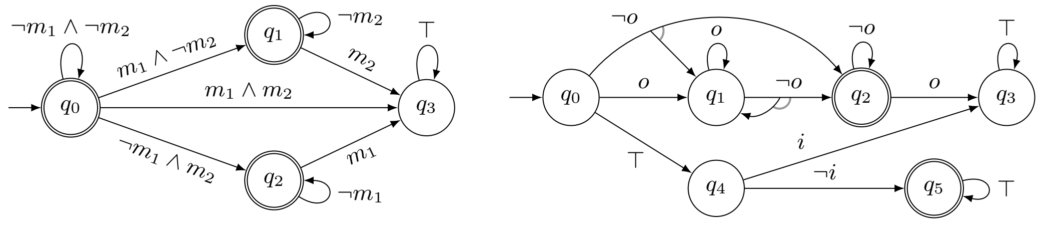
Formally, we present a game-based definition for delay-dominance: we introduce a two-player game, the so-called *delay-dominance game*, which is inspired by the delayed simulation game for alternating Büchi automata [24]. Given an ACA $\mathcal{A} = (Q, q_0, \delta, F)$, two strategies s and t for some process p_i, and an input sequence $\gamma \in (2^{I_i})^\omega$, the delay-dominance game determines whether s delay-dominates t for $\mathcal{A}$ on input γ. Intuitively, the game proceeds in rounds. At the beginning of each round, a pair (p, q) of states $p, q \in Q$ and the number of the iteration $j \in \mathbb{N}$ is given, where p represents a state that is visited by a run of $\mathcal{A}$ induced by $comp(t, \gamma)$, while q represents a state that is visited by a run of $\mathcal{A}$ induced by $comp(s, \gamma)$. We call p the *alternative state* and q the *dominant state*. Let $\sigma^s := comp(s, \gamma)$ and $\sigma^t := comp(t, \gamma)$. The players Duplicator and Spoiler, where Duplicator takes on the role of Player 0, play as follows: **1.** Spoiler chooses a set $c \in \delta(p, \sigma^t_j)$. **2.** Duplicator chooses a set $c' \in \delta(q, \sigma^s_j)$. **3.** Spoiler chooses a state $q' \in c'$. **4.** Duplicator chooses a state $p' \in c$. The starting pair of the next round is then $((p', q'), j + 1)$. Starting from $((q_0, q_0), 0)$, the players construct an infinite play which determines the winner. Duplicator wins for a play if every rejecting dominant state is matched by a rejecting alternative state eventually.

Both the delay-dominant strategy s and the alternative strategy t may control the nondeterministic transitions of $\mathcal{A}$, while the universal ones are uncontrollable. Since, intuitively, strategy t is controlled by an opponent when proving that s delay-dominates t, we thus have a change in control for t: for s, Duplicator controls the existential transitions of $\mathcal{A}$ and Spoiler controls the universal ones. For t, Duplicator controls the universal transitions and Spoiler controls the existential ones. Note that the order in which Spoiler and Duplicator make their moves is crucial to ensure that Duplicator wins the game when considering the very same process strategies. By letting Spoiler move first, Duplicator is able to mimic – or *duplicate* – Spoiler's moves. Formally, the delay-dominance game is defined as follows:

▶ **Definition 3** (Delay-Dominance game). *Let $\mathcal{A} = (Q, q_0, \delta, F)$ be an ACA. Based on $\mathcal{A}$, we define the sets $S_\exists = (Q \times Q) \times \mathbb{N}$, $D_\exists = (Q \times Q \times 2^Q) \times \mathbb{N}$, $S_\forall = (Q \times Q \times 2^Q \times 2^Q) \times \mathbb{N}$, and $D_\forall = (Q \times Q \times Q \times 2^Q) \times \mathbb{N}$. Let $\sigma, \sigma' \in (2^{\Sigma_i})^\omega$ be infinite sequences. Then, the* delay-dominance game *$(\mathcal{A}, \sigma, \sigma')$ is the game $\mathcal{G} = (\mathbb{A}, W)$ defined by $\mathbb{A} = (V, V_0, V_1, v_0, E)$ with $V = S_e \cup D_e \cup S_u \cup D_u$, $V_0 = D_e \cup D_u$, and $V_1 = S_e \cup S_u$ as well as*

$$\begin{aligned} E = {} & \{(((p,q),j),((p,q,c),j)) \mid c \in \delta(p,\sigma_j)\} \cup \{(((p,q,c),j),((p,q,c,c'),j)) \mid c' \in \delta(q,\sigma'_j)\} \\ & \cup \{(((p,q,c,c'),j),((p,q,c,q'),j)) \mid q' \in c'\} \cup \{(((p,q,c,q'),j),((p',q'),j+1)) \mid p' \in c\}, \end{aligned}$$

and the winning condition $W = \{\rho \in V^\omega \mid \forall j \in \mathbb{N}.\ f_{dom}(\rho_j) \in F \rightarrow \exists j' \geq j.\ f_{alt}(\rho_{j'}) \in F\}$, where $f_{alt}(v) := 1(1(v))$ and $f_{dom}(v) := 2(1(v))$, i.e., $f_{alt}(v)$ and $f_{dom}(v)$ map a position v to the alternative state and the dominant state of v, respectively.



(a) ACA $\mathcal{A}_\varphi$ for $\varphi = \diamondsuit m_1 \wedge \diamondsuit m_2$. **(b)** ACA $\mathcal{A}_\psi$ for $\psi = \square\diamondsuit o \vee \bigcirc i$.

Figure 1 Alternating co-Büchi automata $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$. Universal choices are depicted by connecting the transitions with a gray arc. Rejecting states are marked with double circles.

