Check for updates

Synthesizing Permissive Winning Strategy Templates for Parity Games

Ashwani Anand, Satya Prakash Nayak(✉), and Anne-Kathrin Schmuck

Max Planck Institute for Software Systems, Kaiserslautern, Germany
{ashwani,sanayak,akschmuck}@mpi-sws.org

CAV Artifact Evaluation ★ Available

Abstract. We present a novel method to compute *permissive winning strategies* in two-player games over finite graphs with ω-regular winning conditions. Given a game graph G and a parity winning condition Φ, we compute a *winning strategy template* Ψ that collects an infinite number of winning strategies for objective Φ in a concise data structure. We use this new representation of sets of winning strategies to tackle two problems arising from applications of two-player games in the context of cyber-physical system design – (i) *incremental synthesis*, i.e., adapting strategies to newly arriving, *additional* ω-regular objectives Φ', and (ii) *fault-tolerant control*, i.e., adapting strategies to the occasional or persistent unavailability of actuators. The main features of our strategy templates – which we utilize for solving these challenges – are their easy computability, adaptability, and compositionality. For *incremental synthesis*, we empirically show on a large set of benchmarks that our technique vastly outperforms existing approaches if the number of added specifications increases. While our method is not complete, our prototype implementation returns the full winning region in all 1400 benchmark instances, i.e. handling a large problem class efficiently in practice.

1 Introduction

Two-player ω-regular games on finite graphs are an established modeling and solution formalism for many challenging problems in the context of correct-by-construction cyber-physical system (CPS) design [2,7,39]. Here, control software actuating a technical system "plays" against the physical environment. The winning strategy of the system player in this two-player game results in software which ensures that the controlled technical system fulfills a given temporal specification for any (possible) event or input sequence generated by the environment. Examples include warehouse robot coordination [36], reconfigurable manufacturing systems [26], and adaptive cruise control [33]. In these applications, the

S. P. Nayak and A.-K. Schmuck are supported by the DFG project 389792660 TRR 248–CPEC.

A. Anand and A.-K. Schmuck are supported by the DFG project SCHM 3541/1-1.

© The Author(s) 2023

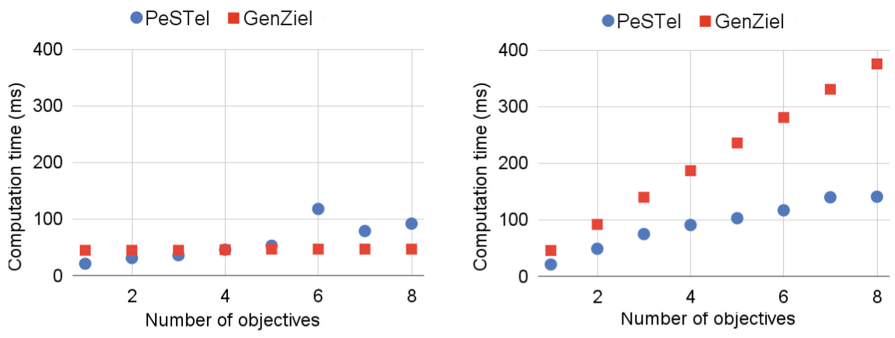


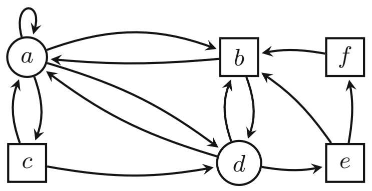
Fig. 1. Experimental results over 1400 generalized parity games comparing the performance of our tool PeSTel against the state-of-the-art generalized parity solver GenZiel [16]. Data points give the average execution time (in ms) over all instances with the same number of parity objectives. Left: all objectives are given *upfront*. Right: objectives are added *one-by-one*. See Sect. 6 for more details on those experiments.

technical system under control, as well as its requirements, are developing and changing during the design process. It is therefore desirable to allow for maintainable and adaptable control software. This, in turn, requires solution algorithms for two-player ω-regular games which allow for this adaptability.

This paper addresses this challenge by providing a new algorithm to efficiently compute *permissive winning strategy templates* in parity games which enable rich *strategy adaptations*. Given a game graph $G = (V, E)$ and an objective Φ a winning strategy template Ψ characterizes the winning region $\mathcal{W} \subseteq V$ along with three types of local edge conditions – a *safety*, a *co-live*, and a *live-group* template. The conjunction of these basic templates allows us to capture infinitely many winning strategies over G w.r.t. Φ in a simple data structure that is both (i) easy to obtain during synthesis, and (ii) easy to adapt and compose.

We showcase the usefulness of *permissive winning strategy templates* in the context of CPS design by two application scenarios: (i) *incremental synthesis*, where strategies need to be adapted to newly arriving *additional* ω-regular objectives Φ', and (ii) *fault-tolerant control*, where strategies need to be adapted to the occasional or persistent unavailability of actuators, i.e., system player edges.

We have implemented our algorithms in a prototype tool PeSTel and run it on more than 1400 benchmarks adapted from the SYNTCOMP benchmark suite [21]. These experiments show that our class of templates effectively avoids re-computations for the required strategy adaptations. For *incremental synthesis*, our experimental results are previewed in Fig. 1, where we compare PeSTel against the state-of-the-art solver GenZiel [16] for generalized parity objectives, i.e., finite conjunction of parity objectives. We see that PeSTel is as efficient as GenZiel whenever all conjuncts of the objective are given *up-front* (Fig. 1(left)) - even outperforming it in more than 90% of the instances. Whenever conjuncts of the objective arrive *one at a time*, PeSTel outperforms the existing approaches significantly if the number of objectives increases (Fig. 1(right)). This shows the potential of PeSTel towards more adaptable and maintainable control software for CPS.



$\Phi_1 = \square\neg\{f\} \Rightarrow \Psi_1 = \Psi_{\text{UNSAFE}}(e_{de})$
$\Phi_2 = \square\lozenge\{c,d\} \Rightarrow \Psi_2 = \Psi_{\text{LIVE}}(\{e_{ac}, e_{ad}\})$
$\Phi_3 = \lozenge\square\neg\{b\} \Rightarrow \Psi_3 = \Psi_{\text{COLIVE}}(e_{ab}, e_{db}, e_{de})$

Fig. 2. A two-player game graph with Player 1 (squares) and Player 0 (circles) vertices, different winning conditions Φ_i, and corresponding winning strategy templates Ψ_i.

Illustrative Example. To appreciate the simplicity and easy adaptability of our strategy templates, consider the game graph in Fig. 2(left). The first winning condition Φ_1 requires vertex f to never be seen along a play. This can be enforced by Player 0 from vertices $\mathcal{W}_0 = \{a, b, c, d\}$ called the *winning region*. The safety template Ψ_1 ensures that the game always stays in $\mathcal{W}_0$ by forcing the edge e_{de} to never be taken. It is easy to see that every Player 0 strategy that follows this rule results in plays which are winning if they start in $\mathcal{W}_0$. Now consider the second winning condition Φ_2 which requires vertex c or d to be seen infinitely often. This induces the live-group template Ψ_2 which requires that whenever vertex a is seen infinitely often, either edge e_{ac} or edge e_{ad} needs to be taken infinitely often. It is easy to see that any strategy that complies with this edge-condition is winning for Player 0 from every vertex and there are infinitely many such compliant winning strategies. Finally, we consider condition Φ_3 requiring vertex b to be seen only finitely often. This induces the strategy template Ψ_3 which is a co-liveness template requiring that all edges from Player 0 vertices which unavoidably lead to b (i.e., e_{ab}, e_{bd}, and e_{de}) are taken only finitely often. We can now combine all templates into a new template $\Psi' = \Psi_1 \wedge \Psi_2 \wedge \Psi_3$ and observe that all strategies compliant with Ψ' are winning for $\Phi' = \Phi_1 \wedge \Phi_2 \wedge \Phi_3$.