We now define the notion of delay-dominance based on the delay-dominance game. Intuitively, the winner of the game for the computations of two strategies s and t determines whether or not s delay-dominates t on a given input sequence. Similar to remorsefree dominance, we then lift this definition to delay-dominant strategies. Formally:

▶ **Definition 4** (Delay-Dominant Strategy). *Let $\mathcal{A}$ be an ACA. Let s and t be strategies for process p_i. Then, s* delay-dominates *t on input sequence $\gamma \in (2^{I_i})^\omega$ for $\mathcal{A}$, denoted $t \trianglelefteq^{\mathcal{A}}_{\gamma} s$, if Duplicator wins the delay-dominance game $(\mathcal{A}, comp(t,\gamma), comp(s,\gamma))$. Strategy s* delay-dominates *t for $\mathcal{A}$, denoted $t \trianglelefteq^{\mathcal{A}} s$, if $t \trianglelefteq^{\mathcal{A}}_{\gamma} s$ holds for all input sequences $\gamma \in (2^{I_i})^\omega$. Strategy s is* delay-dominant *for $\mathcal{A}$ if, for every alternative strategy t for p_i, $t \trianglelefteq^{\mathcal{A}} s$ holds.*

As an example for delay-dominance, consider the message sending system again. Let s_i be a strategy for process p_i that outputs m_i in the very first step and let t_i be a strategy that waits for m_{3-i} before sendings its own message. An ACA $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ is depicted in Figure 1a. Note that $\mathcal{A}_\varphi$ is deterministic and thus every sequence induces a single run tree with a single branch. Hence, for every input sequence $\gamma \in (2^{I_i})^\omega$, the moves of both Spoiler and Duplicator are uniquely defined by the computations of t_1 and s_1 on γ, respectively. Therefore, we only provide the state pairs (p,q) of the delay-dominance game, not the intermediate tuples. First, consider an input sequence $\gamma \in (2^{I_1})^\omega$ that contains the very first m_2 at point in time ℓ. Then, the run of $\mathcal{A}_\varphi$ on $comp(s_1,\gamma)$ starts in q_0, moves to q_1 immediately if $\ell > 0$, stays there up to the occurrence of m_2 and then moves to q_3, where it stays forever. If $\ell = 0$, then the run moves immediately from q_0 to q_3. The run of $comp(t_1,\gamma)$, in contrast, stays in q_0 until m_2 occurs, then moves to q_2 and then immediately to q_3, where it stays forever. Thus, we obtain the unique sequence $(q_0,q_0)(q_0,q_1)^{\ell-1}(q_2,q_3)(q_3,q_3)^\omega$ of state pairs in the delay-dominance game $(\mathcal{A}_\varphi, comp(t_1,\gamma), comp(s_1,\gamma))$. The last rejecting alternative state, i.e., a rejecting state induced by $comp(t_1,\gamma)$ occurs at point in time $\ell+1$, namely q_2, while the last rejecting dominant state i.e., a rejecting state induced by $comp(s_1,\gamma)$, occurs at point in time ℓ, namely q_1. Thus, $t_1 \trianglelefteq^{\mathcal{A}_\varphi}_{\gamma} s_1$ holds. In fact, $t_1' \trianglelefteq^{\mathcal{A}_\varphi}_{\gamma} s_1$ holds for *all* alternative strategies t_1' for such an input sequence γ since every strategy t_1' for p_1 induces at least ℓ visits to rejecting states due to the structure of γ. Second, consider an input sequence $\gamma' \in (2^{I_i})^\omega$ that does not contain any m_2. Then, the run of $\mathcal{A}_\varphi$ on a computation of any strategy t_1' on γ' never reaches q_3 and thus only visits rejecting states. Hence, in particular, every visit to a rejecting state induced by $comp(s_1,\gamma')$ is matched by a visit to a rejecting state induced by $comp(t_1',\gamma')$ for all strategies t_1'. Thus, $t_1' \trianglelefteq^{\mathcal{A}_\varphi}_{\gamma'} s_1$ holds for all alternative strategies t_1' as well. We can thus conclude that s_1 is delay-dominant for $\mathcal{A}_\varphi$, meeting our intuition that s_1 should be allowed to violate φ on input sequences that do not contain any m_2. Strategy t_1, in contrast, is remorsefree dominant for φ but not delay-dominant for $\mathcal{A}_\varphi$: consider again an input sequence $\gamma \in (2^{I_1})^\omega$ that contains the very first m_2 at point in time ℓ. For the delay-dominance game $(\mathcal{A}_\varphi, \mathit{comp}(s_1,\gamma), \mathit{comp}(t_1,\gamma))$, we obtain the following sequence of state pairs: $(q_0,q_0)(q_1,q_0)^{\ell-1}(q_3,q_2)(q_3,q_3)^\omega$. It contains a rejecting dominant state, i.e., a rejecting state induced by $\mathit{comp}(t_1,\gamma)$, at point in time $\ell+1$, while the last rejecting alternative state occurs at point in time ℓ. Hence, t_1 does not delay-dominate s_1, preventing that it is delay-dominant to wait for the other process indefinitely.

Next, consider the LTL formula $\psi = \square\diamond o \vee \bigcirc i$, where i is an input variable and o is an output variable. An ACA $\mathcal{A}_\psi$ with $\mathcal{L}(\mathcal{A}_\psi) = \mathcal{L}(\psi)$ is depicted in Figure 1b. Note that it has both existential and universal transitions. Consider a strategy s that outputs o in every step. Let t be some alternative strategy and let γ be some input sequence. Then, Duplicator encounters an existential choice in state q_0 for s in the very first round of the delay-dominance game $(\mathcal{A}_\psi, \mathit{comp}(t,\gamma), \mathit{comp}(s,\gamma))$: it can choose to move to q_1 or to q_4. If Duplicator chooses to move to q_1, then the only possible successor state in every run of $\mathcal{A}_\psi$ induced by $\mathit{comp}(s,\gamma)$ is q_1. Thus, irrespective of Spoiler's moves, the sequence of dominant states in all consistent initial plays is given by $q_0 q_1^\omega$. Since neither q_0 nor q_1 is rejecting, Duplicator wins the game. Therefore, there exists a winning strategy for Duplicator for the game $(\mathcal{A}_\psi, \mathit{comp}(t,\gamma), \mathit{comp}(s,\gamma))$ for all t and γ, namely choosing to move to q_1 from q_0, and thus s is delay-dominant. Second, consider a strategy t that does not output o in the first step but outputs o in every step afterwards. Let γ be an input sequence that does not contain i at the second point in time. Then, Duplicator encounters an existential choice in state q_0 for t in the very first round of the delay-dominance game $(\mathcal{A}_\psi, \mathit{comp}(s,\gamma), \mathit{comp}(t,\gamma))$. Yet, if Duplicator chooses the transition from q_0 to q_4, then every consistent play will contain infinitely many rejecting dominant states since the structure of γ enforces that every consistent play enters q_5 in its dominant state in the next round of the game. Otherwise, i.e., if Duplicator chooses the universal transition to both q_1 and q_2, then Spoiler decides which of the states is entered. If Spoiler chooses q_2, then every consistent play visits a rejecting dominant state, namely q_2, in the second round of the game. If Spoiler further chooses to move from q_0 to q_1 for the alternative strategy s, then, as shown above, no rejecting dominant states are visited in a consistent play at all. Thus, there exists a winning strategy for Spoiler and therefore t is not delay-dominant for $\mathcal{A}_\psi$.