In addition to their compositionality, strategy templates also allow for local strategy adaptations in case of edge unavailability faults. Consider again the game in Fig. 2 with the objective Φ_2. Suppose that Player 0 follows the strategy π: $a \mapsto d$ and $d \mapsto a$, which is compliant with Ψ_2. If the edge e_{ad} becomes unavailable, we would need to re-solve the game for the modified game graph $G' = (V, E \setminus \{e_{ad}\})$. However, given the strategy template Ψ_2 we see that the strategy π': $a \mapsto c$ and $d \mapsto a$ is actually compliant with Ψ_2 over G'. This allows us to obtain a new strategy without re-solving the game.

While these examples demonstrate the potential of templates for strategy adaptation, there exist scenarios where conflicts between templates or graph modifications arise, which require re-computations. Our empirical results, however, show that such conflicts rarely appear in practical benchmarks. This suggests that our technique can handle a large problem class efficiently in practice.

Related Work. The class of templates we use was introduced in [4] and utilized to represent environment assumptions that enable a system to fulfill its specifications in a cooperative setting. Contrary to [4], this paper uses the same class of templates to represent the system's winning strategies in a zero-sum setting.

While the computation of *permissive strategies* for the control of CPS is an established concept in the field of supervisory control[1] [14,42], it has also been addressed in reactive synthesis where the considered specification class is typically more expressive, e.g., Bernet et al. [8] introduce permissive strategies that encompass all the behaviors of positional strategies and Neider et al. [31] introduce permissiveness to subsume strategies that visit losing loops at most twice. Finally, Bouyer et al. [11] take a quantitative approach to measure the permissiveness of strategies, by minimizing the penalty of not being permissive. However, all these approaches are not optimized towards strategy adaptation and thereby typically fail to preserve enough behaviors to be able to effectively satisfy subsequent objectives. A notable exception is a work by Baier et al. [23]. While their strategy templates are more complicated and more costly to compute than ours, they are *maximally* permissive (i.e., capture *all* winning strategies in the game). However, when composing multiple objectives, they restrict templates substantially which eliminates many compositional solutions that our method retains. This results in higher computation times and lower result quality for incremental synthesis compared to our approach. As no implementation of their method is available, we could not compare both approaches empirically.

Even without the incremental aspect, synthesizing winning strategies for conjunctions of ω-regular objectives is known to be a hard problem – Chatterjee et al. [16] prove that the conjunction of even *two* parity objectives makes the problem NP-complete. They provide a generalization of Zielonka's algorithm, called GenZiel for generalized parity objectives (i.e., finite conjunction of parity objectives) which is compared to our tool PeSTel in Fig. 1. While PeSTel is (in contrast to GenZiel) not complete—i.e., there exist realizable synthesis problems for which PeSTel returns no solution—our prototype implementation returns the full winning region in all 1400 benchmark instances.

Fault-tolerant control is a well-established topic in control engineering [9], with recent emphasis on the logical control layer [19,30]. While most of this work is conducted in the context of supervisory control, there are also some approaches in reactive synthesis. While [29,32] considers the *addition* of "disturbance edges" and synthesizes a strategy that tolerates as many of them as possible, we look at the complementary problem, where edges, in particular system-player edges, disappear. To the best of our knowledge, the only algorithm that is able to tackle this problem without re-computation considers Büchi games [15]. In contrast, our method is applicable to the more expressive class of Parity games.

2 Preliminaries

Notation. We use $\mathbb{N}$ to denote the set of natural numbers including zero. Given two natural numbers $a, b \in \mathbb{N}$ with $a < b$, we use $[a;b]$ to denote the set $\{n \in \mathbb{N} \mid a \leq n \leq b\}$. For any given set $[a;b]$, we write $i \in_{\text{even}} [a;b]$ and $i \in_{\text{odd}} [a;b]$ as shorthand for $i \in [a;b] \cap \{0,2,4,\ldots\}$ and $i \in [a;b] \cap \{1,3,5,\ldots\}$

[1] See [18,28,37] for connections between supervisory control and reactive synthesis.

respectively. Given two sets A and B, a relation $R \subseteq A \times B$, and an element $a \in A$, we write $R(a)$ to denote the set $\{b \in B \mid (a, b) \in R\}$.

Languages. Let Σ be a finite alphabet. The notation Σ^* and Σ^ω respectively denote the set of finite and infinite words over Σ, and Σ^∞ is equal to $\Sigma^* \cup \Sigma^\omega$. For any word $w \in \Sigma^\infty$, w_i denotes the i-th symbol in w. Given two words $u \in \Sigma^*$ and $v \in \Sigma^\infty$, the concatenation of u and v is written as the word uv.

Game Graphs. A *game graph* is a tuple $G = \big(V = V^0 \uplus V^1, E\big)$ where (V, E) is a finite directed graph with *vertices* V and *edges* E, and $V^0, V^1 \subseteq V$ form a partition of V. Without loss of generality, we assume that for every $v \in V$ there exists $v' \in V$ s.t. $(v, v') \in E$. A *play* originating at a vertex v_0 is a finite or infinite sequence of vertices $\rho = v_0 v_1 \ldots \in V^\infty$.

Winning Conditions/Objectives. Given a game graph G, we consider winning conditions/objectives specified using a formula Φ in *linear temporal logic* (LTL) over the vertex set V, that is, we consider LTL formulas whose atomic propositions are sets of vertices V. In this case the set of desired infinite plays is given by the semantics of Φ which is an ω-regular language $\mathcal{L}(\Phi) \subseteq V^\omega$. Every game graph with an arbitrary ω-regular set of desired infinite plays can be reduced to a game graph (possibly with a different set of vertices) with an LTL winning condition, as above. The standard definitions of ω-regular languages and LTL are omitted for brevity and can be found in standard textbooks [6]. To simplify notation we use $e = (u, v)$ in LTL formulas as syntactic sugar for $u \wedge \bigcirc v$, with $\bigcirc$ as the LTL *next* operator. We further use a set of edges $E' = \{e_i\}_{i \in [0;k]}$ as atomic proposition to denote $\bigvee_{i \in [0;k]} e_i$.

Games and Strategies. A *two-player (turn-based) game* is a pair $\mathcal{G} = (G, \Phi)$ where G is a game graph and Φ is a *winning condition* over G. A strategy of Player i, $i \in \{0, 1\}$, is a function $\pi^i \colon V^* V^i \to V$ such that for every $\rho v \in V^* V^i$ holds that $\pi^i(\rho v) \in E(v)$. Given a strategy π^i, we say that the play $\rho = v_0 v_1 \ldots$ is *compliant* with π^i if $v_{k-1} \in V^i$ implies $v_k = \pi^i(v_0 \ldots v_{k-1})$ for all k. We refer to a play compliant with π^i and a play compliant with both π^0 and π^1 as a *π^i-play* and a *$\pi^0\pi^1$-play*, respectively. We collect all plays originating in a set S and compliant with π^i, (and compliant with both π^0 and π^1) in the sets $\mathcal{L}(S, \pi^i)$ (and $\mathcal{L}(S, \pi^0\pi^1)$, respectively). When $S = V$, we drop the mention of the set in the previous notation, and when S is singleton $\{v\}$, we simply write $\mathcal{L}(v, \pi^i)$ (and $\mathcal{L}(v, \pi^0\pi^1)$, respectively).

Winning. Given a game $\mathcal{G} = (G, \Phi)$, a play ρ in $\mathcal{G}$ is *winning for* Player 0, if $\rho \in \mathcal{L}(\Phi)$, and it is winning for Player 1, otherwise. A strategy π^i for Player i is *winning from a vertex* $v \in V$ if all plays compliant with π^i and originating from v are winning for Player i. We say that a vertex $v \in V$ is *winning for* Player i, if there exists a winning strategy π^i from v. We collect all winning vertices of Player i in the Player i *winning region* $\mathcal{W}_i \subseteq V$. We always interpret winning w.r.t. Player 0 if not stated otherwise.

Strategy Templates. Let π^0 be a Player 0 strategy and Φ be an LTL formula. Then we say π^0 *follows* Φ, denoted $\pi^0 \Vdash \Phi$, if for all π^0-plays ρ, ρ belongs to

$\mathcal{L}(\Phi)$, i.e. $\mathcal{L}(\pi^0) \subseteq \mathcal{L}(\Phi)$. We refer to a set $\Psi = \{\Psi_1, \ldots, \Psi_k\}$ of LTL formulas as *strategy templates* representing the set of strategies that follows $\Psi_1 \wedge \ldots \wedge \Psi_k$. We say a strategy template Ψ is *winning from a vertex* v for a game (G, Φ) if every Player 0 strategy following the template Ψ is winning from v. Moreover, we say a strategy template Ψ is *winning* if it is winning from every vertex in $\mathcal{W}_0$. In addition, we call Ψ *maximally permissive* for $\mathcal{G}$, if every Player 0 strategy π which is winning in $\mathcal{G}$ also follows Ψ. With slight abuse of notation, we use Ψ for the set of formulas $\{\Psi_1, \ldots, \Psi_k\}$, and the formula $\Psi_1 \wedge \ldots \wedge \Psi_k$, interchangeably.

Set Transformers. Let $G = (V = V^0 \uplus V^1, E)$ be a game graph, $U \subseteq V$ be a subset of vertices, and $a \in \{0, 1\}$ be the player index. Then

$$\mathsf{upre}_G(U) = \{v \in V \mid \forall (v, u) \in E.\ u \in U\} \tag{1}$$

$$\mathsf{cpre}^a_G(U) = \{v \in V^a \mid \exists (v, u) \in E.\ u \in U\} \cup \{v \in V^{1-a} \mid u \in \mathsf{upre}_G(U)\} \tag{2}$$

The universal predecessor operator $\mathsf{upre}_G(U)$ computes the set of vertices with all the successors in U and the controllable predecessor operator $\mathsf{cpre}^a_G(U)$ the vertices from which Player a can force visiting U in *exactly one* step. In the following, we introduce two types of attractor operators: $\mathsf{attr}^a_G(U)$ that computes the set of vertices from which Player a can force at least a single visit to U in *finitely many* steps, and the universal attractor $\mathsf{uattr}_G(U)$ that computes the set of vertices from which both players are forced to visit U. For the following, let $\mathsf{pre} \in \{\mathsf{upre}, \mathsf{cpre}^a\}$

$$\mathsf{pre}^1_G(U) = \mathsf{pre}_G(U) \cup U \qquad \mathsf{pre}^i_G(U) = \mathsf{pre}_G(\mathsf{pre}^{i-1}_G(U)) \cup \mathsf{pre}^{i-1}_G(U) \tag{3}$$

$$\mathsf{attr}^a_G(U) = \cup_{i \geq 1}\, \mathsf{cpre}^{a,i}_G(U) \qquad \mathsf{uattr}_G(U) = \cup_{i \geq 1}\, \mathsf{upre}^i_G(U) \tag{4}$$

3 Computation of Winning Strategy Templates

Given a 2-player game $\mathcal{G}$ with an objective Φ, the goal of this section is to compute a *strategy template* that characterizes a large class of winning strategies of Player 0 from a set of vertices $U \subseteq V$ in a local, permissive, and computationally efficient way. These templates are then utilized in Sect. 5.1 for computational synthesis. In particular, this section introduces three distinct template classes—safety templates (Sect. 3.1), live-group-templates (Sect. 3.2), and co-live-templates (Sect. 3.3) along with algorithms for their computation via safety, Büchi, and co-Büchi games, respectively. We then turn to general parity objectives which can be thought of as a sophisticated combination of Büchi and co-Büchi games. We show in Sect. 3.4 how the three introduced templates can be derived for a general parity objective by a suitable combination of the previously introduced algorithms for single templates. All presented algorithms have the same worst-case computation time as the standard algorithms solving the respective game. This shows that extracting strategy *templates* instead of 'normal' strategies does not incur an additional computational cost. We prove the soundness of the algorithms and discuss the complexities in the full version [5, Appendix A].

3.1 Safety Templates

We start the construction of strategy templates by restricting ourselves to games with a safety objective—i.e., $\mathcal{G} = (G, \Phi)$ with $\Phi := \Box U$ for some $U \subseteq V$. A winning play in a safety game never leaves $U \subseteq V$. It is well known that such games allow capturing *all* winning strategies by a simple local template which essentially only allows Player 0 moves from winning vertices to other winning vertices. This is formalized in our notation as a safety template as follows,

Theorem 1 ([8, Fact 7]). *Let $\mathcal{G} = (G, \Box U)$ be a safety game with winning region $\mathcal{W}_0$ and $S = \{(u, v) \in E \mid \big(u \in V^0 \cap \mathcal{W}_0\big) \wedge (v \notin \mathcal{W}_0)\}$. Then*

$$\Psi_{\text{UNSAFE}}(S) := \Box \textstyle\bigwedge_{e \in S} \neg e, \tag{5}$$

is a winning strategy template for the game $\mathcal{G}$ which is also maximally permissive.

It is easy to see that the computation of the safety template $\Psi_{\text{UNSAFE}}(S)$ reduces to computing the winning region $\mathcal{W}_0$ in the safety game $(G, \Box U)$ and extracting S. We refer to the edges in S as *unsafe edges* and we call this algorithm computing the set S as SafetyTemplate(G, U). Note that it runs in $\mathcal{O}(m)$ time, where $m = |E|$, as safety games are solvable in $\mathcal{O}(m)$ time.

3.2 Live-Group Templates

As the next step, we now move to simple liveness objectives which require a particular vertex set $I \subseteq V$ to be seen infinitely often. Here, winning strategies need to stay in the winning region (as before) but in addition always eventually need to make progress towards the vertex set I. We capture this required progress by *live-group templates*—given a group of edges $H \subseteq E$, we require that whenever a source vertex v of an edge in H is seen infinitely often, an edge $e \in H$ (not necessarily starting at v) also needs to be taken infinitely often. This template ensures that compliant strategies always eventually make progress towards I, as illustrated by the following example.

Example 1. Consider the game graph in Fig. 2 where we require visiting $\{c, d\}$ infinitely often. To satisfy this objective from vertex a, Player 0 needs to not get stuck at a, and should not visit b always (since Player 1 can force visiting a again, and stop Player 0 from satisfying the objective). Hence, Player 0 has to always eventually leave a and go to $\{c, d\}$. This can be captured by the live-group $\{e_{ac}, e_{ad}\}$. Now if the play comes to a infinitely often, Player 0 will go to either c or d infinitely often, hence satisfying the objective.