Recall that one of the main weaknesses of bounded dominance is that every strategy, even a non-dominant one, is trivially n-dominant if the bound n is chosen too small. Every delay-dominant strategy, in contrast, is also remorsefree dominant. The main idea is that a winning strategy τ of Duplicator in the delay-dominance game defines a run tree of the automaton induced by the delay-dominant strategy s such that all branches either visit only finitely many rejecting states or such that all rejecting states are matched eventually with a rejecting state in some branch, which is also defined by τ, of all run trees induced by an alternative strategy. Thus, s either satisfies the specification, or an alternative strategy does not satisfy it either. For the formal proof, we refer the reader to [19].

▶ **Theorem 5.** *Let φ be an LTL formula. Let $\mathcal{A}_\varphi$ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. Let s be a strategy for process p_i. If s is delay-dominant for $\mathcal{A}_\varphi$, then s is remorsefree dominant for φ.*

Clearly, the converse does not hold: for instance, a strategy in the message sending system that waits for the other process to send its message first is remorsefree dominant for φ but not delay-dominant for the ACA depicted in Figure 1a as pointed out above.

Given an LTL formula φ, for remorsefree dominance it holds that if φ is realizable, then every strategy that is dominant for φ is also winning for φ [10]. This is due to the fact that the winning strategy needs to be taken into account as an alternative strategy for every dominant one, and that remorsefree dominance is solely defined on the satisfaction of the specification. With Theorem 5 the same property follows for delay-dominance.

▶ **Lemma 6.** *Let φ be an LTL formula. Let $\mathcal{A}_\varphi$ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. If φ is realizable, then every delay-dominant strategy for $\mathcal{A}_\varphi$ is winning for φ as well.*

A critical shortcoming of remorsefree dominance is its non-compositionality for liveness properties. This restricts the usage of dominance-based compositional synthesis algorithms to safety specifications, which are in many cases not expressive enough. Delay-dominance, in contrast, is specifically designed to be compositional for both safety and liveness properties. Intuitively, this is the case since there is a smallest bad prefix of the computation of a strategy for the delay-dominance requirement. This heavily relies on the fact that delay-dominance is defined using a two-player game and thus we require the existence of a *strategy* for Duplicator, i.e., determining which decisions to make for the existential choices of the delay-dominant strategy and the universal ones for the alternative strategy has to be possible without knowledge about the future input as well as the future decisions for the other choices. If the parallel composition of two delay-dominant strategies is not delay-dominant, then the behavior of both processes at the last position of this prefix reveals which one of them is responsible for the violation of φ. Note that also both processes can be responsible simultaneously. Since there is an alternative strategy t for the composed system for which Duplicator wins the game, as otherwise the parallel composition would be delay-dominant, the strategy of the process p_i which is responsible for Duplicator losing the game cannot be delay-dominant since there is an alternative strategy, namely t restricted to the outputs of p_i, that allows Duplicator to win the game. For the formal proof, we refer to [19].

▶ **Theorem 7** (Compositionality of Delay-Dominance). *Let $\mathcal{A}$ be an ACA. Let s_1 and s_2 be delay-dominant strategies for $\mathcal{A}$ and processes p_1 and p_2, respectively. Then, their parallel composition $s_1 \,||\, s_2$ is delay-dominant for $\mathcal{A}$ and $p_1 \,||\, p_2$.*

From both Theorem 5 and Theorem 7 it then follows immediately that the parallel composition of two delay-dominant strategies is also remorsefree dominant:

▶ **Corollary 8.** *Let φ be an LTL formula and let $\mathcal{A}_\varphi$ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. Let s_1 and s_2 be delay-dominant strategies for $\mathcal{A}_\varphi$ and processes p_1 and p_2, respectively. Then, $s_1 \,||\, s_2$ is remorsefree dominant for φ and $p_1 \,||\, p_2$.*

We obtain with Lemma 6 and Theorem 7 that, given a specification φ, the parallel composition of delay-dominant strategies for all processes of a distributed system is winning if φ is realizable. Hence, delay-dominance can be soundly used for dominance-based compositional synthesis approaches. In the next section, we thus introduce an automaton construction for synthesizing delay-dominant strategies.

5 Synthesizing Delay-Dominant Strategies

In this section, we introduce how delay-dominant strategies can be synthesized using existing tools for synthesizing winning strategies. We focus on utilizing *bounded synthesis* tools such as BoSy [12]. Mostly, we use bounded synthesis as a black box procedure throughout this section. Therefore, we do not go into detail here and refer the interested reader to [22, 11]. A crucial observation regarding bounded synthesis that we utilize, however, is that it translates the given specification φ into an equivalent universal co-Büchi automaton $\mathcal{A}_\varphi$ and then derives a strategy such that, for every input sequence, the runs of $\mathcal{A}_\varphi$ induced by the computation of the strategy on the input sequence visits only finitely many rejecting states.