Formally, such games are called *Büchi games*, denoted by $\mathcal{G} = (G = (V, E), \Phi)$ with $\Phi := \Box\Diamond I$, for some $I \subseteq V$. In addition, a *live-group* $H = \{e_j\}_{j \geq 0}$ is a set of edges $e_j = (s_j, t_j)$ with source vertices $src(H) := \{s_j\}_{j \geq 0}$. Given a set of live-groups $\mathcal{H} = \{H_i\}_{i \geq 0}$ we define a live-group template as

$$\Psi_{\text{LIVE}}(\mathcal{H}) := \bigwedge_{i \geq 0} \Box\Diamond src(H_i) \implies \Box\Diamond H_i. \tag{6}$$

Algorithm 1. BüchiTemplate(G, I)

```
Input: A game graph $G$, and a subset of vertices $I$
Output: A set of unsafe edges $S$ and a set of live-groups $\mathcal{H}$
1: $\mathcal{W}_0 \leftarrow$ Büchi$(G, I)$; $S \leftarrow$ SafetyTemplate$(G, \mathcal{W}_0)$;
2: $G \leftarrow G|_{\mathcal{W}_0}$; $I \leftarrow I \cap \mathcal{W}_0$;
3: $\mathcal{H} \leftarrow$ ReachTemplate$(G, I)$;
4: return $(S, \mathcal{H})$
5: procedure ReachTemplate$(G, I \subseteq V)$
6:     $\mathcal{H} \leftarrow \emptyset$;
7:     while $I \neq V$ do
8:         $A \leftarrow \mathsf{uattr}_G(I)$; $B \leftarrow \mathsf{cpre}_G^0(A)$; $\mathcal{H} \leftarrow \mathcal{H} \cup \{$Edges$(B, A)\}$; $I \leftarrow A \cup B$;
9:     return $\mathcal{H}$
```

The live-group template says that if some vertex from the source of a live-group is visited infinitely often, then some edge from this group should be taken infinitely often by the following strategy.

Intuitively, winning strategy templates for Büchi games consist of a safety template conjuncted with a live-group template. While the former enforces all strategies to stay within the winning region $\mathcal{W}$, the latter enforces progress w.r.t. the goal set I within $\mathcal{W}$. Therefore, the computation of a winning strategy template for Büchi games reduces to the computation of the unsafe set S to define $\Psi_{\text{UNSAFE}}(S)$ in (5) and the live-group $\mathcal{H}$ to define $\Psi_{\text{LIVE}}(\mathcal{H})$ in (6). We denote by BüchiTemplate(G, I) the algorithm computing the above as detailed in Algorithm 1. The algorithm uses some new notations that we define here. Here, the function Büchi solves a Büchi game and returns the winning region (e.g., using the standard algorithm from [17]), Edges$(X, Y) = \{(u, v) \in E \mid u \in X, v \in Y\}$, is the set of edges between two subsets of vertices X and Y. $G|_U \coloneqq \big(U = U^0 \uplus U^1, E'\big)$ s.t. $U^0 \coloneqq V^0 \cap U$, $U^1 \coloneqq V^1 \cap U$, and $E' \coloneqq E \cap (U \times U)$ denotes the restriction of a game graph $G \coloneqq \big(V = V^0 \uplus V^1, E\big)$ to a subset of its vertices $U \subseteq V$. We have the following formal result.

Theorem 2. *Given a Büchi game $\mathcal{G} = (G, \Box\Diamond I)$ for some $I \subseteq V$, if $(S, \mathcal{H}) =$ BüchiTemplate(G, I) then $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{LIVE}}(\mathcal{H})\}$ is a winning strategy template for the game $\mathcal{G}$, computable in time $\mathcal{O}(nm)$, where $n = |V|$ and $m = |E|$.*

While live-group templates capture infinitely many winning strategies in Büchi games, they are *not* maximally permissive, as exemplified next.

Example 2. Consider the game graph in Fig. 2 restricted to the vertex set $\{a, b, d\}$ with the Büchi objective $\Box\Diamond d$. Our algorithm outputs the live-group template $\Psi = \Psi_{\text{LIVE}}(\{e_{ad}\})$. Now consider the winning strategy with memory that takes edge e_{da} from d, and takes e_{ab} for play suffix bda and e_{ad} for play suffix aba. This strategy does not follow the template—the play $(abd)^\omega$ is in $\mathcal{L}(\pi^0)$ but not in $\mathcal{L}(\Psi)$.

3.3 Co-live Templates

We now turn to yet another objective which is the dual of the one discussed before. The objective requires that eventually, only a particular subset of vertices I is seen. A winning strategy for this objective would try to restrict staying or going away from I after a finite amount of time. It is easy to notice that live-group templates can not ensure this, but it can be captured by *co-live templates*: given a set of edges, eventually these edges are not taken anymore. Intuitively, these are the edges that take or keep a play away from I.

Example 3. Consider the game graph in Fig. 2 where we require eventually stop visiting b, i.e. staying in $I = \{a, c, d\}$. To satisfy this objective from vertex a, Player 0 needs to stop getting out of I eventually. Hence, Player 0 has to stop taking the edges $\{e_{ab}, e_{db}, e_{de}\}$, which can be ensured by marking both edges co-live. Now since no edges are leading to b, the play eventually stays in I, satisfying the objective. We note that this can not be captured by live-groups $\{e_{aa}, e_{ac}, e_{ad}\}$ and $\{e_{da}\}$, since now the strategy that visits c and b alternatively from Player 0's vertices, does not satisfy the objective, but follows the live-group.

Formally, a co-Büchi game is a game $\mathcal{G} = (G, \Phi)$ with co-Büchi winning condition $\Phi := \Diamond\Box I$, for some goal vertices $I \subseteq V$. A play is winning for Player 0 in such a co-Büchi game if it eventually stays in I forever. The *co-live* template is defined by a set of *co-live* edges D as follows,

$$\Psi_{\text{COLIVE}}(D) := \bigwedge_{e\in D} \Diamond\Box\neg e.$$

The intuition behind the winning template is that it forces staying in the winning region using the safety template, and ensures that the play does not go away from the vertex set I infinitely often using the co-live template. We provide the procedure in Algorithm 2 and its correctness in the following theorem. Here, $\textsc{CoBüchi}(G, I)$ is a standard algorithm solving the co-Büchi game with the goal vertices I, and outputs the winning regions for both players [17]. We also use the standard algorithm $\textsc{Safety}(G, I)$ that solves the safety game with the objective to stay in A forever.

Theorem 3. *Given a co-Büchi game $\mathcal{G} = (G, \Diamond\Box I)$ for some $I \subseteq V$, if $(S, D) = \textsc{coBüchiTemplate}(G, I)$ then $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{COLIVE}}(D)\}$ is a winning strategy template for Player 0, computable in time $\mathcal{O}(nm)$ with $n = |V|$ and $m = |E|$.*

3.4 Parity Games

We now consider a more complex but canonical class of ω-regular objectives. Parity objectives are of central importance in the study of synthesis problems as they are general enough to model a huge class of qualitative requirements of cyber-physical systems, while enjoying the properties like positional determinacy.

Algorithm 2. coBüchiTemplate(G, I)

```
Input: A game graph $G$, and a subset of vertices $I$
Output: A set of unsafe edges $S$ and a set of co-live edges $D$
 1: $S \leftarrow \emptyset$; $D \leftarrow \emptyset$
 2: $\mathcal{W}_0 \leftarrow$ CoBüchi$(G, I)$; $S \leftarrow$ SafetyTemplate$(G, \mathcal{W}_0)$
 3: $G \leftarrow G|_{\mathcal{W}_0}$; $I \leftarrow I \cap \mathcal{W}_0$;
 4: while $V \neq \emptyset$ do
 5:     $A \leftarrow$ Safety$(G, I)$; $D \leftarrow D \cup$ Edges$(A, V \setminus A)$;
 6:     while $\mathsf{cpre}^0_G(A) \neq A$ do                    ▷ Outputs $\mathsf{attr}^0_G(A)$
 7:         $B \leftarrow \mathsf{cpre}^0_G(A)$;
 8:         $D \leftarrow D \cup$ Edges$(B, V \setminus (A \cup B)) \cup$ Edges$(B, B)$;
 9:         $A \leftarrow A \cup B$;
10:     $G \leftarrow G|_{V \setminus A}$; $I \leftarrow I \cap V \setminus A$;
11: return $(S, D)$
```

A parity game is a game $\mathcal{G} = (G, \Phi)$ with parity winning condition $\Phi = \mathit{Parity}(\mathbb{P})$, where

$$\mathit{Parity}(\mathbb{P}) := \bigwedge_{i \in \text{odd}[0;k]} \left(\Box\Diamond P_i \implies \bigvee_{j \in \text{even}[i+1;k]} \Box\Diamond P_j \right), \tag{7}$$

with $P_i = \{q \in Q \mid \mathbb{P}(q) = i\}$ for some priority function $\mathbb{P} : V \to [0; d]$ that assigns each vertex a priority. A play is winning for Player 0 in such a game if the maximum of priorities seen infinitely often is even.

Although parity objectives subsume previously described objectives, we can construct strategy templates for parity games using the combinations of previously defined templates. To this end, we give the following algorithm.

Theorem 4. *Given a parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ with priority function $\mathbb{P} : V \to [0; d]$, if $((\mathcal{W}_0, \mathcal{W}_1), \mathcal{H}, D) =$ ParityTemplate$(G, \mathbb{P})$, then $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{LIVE}}(\mathcal{H}), \Psi_{\text{COLIVE}}(D)\}$ is a winning strategy template for the game $\mathcal{G}$, where $S =$ Edges$(\mathcal{W}_0, \mathcal{W}_1)$. Moreover, the algorithm terminates in time $\mathcal{O}(n^{d+\mathcal{O}(1)})$, which is same as that of Zielonka's algorithm.*

We refer the readers to the full version [5, Appendix A.3] for the complete proofs, and here we provide the intuition behind the algorithm and the computation of the algorithm on the parity game in Fig. 3. The algorithm follows the divide-

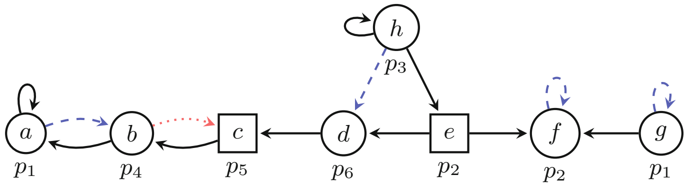


Fig. 3. A parity game, where a vertex with priority i has label p_i. The dotted edge in red is a co-live edge, while the dashed edges in blue are singleton live-groups. (Color figure online)

Algorithm 3. ParityTemplate($G, \mathbb{P}$)

```
Input: A game graph $G$, and a priority function $\mathbb{P} : V \to \{0, \ldots, d\}$
Output: Winning regions $(\mathcal{W}_0, \mathcal{W}_1)$, live-groups $\mathcal{H}$, and co-live edges $D$
 1: if $d$ is odd then
 2:     $A = \mathsf{attr}^1_G(P_d)$
 3:     if $A = V$ then return $(\emptyset, V), \emptyset, \emptyset$
 4:     else
 5:         $(\mathcal{W}_0, \mathcal{W}_1), \mathcal{H}, D \leftarrow$ ParityTemplate($G|_{V\setminus A}, \mathbb{P}$)
 6:         if $\mathcal{W}_0 = \emptyset$ then return $(\emptyset, V), \emptyset, \emptyset$
 7:         else
 8:             $B = \mathsf{attr}^0_G(\mathcal{W}_0)$
 9:             $D \leftarrow D \cup$ Edges($\mathcal{W}_0, V \setminus \mathcal{W}_0$)
10:             $\mathcal{H} \leftarrow \mathcal{H} \cup$ ReachTemplate($G, \mathcal{W}_0$)
11:             $(\mathcal{W}'_0, \mathcal{W}'_1), \mathcal{H}', D' \leftarrow$ ParityTemplate($G|_{V\setminus B}, \mathbb{P}$)
12:             return $(\mathcal{W}'_0 \cup B, \mathcal{W}'_1), \mathcal{H} \cup \mathcal{H}', D \cup D'$
13: else                                                        ▷ If $d$ is even
14:     $A = \mathsf{attr}^0_G(P_d)$
15:     if A=V then return $(V, \emptyset)$, ReachTemplate($G, P_d$), $\emptyset$
16:     else
17:         $(\mathcal{W}_0, \mathcal{W}_1), \mathcal{H}, D \leftarrow$ ParityTemplate($G|_{V\setminus A}, \mathbb{P}$)
18:         if $\mathcal{W}_1 = \emptyset$ then return $(V, \emptyset), \mathcal{H} \cup$ ReachTemplate($G|_A, P_d$), $D$
19:         else
20:             $B = \mathsf{attr}^1_G(\mathcal{W}_1)$
21:             $(\mathcal{W}'_0, \mathcal{W}'_1), \mathcal{H}', D' \leftarrow$ ParityTemplate($G|_{V\setminus B}, \mathbb{P}$)
22:             return $(\mathcal{W}'_0, \mathcal{W}'_1 \cup B), \mathcal{H}', D'$
```

and-conquer approach of Zeilonka's algorithm. Since the highest priority occurring is 6 which is even, we first find the vertices $A = \{d, h\}$ from which Player 0 can force visiting $\{d\}$ (vertices with priority 6) in line 14. Then since $A \neq V$, we find the winning strategy template in the rest of the graph $G_1 = G|_{V\setminus A}$. Then the highest priority 5 is odd, hence we compute the region $\{c\}$ from which Player 1 can ensure visiting 5. We again restrict our graph to $G_2 = G|_{\{a,b,e,f,g\}}$. Again, the highest priority is even. We further compute the region $A_2 = \{a, b\}$ from which Player 0 can ensure visiting the priority 4, giving us $G_3 = G|_{\{e,f,g\}}$. In G_3, Player 0 can ensure visiting the highest priority 2, hence satisfying the condition in line 15. Then since in this small graph, Player 0 needs to keep visiting priority 2 infinitely often, which gives us the live-groups $\{e_{gf}\}$ and $\{e_{ff}\}$ in line 15. Coming one recursive step back to G_2, since G_3 doesn't have a winning vertex for Player 1, the if condition in the line 18 is satisfied. Hence, for the vertices in A_2, it suffices to keep visiting priority 4 to win, which is ensured by the live-group $\{e_{ab}\}$ added in the line 18. Now, again going one recursive step back to G_1, we have $\mathcal{W}_0 = \{a, b, e, f, g\}$. If Player 0 can ensure reaching and staying in $\mathcal{W}_0$ from the rest of the graph G_1, it can satisfy the parity condition. Since from the vertex c, $\mathcal{W}_0$ will anyway be reached, we get a co-live edge e_{bc} in line 9 to eventually keep the play in $\mathcal{W}_0$. Coming back to the initial recursive call, since now again G_1 was winning for Player 0, they only need to be able to visit the priority 6 from every vertex in A, giving another live-group $\{e_{hd}\}$.