To synthesize delay-dominant strategies instead of winning ones, we can thus use existing bounded synthesis algorithms by replacing the universal co-Büchi automaton $\mathcal{A}_\varphi$ with one

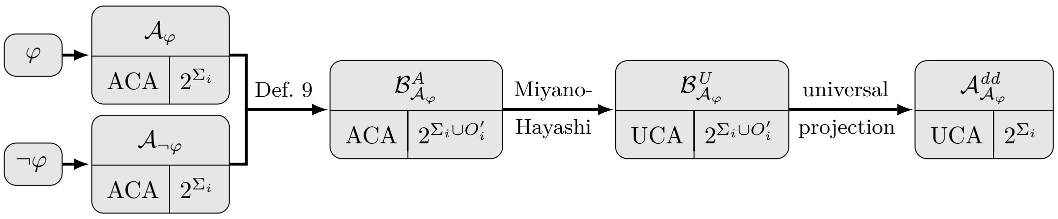


Figure 2 Overview of the construction of a universal co-Büchi automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ recognizing delay-dominant strategies for the alternating co-Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. The lower parts of the boxes list the automaton type (alternating or universal) and the alphabet.

encoding delay-dominance, i.e., with an automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ such that its runs induced by the computations of a delay-dominant strategy on all input sequences visits only finitely many rejecting states. This idea is similar to the approach for synthesizing remorsefree dominant strategies [10, 16]. The automaton for recognizing delay-dominant strategies, however, differs inherently from the one for recognizing remorsefree dominant strategies.

The automaton construction consists of several steps. An overview is given in Figure 2. Since delay-dominance is not defined on the LTL specification φ itself but on an equivalent alternating co-Büchi automaton, we first translate φ into an alternating co-Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. For this, we utilize well-known algorithms for translating LTL formulas into equivalent alternating Büchi automata as well as the duality of the Büchi and co-Büchi acceptance condition and of nondeterministic and universal branching. More details on the translation of LTL formulas into alternating co-Büchi automata are provided in [19]. Similarly, we construct an alternating co-Büchi automaton $\mathcal{A}_{\neg\varphi}$ with $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$ from $\neg\varphi$. The centerpiece of the construction is an alternating co-Büchi automaton $\mathcal{B}^A_{\mathcal{A}_\varphi}$ constructed from $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ that recognizes whether $t \trianglelefteq^{\mathcal{A}_\varphi}_\gamma s$ holds for $\mathcal{A}_\varphi$, input sequence $\gamma \in (2^{I_i})^\omega$ and strategies s and t for process p_i. The alternating automaton $\mathcal{B}^A_{\mathcal{A}_\varphi}$ is then translated into an equivalent universal co-Büchi automaton $\mathcal{B}^U_{\mathcal{A}_\varphi}$, e.g., with the Miyano-Hayashi algorithm [30]. Lastly, we translate $\mathcal{B}^U_{\mathcal{A}_\varphi}$ into a universal co-Büchi automaton that accounts for requiring a strategy s to delay-dominate *all* other strategies t and not only a particular one utilizing universal projection. In the remainder of this section, we describe all steps of the construction in detail and prove their correctness.

5.1 Construction of the ACA $\mathcal{B}^A_{\mathcal{A}_\varphi}$

From the two ACAs $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$, we construct an alternating co-Büchi automaton $\mathcal{B}^A_{\mathcal{A}_\varphi}$ that recognizes whether $t \trianglelefteq^{\mathcal{A}_\varphi}_\gamma s$ holds for $\mathcal{A}_\varphi$, input sequence $\gamma \in (2^{I_i})^\omega$ and strategies s and t for process p_i. The construction relies on the observation that $t \trianglelefteq^{\mathcal{A}_\varphi}_\gamma s$ holds if, and only if, either (i) $comp(t, \gamma) \not\models \varphi$ holds or (ii) we have $t \trianglelefteq^{\mathcal{A}_\varphi}_\gamma s$ and every initial play of the delay-dominance game that is consistent with the winning strategy of Duplicator visits only finitely many rejecting dominant states. The proof of this observation is provided in [19]. Therefore, the automaton $\mathcal{B}^A_{\mathcal{A}_\varphi}$ consists of two parts, one accounting for (i) and one accounting for (ii), and guesses nondeterministically in the initial state which part is entered. The ACA $\mathcal{A}_{\neg\varphi}$ with $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$ accounts for (i). For (ii), we intuitively build the product of two copies of the ACA $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$, one for each of the considered process strategies s and t. Note that similar to the change of control for t in the delay-dominance game, we consider the *dual* transition function of $\mathcal{A}_\varphi$, i.e., the one where conjunctions and disjunctions are swapped, for the copy of $\mathcal{A}_\varphi$ for t. We keep track of whether we encountered a situation in which a rejecting state was visited for s while it was not for t. This allows for defining the set of rejecting states.

Note that we need to allow for differentiating valuations of output variables computed by s and t on the same input sequence. Therefore, we extend the alphabet of $\mathcal{B}^A_{\mathcal{A}_\varphi}$: in addition to the set Σ_i of variables of process p_i, which contains input variables I_i and output variables O_i, we consider the set $O'_i := \{o' \mid o \in O_i\}$ of *primed output variables* of p_i, where every output variable is marked with a prime to obtain a fresh symbol. The set Σ'_i of *primed variables* of p_i is then given by $\Sigma'_i := I_i \cup O'_i$. Intuitively, the output variables O_i depict the behavior of the delay-dominant strategy s, while the primed output variables O'_i depict the behavior of the alternative t. The alphabet of $\mathcal{B}^A_{\mathcal{A}_\varphi}$ is then given by $2^{\Sigma_i \cup O'_i}$. This is equivalent to $2^{\Sigma_i \cup \Sigma'_i}$ since the input variables are never primed to ensure that we consider the same input sequence for both strategies. In the following, we use the functions $pr : \Sigma_i \to \Sigma'_i$ and $unpr : \Sigma'_i \to \Sigma_i$ to switch between primed variables and normal ones: given a valuation $a \in \Sigma_i$ of variables, $pr(a)$ replaces every output variable $o \in O_i$ occurring in a with its primed version o'. For a valuation $a \in \Sigma'_i$, $unpr(a)$ replaces every primed output variable $o' \in O'_i$ occurring in a with its normal unprimed version o. We extend pr and $unpr$ to finite and infinite sequences as usual. The ACA $\mathcal{B}^A_{\mathcal{A}_\varphi}$ is then constructed as follows:

▶ **Definition 9.** *Let φ be an LTL formula over alphabet 2^{Σ_i}. Let $\mathcal{A}_\varphi = (Q, q_0, \delta, F)$ be an ACA with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. Let $\mathcal{A}_{\neg\varphi} = (Q^c, q^c_0, \delta^c, F^c)$ be an ACA with $\mathcal{L}(\neg\varphi) = \mathcal{L}(\mathcal{A}_{\neg\varphi})$. We construct the ACA $\mathcal{B}^A_{\mathcal{A}_\varphi} = (Q^A, Q^A_0, \delta^A, F^A)$ with alphabet $2^{\Sigma_i \cup O'_i}$ as follows.*

- $Q^A := (Q \times Q \times \{\top, \bot\}) \cup Q^c$
- $Q^A_0 := (q_0, q_0, \top)$
- $F^A := (Q \times Q \times \{\bot\}) \cup F^c$
- $\delta^A : ((Q \times Q \times \{\top, \bot\}) \cup Q^c) \times 2^{\Sigma_i \cup O'_i} \to (Q \times Q \times \{\top, \bot\}) \cup Q^c$ *with*

$$\delta^A(q_c, \tilde{\iota}) := \delta^c(q_c, \iota') \quad \textit{for } q_c \in Q^c$$
$$\delta^A((q_0, q_0, \top), \tilde{\iota}) := \delta^c(q_0, \iota') \vee \bigwedge_{c \in \delta(q_0, \iota')} \bigvee_{c' \in \delta(q_0, \iota)} \bigwedge_{q' \in c'} \bigvee_{p' \in c} \vartheta(p', q', \top)$$
$$\delta^A((p, q, m), \tilde{\iota}) := \bigwedge_{c \in \delta(p, \iota')} \bigvee_{c' \in \delta(q, \iota)} \bigwedge_{q' \in c'} \bigvee_{p' \in c} \vartheta(p', q', m)$$

where $\iota := \tilde{\iota} \cap \Sigma_i$, $\iota' := unpr(\tilde{\iota} \cap \Sigma'_i)$, and $\vartheta : (Q \times Q \times \{\top, \bot\}) \to Q \times Q \times \{\top, \bot\}$ with

$$\vartheta(p, q, m) := \begin{cases} (p, q, \bot) & \textit{if } p \notin F,\ q \in F, \textit{ and } m = \top \\ (p, q, \bot) & \textit{if } p \notin F \textit{ and } m = \bot \\ (p, q, \top) & \textit{otherwise} \end{cases}$$

Note that $\mathcal{B}^A_{\mathcal{A}_\varphi}$ indeed consists of two parts: the one defined by states of the form (p, q, m), and the one defined by the states of $\mathcal{A}_{\neg\varphi}$. By definition of δ^A, these parts are only connected in the initial state of $\mathcal{B}^A_{\mathcal{A}_\varphi}$, where a nondeterministic transition to the respective successors in both parts ensures that choosing nondeterministically whether (i) or (ii) will be satisfied is possible. For states of the form (p, q, m), the mark $m \in \{\top, \bot\}$ determines whether there are *pending visits to rejecting states* in the copy of $\mathcal{A}_\varphi$ for the dominant strategy, i.e., the second component q of (p, q, m). A pending visit to a rejecting state is one that is not yet matched by a visit to a rejecting state in the copy of $\mathcal{A}_\varphi$ for the alternative strategy. Thus, ϑ defines that if a visit to a rejecting dominant state, that is not immediately matched with a rejecting alternative state, is encountered, the mark is set to $\bot$. As long as no rejecting alternative state is visited, the mark stays set to $\bot$. If a matching rejecting alternative state occurs, however, the mark is reset to $\top$. States of $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ marked with $\bot$ are then defined to be rejecting states, ensuring that a visit to a rejecting dominant state is not pending forever.

The ACA $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ constructed from ACAs $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ according to Definition 9 is sound and complete in the sense that it recognizes whether or not a strategy s delay-dominates another strategy t on an input sequence $\gamma \in (2^{I_i})^\omega$. That is, $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ accepts the infinite word $comp(s,\gamma) \cup pr(comp(t,\gamma) \cap O_i)$ if, and only if, $t \trianglelefteq^{\mathcal{A}_\varphi}_{\gamma} s$ holds for $\mathcal{A}_\varphi$. The main idea is that a run tree of $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ can be translated into a strategy for Duplicator in the delay-dominance game and vice versa since, by construction, both define the existential choices in $\mathcal{A}_\varphi$ for s and the universal choices in $\mathcal{A}_\varphi$ for t. Thus, for a run tree of $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ whose branches all visit only finitely many rejecting states, there exists a strategy for Duplicator in the delay-dominance game that ensures that for all consistent plays either $comp(t,\gamma) \not\models \varphi$ holds or, by construction of ϑ and δ^A, every rejecting dominant state is matched by a rejecting alternative state eventually. Similarly, a winning strategy for Duplicator can be translated into a run tree r of $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$. If $comp(t,\gamma) \models \varphi$ holds, then r visits only finitely many rejecting states since only finitely many rejecting dominant states are visited. If $comp(t,\gamma) \not\models \varphi$, then there exists a run tree, namely one entering the part of $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ that coincides with $\mathcal{A}_{\neg\varphi}$, whose branches all visit only finitely many rejecting states. The proof is given in [19].