4 Extracting Strategies from Strategy Templates

This section discusses how a strategy that follows a computed winning strategy template can be extracted from the template. As our templates are just particular LTL formulas, one can of course use automata-theoretic techniques for this. However, as the types of templates we presented put some local restrictions on strategies, we can extract a strategy much more efficiently. For instance, the game in Fig. 2 with strategy template $\Psi = \Psi_{\text{LIVE}}(\{e_{ac}, e_{ad}\})$ allows the strategy that simply uses the edges e_{ac} and e_{ad} alternatively from vertex a.

However, strategy extraction is not as straightforward for every template, even if it only conjuncts the three template types we introduced in Sect. 3. For instance, consider again the game graph from Fig. 2 with a strategy template $\Psi = \{\Psi_{\text{UNSAFE}}(e_{ac}, e_{ad}), \Psi_{\text{COLIVE}}(e_{aa}, e_{ab})\}$. Here, non of the four choices of Player 0 (i.e., outgoing edges) from vertex a can be taken infinitely often, and, hence, the only way a play satisfies Ψ is to not visit vertex a infinitely often. On the other hand, given strategy template $\Psi' = \{\Psi_{\text{COLIVE}}(e_{ab}, e_{db}), \Psi_{\text{LIVE}}(\{e_{ab}, e_{ac}, e_{db}\})\}$, edge e_{db} is both live and co-live, which raises a conflict for vertex d. Hence, the only way a strategy can follow Ψ' is again to ensure that d is not visited infinitely often. We call such situations *conflicts*. Interestingly, the methods we presented in Sect. 3 never create such conflicts and the computed templates are therefore *conflict-free*, as formalized next and proven in the full version [5, Appendix A.4].

Definition 1. *A strategy template $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{COLIVE}}(D), \Psi_{\text{LIVE}}(\mathcal{H})\}$ in a game graph $G = (V, E)$ is* conflict-free *if the following are true:*

(i) or every vertex v, there is an outgoing edge that is neither co-live nor unsafe, i.e., $v \times E(v) \nsubseteq D \cup S$, and
(ii) for every source vertex v in a live-group $H \in \mathcal{H}$, there exists an outgoing edge in H which is neither co-live nor unsafe, i.e., $v \times H(v) \nsubseteq D \cup S$.

Proposition 1. *Algorithms 1, 2, and 3 always return conflict-free templates.*

Due to the given conflict-freeness, winning strategies are indeed easy to extract from winning strategy templates, as formalized next.

Proposition 2. *Given a game graph $G = (V, E)$ with* conflict-free *winning strategy template $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{COLIVE}}(D), \Psi_{\text{LIVE}}(\mathcal{H})\}$, a winning strategy π_0 that follows Ψ can be extracted in time $\mathcal{O}(m)$, where m is the number of edges.*

The proof is straightforward by constructing the winning strategy as follows. We first remove all unsafe and co-live edges from G and then construct a strategy π_0 that alternates between all remaining edges from every vertex in $\mathcal{W}_0$. This strategy is well defined as condition (i) in Definition 1 ensures that after removing all the unsafe and co-live edges a choice from every vertex remains. Moreover, if the vertex is a source of a live-group edge, condition (ii) in Definition 1 ensures that there are outgoing edges satisfying every live-group. It is easy to see that the constructed strategy indeed follows Ψ and is hence winning from vertices in $\mathcal{W}_0$, as Ψ was a winning strategy template. We call this procedure of strategy extraction $\textsc{ExtractStrategy}(G, \Psi)$.

5 Applications of Strategy Templates

This section considers two concrete applications of strategy templates which utilize their structural simplicity and easy adaptability.

In the context of CPS control design problems, it is well known that the game graph of the resulting parity game used for strategy synthesis typically has a physical interpretation and results from behavioral constraints on the *existing technical system* that is subject to control. In particular, following the well-established paradigm of abstraction-based control design (ABCD) [2,7,39], an underlying (stochastic) disturbed non-linear dynamical system can be automatically abstracted into a two-player game graph using standard abstraction tools, e.g. SCOTS [35], ARCS [13], MASCOT [20], P-FACES [22], or ROCS [27].

In contrast to classical problems in reactive synthesis, it is very natural in this context to think about the game graph and the specification as two *different* objects. Here, specifications are naturally expressed via propositions that are defined over sets of states of this underlying game graph, without changing its structure. This separation is for example also present in the known LTL fragment GR(1) [10]. Arguably, this feature has contributed to the success of GR(1)-based synthesis for CPS applications, e.g. [1,3,24,25,38,40,41].

Given this insight, it is natural to define the incremental synthesis problem such that the game graph stays unchanged, while newly arriving specifications are modeled as new parity conditions over the same game graph. Formally, this results in a *generalized parity game* where the different objectives arrive *one at a time*. We show an incremental algorithm for synthesizing winning strategies for such games in Sect. 5.1. Similarly, fault-tolerant control requires the controller to adapt to unavailable actuators within the technical system under control. This naturally translates to the removal of Player 0 edges within the game graph given its physical interpretation. We show how strategy templates can be used to adapt winning strategies to these game graph modifications in Sect. 5.2.

5.1 Incremental Synthesis via Strategy Templates

In this section we consider a 2-player game $\mathcal{G}$ with a conjunction $\Phi = \bigwedge_{i=1}^{k} \Phi_i$ of multiple parity objectives Φ_i, also called a *generalized* parity objective. However, in comparison to existing work [12,16], we consider the case that different objectives Φ_i might not arrive all at the same time. The intuition of our algorithm is to solve each parity game (G, Φ_i) separately and then combine the resulting strategy templates Ψ_i to a global template $\Psi = \bigwedge_{i=1}^{k} \Psi_i$. This allows to easily incorporate newly arriving objectives Φ_{k+1}. We only need to solve the parity game (G, Φ_{k+1}) and then combine the resulting template Ψ_{k+1} with Ψ.

While Proposition 1 ensures that every individual template Ψ_i is *conflict-free*, this does unfortunately not imply that their conjunction is also *conflict-free*. Intuitively, combinations of strategy templates can cause the condition (i) and (ii) in Definition 1 to not hold anymore, resulting in a *conflict*. As already discussed in Sect. 4, this requires source vertices $U \subseteq V$ with such conflicts to

Algorithm 4. ComposeTemplate$(G, (\mathcal{W}_0', \mathcal{H}', D', (\Phi_i)_{i<\ell}), (\Phi_i)_{\ell \leq i \leq k})$ where $\Phi_i = \mathit{Parity}(\mathbb{P}_i)$

Input: A generalized parity game $G = (V, E)$ and objectives $(\Phi_i)_{i \leq k}$ with $\Phi_i = \mathit{Parity}(\mathbb{P}_i)$ such that $\mathbb{P}_i : V \to [0; 2d_i + 1]$ along with a partial winning region, live-groups, and co-live edges $(\mathcal{W}_0, \mathcal{H}, D)$ for the generalized parity game $(G, \bigwedge_{i<\ell} \Phi_i)$.