▶ **Lemma 10.** *Let φ be an LTL formula. Let $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ be ACAs with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$ and $\mathcal{L}(\neg\varphi) = \mathcal{L}(\mathcal{A}_{\neg\varphi})$. Let $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ be the ACA constructed from $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ according to Definition 9. Let s and t be strategies for process p_i. Let $\gamma \in (2^{I_i})^\omega$. Let $\sigma \in (2^{\Sigma_i \cup O'_i})^\omega$ with $\sigma := comp(s,\gamma) \cup pr(comp(t,\gamma) \cap O_i)$. Then, $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ accepts σ if, and only if, $t \trianglelefteq^{\mathcal{A}_\varphi}_{\gamma} s$ holds.*

Thus, $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ determines whether or not a strategy s delay-dominates a strategy t. However, $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ cannot directly be used for synthesizing delay-dominant strategies since (i) $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ is an alternating automaton, while we require a universal automaton for bounded synthesis, and (ii) $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ considers *one particular* alternative strategy t. For recognizing delay-dominance, we need to consider *all* alternative strategies, though. Thus, we describe in the remainder of this section how $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ can be translated into a UCA for bounded synthesis.

5.2 Construction of the UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$

Next, we translate the ACA $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ constructed in the previous subsection to a UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ that can be used for synthesizing delay-dominant strategies. As outlined before, we need to (i) translate $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ into a UCA, and (ii) ensure that the automaton considers *all* alternative strategies instead of a particular one. Thus, we proceed in two steps. First, we translate $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ into an equivalent UCA $\mathcal{B}^{U}_{\mathcal{A}_\varphi}$. We utilize the Miyano-Hayashi algorithm [30] for translating ABAs into NBAs. Since we are considering co-Büchi automata instead of Büchi automata, we further make use of the duality of nondeterministic and universal branching and the Büchi and co-Büchi acceptance conditions. The translation introduces an exponential blow-up in the number of states. For the full construction, we refer to [19].

▶ **Lemma 11.** *Let $\mathcal{A}$ be an alternating co-Büchi automaton with m states. There exists a universal co-Büchi automaton $\mathcal{B}$ with $\mathcal{O}(2^m)$ states such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ holds.*

Next, we construct the desired universal co-Büchi automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ that recognizes delay-dominant strategies for $\mathcal{A}_\varphi$. For this sake, we need to adapt $\mathcal{B}^{U}_{\mathcal{A}_\varphi}$ to consider *all* alternative strategies instead of a particular one. Similar to the automaton construction for synthesizing remorsefree dominant strategies [10, 16], we utilize *universal projection*:

▶ **Definition 12** (Universal Projection). *Let $\mathcal{A} = (Q, Q_0, \delta, F)$ be a UCA over alphabet 2^Σ and let $X \subset \Sigma$. The* universal projection of $\mathcal{A}$ to X *is the UCA $\pi_X(\mathcal{A}) = (Q, Q_0, \pi_X(\delta), F)$ over alphabet 2^X, where $\pi_X(\delta) = \{(q, a, q') \in Q \times 2^X \times Q \mid \exists b \in 2^{\Sigma \setminus X}.\ (q, a \cup b, q') \in \delta\}$.*

The projected automaton $\pi_X(\mathcal{A})$ for a UCA $\mathcal{A}$ over 2^Σ and a set $X \subset \Sigma$ contains the transitions of $\mathcal{A}$ for *all* possible valuations of the variables in $\Sigma \setminus X$. Hence, for a sequence $\sigma \in (2^X)^\omega$, all runs of $\mathcal{A}$ on sequences extending σ with some valuation of the variables in $\Sigma \setminus X$ are also runs of $\pi_X(\mathcal{A})$. Since both $\mathcal{A}$ and $\pi_X(\mathcal{A})$ are universal automata, $\pi_X(\mathcal{A})$ thus accepts a sequence $\sigma \in (2^X)^\omega$ if, and only if, $\mathcal{A}$ accepts all sequences extending σ with some valuation of the variables in $\Sigma \setminus X$. The proof is given in [19].

▶ **Lemma 13.** *Let $\mathcal{A}$ be a UCA over alphabet 2^Σ and let $X \subset \Sigma$. Let $\sigma \in (2^X)^\omega$. Then, $\pi_X(\mathcal{A})$ accepts σ if, and only if $\mathcal{A}$ accepts all $\sigma' \in (2^\Sigma)^\omega$ with $\sigma' \cap X = \sigma$.*

We utilize this property to obtain a universal co-Büchi automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ from $\mathcal{B}^U_{\mathcal{A}_\varphi}$ that considers *all* possible alternative strategies instead of only a particular one: we project to the unprimed variables of p_i, i.e., to Σ_i, thereby quantifying universally over the alternative strategies. We thus obtain a UCA that recognizes delay-dominant strategies as follows:

▶ **Definition 14** (Delay-Dominance Automaton). *Let φ be an LTL formula. Let $\mathcal{A}_\varphi$, $\mathcal{A}_{\neg\varphi}$ be ACAs with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$, $\mathcal{L}(\mathcal{A}_{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. Let $\mathcal{B}^A_{\mathcal{A}_\varphi}$ be the ACA constructed from $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ according to Definition 9. Let $\mathcal{B}^U_{\mathcal{A}_\varphi}$ be a UCA with $\mathcal{L}(\mathcal{B}^A_{\mathcal{A}_\varphi}) = \mathcal{L}(\mathcal{B}^U_{\mathcal{A}_\varphi})$. The* delay-dominance universal co-Büchi automaton *$\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ for $\mathcal{A}_\varphi$ is then given by $\mathcal{A}^{dd}_{\mathcal{A}_\varphi} := \pi_{\Sigma_i}(\mathcal{B}^U_{\mathcal{A}_\varphi})$.*

Utilizing the previous results, we can now show soundness and completeness of the delay-dominance universal co-Büchi automaton $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$: from Lemma 10, we know that $\mathcal{B}^A_{\mathcal{A}_\varphi}$ recognizes whether or not a strategy s for a process p_i delay-dominates another strategy t for p_i for $\mathcal{A}_\varphi$ on an input sequence $\gamma \in (2^{I_i})^\omega$. By Lemma 11, we have $\mathcal{L}(\mathcal{B}^U_{\mathcal{A}_\varphi}) = \mathcal{L}(\mathcal{B}^A_{\mathcal{A}_\varphi})$. With the definition of the delay-dominance UCA, namely $\mathcal{A}^{dd}_{\mathcal{A}_\varphi} := \pi_{\Sigma_i}(\mathcal{B}^U_{\mathcal{A}_\varphi})$, as well as with Lemma 13, it then follows that $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ accepts $\mathit{comp}(s, \gamma)$ for all input sequences $\gamma \in (2^{I_i})^\omega$ if, and only if, s is delay-dominant for $\mathcal{A}_\varphi$. For the formal proof, we refer to [19].

▶ **Theorem 15** (Soundness and Completeness). *Let φ be an LTL formula and let $\mathcal{A}_\varphi$ be an ACA with $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$. Let $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ be the delay-dominance UCA for $\mathcal{A}_\varphi$ as constructed in Definition 14. Let s be a strategy for process p_i. Then $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ accepts $\mathit{comp}(s, \gamma)$ for all $\gamma \in (2^{I_i})^\omega$, if, and only if s is delay-dominant for $\mathcal{A}_\varphi$.*

Furthermore, $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ is of convenient size: for an LTL formula φ, there is an ACA $\mathcal{A}_\varphi$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ such that $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ constructed from $\mathcal{A}_\varphi$ is of exponential size in the squared length of the formula φ. This follows from Lemma 11 and from the facts that (i) $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ both are of linear size in the length of the formula, and (ii) universal projection preserves the automaton size. The proof is given in [19].

▶ **Lemma 16.** *Let φ be an LTL formula and let s be a strategy for process p_i. There is an ACA $\mathcal{A}_\varphi$ of size $\mathcal{O}(|\varphi|)$ with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$ and a UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ of size $\mathcal{O}(2^{|\varphi|^2})$ such that $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ accepts $\mathit{comp}(s, \gamma)$ for all $\gamma \in (2^{I_i})^\omega$, if, and only if, s is delay-dominant for $\mathcal{A}_\varphi$.*

Since the automaton construction described in this section is sound and complete, the UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ can be used for synthesizing delay-dominant strategies. In fact, it immediately enables utilizing existing bounded synthesis tools for the synthesis of delay-dominant strategies by replacing the UCA recognizing winning strategies with $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$.

Note that, similar as for the UCA recognizing remorsefree dominance [10], $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ can be translated into a nondeterministic parity tree automaton with an exponential number of colors and a doubly-exponential number of states in the squared length of the formula. Synthesizing delay-dominant strategies thus reduces to checking tree automata emptiness and, if the automaton is non-empty, to extracting a Moore machine representing a process strategy from an accepted tree. This can be done in exponential time in the number of colors and in polynomial time in the number of states [25]. With Lemma 16, a doubly-exponential complexity for synthesizing delay-dominant strategies thus follows:

▶ **Theorem 17.** *Let φ be an LTL formula and let $\mathcal{A}_\varphi$ be an ACA with $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. If there exists a delay-dominant strategy for $\mathcal{A}_\varphi$, then it can be computed in* 2EXPTIME.

It is well-known that synthesizing winning strategies is 2EXPTIME-complete [32]. Since there exists a UCA of exponential size in the length of the formula which recognizes remorsefree dominant strategies, dominant strategies can also be synthesized in 2EXPTIME [10]. Synthesizing delay-dominant strategies rather than winning or remorsefree dominant ones thus does not introduce any overhead, while it allows for a simple compositional synthesis approach for distributed systems for both safety and liveness specifications.

6 Compositional Synthesis with Delay-Dominant Strategies

In this section, we describe a compositional synthesis approach that utilizes delay-dominant strategies. We extend the algorithm described in [10] from safety specifications to general properties by synthesizing delay-dominant instead of remorsefree dominant ones. Hence, given a distributed architecture and an LTL specification φ, the compositional synthesis algorithm proceeds in four steps. First, φ is translated into an equivalent ACA $\mathcal{A}_\varphi$ using standard algorithms. Second, for each system process p_i, we construct the UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ that recognizes delay-dominant strategies for φ and p_i as described in Section 5. Note that although the initial automaton $\mathcal{A}_\varphi$ is the same for every process p_i, the UCAs recognizing delay-dominant strategies differ: since the processes have different sets of output variables, already the alphabets of the intermediate ACA $\mathcal{B}^{A}_{\mathcal{A}_\varphi}$ differ for different processes. Third, a delay-dominant strategy s_i is synthesized for each process p_i from the respective UCA $\mathcal{A}^{dd}_{\mathcal{A}_\varphi}$ with bounded synthesis. Lastly, the strategies $s_1, \dots, s_n$ are composed according to the definition of the parallel composition of Moore machines (see Section 2) into a single strategy s for the whole distributed system. By Theorem 7, the composed strategy s is delay-dominant for $\mathcal{A}_\varphi$. If φ is realizable, then, by Lemma 6, s is guaranteed to be winning for φ.

Note that even for realizable LTL formulas φ, there does not necessarily exist a delay-dominant strategy since delay-dominance is not solely defined on the satisfaction of φ but on the *structure* of an equivalent ACA $\mathcal{A}_\varphi$. In certain cases, $\mathcal{A}_\varphi$ can thus "punish" the delay-dominant strategy by introducing rejecting states at clever positions that do not influence acceptance but delay-dominance, preventing the existence of a delay-dominant strategy. However, we experienced that ACAs constructed with standard algorithms from LTL formulas do not punish delay-dominant strategies since such ACAs thoroughly follow the structure of the LTL formula and thus oftentimes do not contain unnecessary rejecting states. Simple optimizations such as removing states from the ACA that do not lie in a cycle from the set of rejecting states allow for delay-dominant strategies in even further cases while not altering the language of the automaton. Thus, we experienced that if an LTL formula φ allows for a remorsefree dominant strategy, then the ACA constructed from φ with standard algorithms allows for an delay-dominant strategy in many cases as well. Therefore, the compositional synthesis algorithm presented in this section is indeed applicable for many LTL formulas.