Output: A partial winning region $\mathcal{W}_0$, live-groups $\mathcal{H}$, co-live edges D, and modified parity objectives $(\Phi_i')_{i \leq k}$.

```
1: $(W_i, V \setminus W_i), \mathcal{H}_i, D_i \leftarrow$ ParityTemplate$(G|_{\mathcal{W}_0}, \Phi_i)$ for each $\ell \leq i \leq k$
2: $\mathcal{H} = \mathcal{H}' \cup \bigcup_{\ell \leq i \leq k} \mathcal{H}_i$; $D = D' \cup \bigcup_{\ell \leq i \leq k} D_i$; $\mathcal{W}_0 = \mathcal{W}_0' \cap \bigcap_{\ell \leq i \leq k} W_i$
3: $\mathcal{C}_1 = \{u \in \mathcal{W}_0 \mid u \times (E(u) \cap \mathcal{W}_0) \subseteq D\}$
4: $\mathcal{C}_2 = \{u \in \mathcal{W}_0 \mid u \times (H(u) \cap \mathcal{W}_0) \subseteq D,\ H \in \mathcal{H},\ H(u) \neq \emptyset\}$
5: if $\mathcal{C}_1 \cup \mathcal{C}_2 = \emptyset$ then
6:     return $(\mathcal{W}_0, \mathcal{H}, D, (\Phi_i)_{i \leq k})$
7: else
8:     $\mathbb{P}_i'(u) \leftarrow \mathbb{P}[\mathcal{C}_1 \cup \mathcal{C}_2 \to 2d_i' + 1]$ for each $i \leq k$
9:     return ComposeTemplate$(G, (\mathcal{W}_0, \emptyset, \emptyset, \emptyset), (\Phi_i')_{i \leq k})$ with $\Phi_i' = \mathit{Parity}(\mathbb{P}_i')$)
```

eventually not be visited anymore. We therefore resolve such conflicts by adding the specification $\Diamond\Box\neg U$ to every objective and recomputing the templates.

To efficiently formalize this objective change, we note that a parity objective $\mathit{Parity}(\mathbb{P})$ with an additional specification $\Diamond\Box\neg U$ for some $U \subseteq V$ is equivalent to another parity objective $\mathit{Parity}(\mathbb{P}')$, where priority function $\mathbb{P}'$ can be obtained from $\mathbb{P} : V \to [0; 2d+1]$ just by modifying the priorities of vertices in U to $2d+1$. Let us denote such a priority function by $\mathbb{P}[U \to 2d+1]$. In particular, we have the following result:

Lemma 1. *Given a game graph G and two parity objectives $\Phi = \mathit{Parity}(\mathbb{P})$, $\Phi' = \mathit{Parity}(\mathbb{P}')$ such that $\mathbb{P} : V \to [0; 2d+1]$ and $\mathbb{P}' = \mathbb{P}[U \to 2d+1]$ for some vertex set $U \subseteq V$, it holds that $\mathcal{L}(\Phi') = \mathcal{L}(\Phi \wedge \Diamond\Box\neg U)$. Moreover, if a strategy template is winning from some vertex u in the game $\mathcal{G}' = (G, \Phi')$, then it is also winning from u in the game $\mathcal{G} = (G, \Phi)$.*

Using the above ideas, we present Algorithm 4 to solve generalized parity games (possibly incrementally). If no partial solution to the synthesis problem exists so far we have $\ell = 0$, otherwise the game $(G, \bigwedge_{i<\ell} \Phi_i)$ was already solved and the respective winning region and templates are known. In both cases, the algorithm starts with computing a winning strategy template for each game (G, Φ_i) for $i \in \{\ell + 1, k\}$ (line 1) and conjuncts them with the already computed ones (line 2). Then the algorithm checks for conflicts (line 3–4). If there is some conflict the algorithm modifies the objectives to ensure that the conflicted vertices are eventually not visited anymore (line 8), and then re-computes the templates in the game graph restricted to the intersection of winning regions for all objectives (line 9). If there is no conflict, then the algorithm returns the conjunction of the templates which is conflict-free, and hence, is winning from the intersection of winning regions for every objective (line 6). The latter is formalized in the following theorem. The proof can be found in the full version [5, Appendix B.2].

Theorem 5. *Given a generalized parity game* $\mathcal{G} = (G, \bigwedge_{i\leq k} \Phi_i)$ *with* $\Phi_i = \mathit{Parity}(\mathbb{P}_i)$ *and priority functions* $\mathbb{P}_i : V \to [0; 2d_i + 1]$*, if* $(\mathcal{W}_0, \mathcal{H}, D, (\Phi'_i)_{i\leq k}) = \textsc{ComposeTemplate}(G, \emptyset, (V, \emptyset, \emptyset), (\Phi_i)_{i\leq k})$*, then* $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{LIVE}}(\mathcal{H}), \Psi_{\text{COLIVE}}(D)\}$ *is an conflict-free strategy template that is winning from* $\mathcal{W}_0$ *in the game* $\mathcal{G}$*, where* $S = \textsc{Edges}(\mathcal{W}_0, V \setminus \mathcal{W}_0)$*. Further,* Ψ *is computable in time* $\mathcal{O}(kn^{2d+3})$ *time, where* $n = |V|$ *and* $d = \max_{i\leq k} d_i$*.*

Due to the conflict checks carried out within Algorithm 4 the returned modified objectives Φ'_i ensure that the *conjunction* $\Psi := \bigwedge_{i=1}^{k} \Psi'_i$ of winning strategy templates Ψ'_i for the games (G, Φ'_i) is indeed conflict-free. In particular, the conjuncted template Ψ is actually returned by the algorithm. Hence, incrementally running Algorithm 4 is actually sound. This is an immediate consequence of Theorem 5 and stated as a corollary next.

Corollary 1. *Given a generalized parity game* $\mathcal{G} = (G, \bigwedge_{i\leq k} \Phi_i)$ *with* $\Phi_i = \mathit{Parity}(\mathbb{P}_i)$ *and priority functions* $\mathbb{P}_i : V \to [0; 2d_i + 1]$*, s.t.*

$$(\mathcal{W}'_0, \mathcal{H}', D', (\Phi'_i)_{i<\ell}) := \textsc{ComposeTemplate}(G, (V, \emptyset, \emptyset, \emptyset), (\Phi_i)_{i<\ell}), \text{ and}$$
$$(\mathcal{W}_0, \mathcal{H}, D, (\Phi''_i)_{i\leq k}) := \textsc{ComposeTemplate}(G, (\mathcal{W}'_0, \mathcal{H}', D', (\Phi'_i)_{i<\ell}), (\Phi_i)_{\ell\leq i\leq k})$$

then $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{LIVE}}(\mathcal{H}), \Psi_{\text{COLIVE}}(D)\}$ *is an conflict-free strategy template that is winning from* $\mathcal{W}_0$ *in the game* $\mathcal{G}$*, where* $S = \textsc{Edges}(\mathcal{W}_0, V \setminus \mathcal{W}_0)$*. Further,* Ψ *is computable in time* $\mathcal{O}(kn^{2d+3})$*, where* $n = |V|$ *and* $d = \max_{i\leq k} d_i$*.*

We note that the generalized Zielonka algorithm [16] for solving generalized parity games has time complexity $\mathcal{O}(mn^{\sum 2d_i})\binom{\sum d_i}{d_1, d_2, \ldots, d_k}$ for a game with n vertices, m edges and k priority functions: $\mathbb{P}_i$ with $2d_i$ priorities for each i. Clearly, Algorithm 4 has a much better time complexity. However, it is not complete, i.e., it does not always return the complete winning region. This is due to templates being not maximally permissive and hence potentially raising conflicts which result in additional specifications that are not actually required. The next example shows such an incomplete instance for illustration. We however note that Algorithm 4 returned the *full winning region* on *all* benchmarks considered during evaluation, suggesting that such instances rarely occur in practice.