7 Conclusion

We have presented a new winning condition for process strategies, delay-dominance, that allows a strategy to violate a given specification in certain situations. In contrast to the classical notion of winning, delay-dominance can thus be used for individually synthesizing strategies for the processes in a distributed system in many cases, therefore enabling a simple compositional synthesis approach. Delay-dominance builds upon remorsefree dominance, where a strategy is allowed to violate the specification as long as no other strategy would have satisfied it in the same situation. However, remorsefree dominance is only compositional for safety properties. For liveness properties, the parallel composition of dominant strategies is not necessarily dominant. This restricts the use of dominance-based compositional synthesis algorithms to safety specifications, which are often not expressive enough. Delay-dominance, in contrast, is specifically designed to be compositional for liveness properties as well. We have introduced a game-based definition of delay-dominance and have proven compositionality for both safety and liveness specifications. Moreover, every delay-dominant strategy is also remorsefree dominant, and, for realizable system specifications, the parallel composition of delay-dominant strategies for all system processes is guaranteed to be winning for the whole system. Hence, delay-dominance is a suitable notion for compositional synthesis algorithms. We have introduced an automaton construction for recognizing delay-dominant strategies. The resulting universal co-Büchi automaton can immediately be used to synthesize delay-dominant strategies utilizing existing bounded synthesis approaches. It is of single-exponential size in the squared length of the specification. Thus, synthesizing delay-dominant strategies is, as for winning and remorsefree ones, in 2EXPTIME.

References

1 Shaull Almagor and Orna Kupferman. Good-Enough Synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

2 Benjamin Aminof, Giuseppe De Giacomo, Aniello Murano, and Sasha Rubin. Synthesis under Assumptions. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, pages 615–616. AAAI Press, 2018.

3 Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Best-Effort Synthesis: Doing Your Best is Not Harder Than Giving Up. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1766–1772. ijcai.org, 2021. doi:10.24963/ijcai.2021/243.

4 Jan E. Baumeister. Encodings of Bounded Synthesis for Distributed Systems. Bachelor's Thesis, Saarland University, 2017.

5 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to Handle Assumptions in Synthesis. In Krishnendu Chatterjee, Rüdiger Ehlers, and Susmit Jha, editors, *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014*, volume 157 of *EPTCS*, pages 34–50, 2014. doi:10.4204/EPTCS.157.7.

6 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-Admissible Synthesis. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 100–113. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.100.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment Assumptions for Synthesis. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008. doi:10.1007/978-3-540-85361-9_14.

8 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Complexity of Rational Synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.121.

9 Werner Damm and Bernd Finkbeiner. Does It Pay to Extend the Perimeter of a World Model? In Michael J. Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, volume 6664 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2011. doi:10.1007/978-3-642-21437-0_4.

10 Werner Damm and Bernd Finkbeiner. Automatic Compositional Synthesis of Distributed Systems. In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2014. doi:10.1007/978-3-319-06410-9_13.

11 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of Bounded Synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017. doi:10.1007/978-3-662-54577-5_20.

12 Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. BoSy: An Experimentation Framework for Bounded Synthesis. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 325–332. Springer, 2017. doi:10.1007/978-3-319-63390-9_17.

13 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Compositional Algorithms for LTL Synthesis. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*, volume 6252 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2010. doi:10.1007/978-3-642-15643-4_10.

14 Bernd Finkbeiner, Gideon Geier, and Noemi Passing. Specification Decomposition for Reactive Synthesis. In Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, editors, *NASA Formal Methods - 13th International Symposium, NFM 2021, Virtual Event, May 24-28, 2021, Proceedings*, volume 12673 of *Lecture Notes in Computer Science*, pages 113–130. Springer, 2021. doi:10.1007/978-3-030-76384-8_8.

15 Bernd Finkbeiner, Gideon Geier, and Noemi Passing. Specification decomposition for for reactive synthesis. *Innovations Syst. Softw. Eng.*, 2022. doi:10.1007/s11334-022-00462-6.

16 Bernd Finkbeiner and Noemi Passing. Dependency-Based Compositional Synthesis. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020,* *Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 2020. doi:10.1007/978-3-030-59152-6_25.

17 Bernd Finkbeiner and Noemi Passing. Compositional Synthesis of Modular Systems. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 303–319. Springer, 2021. doi:10.1007/978-3-030-88885-5_20.

18 Bernd Finkbeiner and Noemi Passing. Compositional synthesis of modular systems. *Innov. Syst. Softw. Eng.*, 18(3):455–469, 2022. doi:10.1007/s11334-022-00450-w.

19 Bernd Finkbeiner and Noemi Passing. Synthesizing Dominant Strategies for Liveness (Full Version). *CoRR*, abs/2210.01660, 2022. arXiv:2210.01660, doi:10.48550/arXiv.2210.01660.

20 Bernd Finkbeiner and Sven Schewe. Uniform Distributed Synthesis. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 321–330. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.53.

21 Bernd Finkbeiner and Sven Schewe. SMT-Based Synthesis of Distributed Systems. In *Proc. AFM*, 2007.

22 Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.*, 15(5-6):519–539, 2013. doi:10.1007/s10009-012-0228-z.

23 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational Synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

24 Carsten Fritz and Thomas Wilke. Simulation Relations for A;ternating Büchi Automata. *Theor. Comput. Sci.*, 338(1-3):275–314, 2005. doi:10.1016/j.tcs.2005.01.016.

25 Marcin Jurdzinski. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.

26 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with Rational Environments. In Nils Bulling, editor, *Multi-Agent Systems - 12th European Conference, EUMAS 2014, Prague, Czech Republic, December 18-19, 2014, Revised Selected Papers*, volume 8953 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2014. doi:10.1007/978-3-319-17130-2_15.

27 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safraless Compositional Synthesis. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2006. doi:10.1007/11817963_6.

28 Orna Kupferman and Moshe Y. Vardi. Safraless Decision Procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.

29 Yong Li, Andrea Turrini, Moshe Y. Vardi, and Lijun Zhang. Synthesizing Good-Enough Strategies for LTLf Specifications. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4144–4151. ijcai.org, 2021. doi:10.24963/ijcai.2021/570.

30 Satoru Miyano and Takeshi Hayashi. Alternating Finite Automata on ω-Words. *Theor. Comput. Sci.*, 32:321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

31 Amir Pnueli. The Temporal Logic of Programs. In *Annual Symposium on Foundations of Computer Science, 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

32 Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

33 Amir Pnueli and Roni Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.