Example 4. Consider the game in Fig. 2 with objectives $\Phi_3 \wedge \Phi_4$ with $\Phi_4 = \mathit{Parity}(\mathbb{P})$, where $\mathbb{P}$ maps vertices a, b, c, d, e, f to $0, 2, 1, 1, 1, 1$, respectively. The winning strategy templates computed by ParityTemplate for objectives Φ_3 and Φ_4 are $\Psi_3 = \Psi_{\text{COLIVE}}(e_{ab}, e_{db}, e_{de})$ and $\Psi_4 = \Psi_{\text{LIVE}}(\{e_{ab}, e_{db}, e_{de}\})$, respectively. The conjunction of both templates marks all outgoing edges of vertex a and d in the live-group co-live. Hence, the algorithm would ensure that these conflicted vertices a and d are eventually not visited anymore. However, the only way to satisfy $\Phi_3 \wedge \Phi_4$ is by eventually looping on vertex a. But this solution was skipped by the strategy template Ψ_4 by putting edge e_{ab} in a live-group. Therefore, the algorithm returns the empty set as the winning region, whereas the actual winning region is the whole vertex set.

5.2 Fault-Tolerant Strategy Adaptation

In this section we consider a 2-player parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ and a set of faulty Player 0 edges $F \subseteq E \cap (V^0 \times V)$ which might become unavailable during runtime. Given a strategy template Ψ for $\mathcal{G}$, we can use $\Psi' = \{\Psi, \Psi_{\textsc{unsafe}}(F)\}$ for the (linear-time) extraction of a new strategy for the game, if Ψ' is conflict-free for G. In this case, no re-computation is needed. If Ψ' is not conflict-free for G, then we can remove the edges in F and compute a new winning strategy template using Algorithm 3. This is formalized in Algorithm 5, where we slightly abuse notation and assume that ParityTemplate only outputs strategy templates. The correctness of Algorithm 5 follows directly from Theorem 4.

Corollary 2. *Given a 2-player parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ with a strategy template $\Psi = \textsc{ParityTemplate}(G, \mathbb{P})$ and faulty edge set $F \subseteq E \cap (V^0 \times V)$ it holds that Ψ' obtained from Algorithm 5 is a winning strategy template for $\mathcal{G}|_{E \setminus F}$.*

Faulty edges introduce an additional safety specification for which our templates are maximally permissive. This implies that Algorithm 5 is *sound and complete* – if there exists a winning strategy for $(G|_{E\setminus F}, \mathit{Parity}(\mathbb{P}))$ Algorithm 5 finds one.

Let us now assume that F collects all edges controlling *vulnerable* actuators that *might* become unavailable. In this scenario, Algorithm 5 returns a conservative strategy that *never* uses vulnerable actuators. It might however be desirable to use actuators as long as they are available to obtain better performance. Formally, this application scenario can be defined via a time-dependent graph who's edges change over time, i.e., E_t with $E_0 = E$ are the edges available at time $t \in \mathbb{N}$ and $F := \{e \in E \mid e \notin E_i, \text{ for some } i\}$. Given the original parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ with a winning strategy template Ψ we can easily modify $\textsc{ExtractStrategy}(\mathcal{G}, \Psi)$ to obtain a time-dependent strategy π_g which reacts to the unavailability of edges, i.e., at time t, π_g takes an edge $e \in E_t \backslash (S \cup D)$ for all vertices without any live-group, and for the ones with live-groups, it alternates between the edges satisfying the live-groups whenever they are available, and an edge $e \in E_t \backslash (S \cup D)$ when no live-group edge is available.

The online strategy π_g can be implemented even without knowing when edges are available[2], i.e., without knowing the time dependent edge sequence $\{E_t\}_{t\in\mathbb{N}}$

Algorithm 5. $\textsc{FaultCorrection}(G, \Psi, F)$

Input: A parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$, a strategy template Ψ, and a set of faulty edges F
Output: A new strategy template Ψ'

```
1: $\Psi' \leftarrow \{\Psi, \Psi_{\textsc{unsafe}}(F)\}$
2: if $\textsc{CheckTemplate}(G, \Psi')$ then return $\Psi'$
3: else
4:     return $\textsc{ParityTemplate}(G|_{E\setminus F}, \mathbb{P}|_{E\setminus F})$
```

[2] We note that it is reasonable to assume that current actuator faults are visible to the controller at runtime, see e.g. [34] for a real water gate control example.

up front. In this case π_g is obviously winning in $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ if Ψ is conflict-free for $\mathcal{G}|_{E \setminus F}$. If this is not the case, one needs to ensure that edges that cause conflicts are always eventually available again, as formalized next.

Definition 2. *Given a parity game $\mathcal{G} = (G, \mathit{Parity}(\mathbb{P}))$ we call the dynamic edge set $\{E_i\}_{i \geq 0}$ a* guaranteed availability fault (GAF) *if $\forall$ plays $\rho = v_0 v_1 \ldots$, $\forall v \in V$, if $v \in \mathit{inf}(\rho)$, then $\forall e = (v, w) \in F$, $\exists$ infinitely many times $t_0, t_1 \ldots$ such that $v_{t_j} = v$ and $e \in E_{t_j}$, $\forall j \geq 0$.*

Intuitively, guaranteed availability faults (GAF) ensure that a faulty edge is always eventually available when a play is in its source vertex. Under this fault, the following fault-correction result holds, which is proven in the full version [5, Appendix B.3].

Proposition 3. *Given a game graph G with a parity objective Φ, a strategy template $\Psi = \{\Psi_{\text{UNSAFE}}(S), \Psi_{\text{LIVE}}(\mathcal{H}), \Psi_{\text{COLIVE}}(D)\}$ computed by Algorithm 3 and a set $F = \{e \in E \mid e \notin E_i,$ for some $i\}$ of faulty edges, the game with the objective is realizable under GAF if for every vertex $v \in V^0$, there is an outgoing edge which is not in $S \cup D \cup F$.*

This proposition allows a simple linear-time algorithm to check if the templates computed by Algorithm 3 are GAF-tolerant: check if every vertex in the winning region has an outgoing edge which is not in $S \cup D \cup F$. If this is not the case, the recomputation is non-trivial and is out of scope of this paper. We can however collect the vertices which do not satisfy the above property and alert the system engineer that these vulnerable actuators require additional maintenance or protective hardware. Our experimental results in Sect. 6 show that conflicts arising from actuator faults are rare and very local. Our strategy templates allow to easily localize them, which supports their use for CPS applications.

6 Empirical Evaluation

We have developed a C++-based prototype tool PeSTel[3] (computing **PE**rmissive **S**trategy **TE**mp**L**ates) that implements Algorithms 1–5. We have used PeSTel to show its superior performance on the two applications considered in Sect. 5, suggesting its practical relevance. All our experiments were performed on a computer equipped with Apple M1 Pro 8-core CPU and 16 GB RAM.

Incremental Synthesis. We used PeSTel to solve generalized parity games both in one shot and incremental. We compare our algorithm with existing algorithms, i.e., GenZiel from [16] and three partial solvers[4] from [12], by executing

[3] Repository URL: https://github.com/satya2009rta/pestel.

[4] While GenZiel is sound and complete [16], we found different randomly generated games where the algorithms from [12] either return a superset or a subset of the winning region, hence compromising soundness and completeness. Since [12] lacks rigorous proof, it is not clear whether this is an implementation bug or a theoretical mishap, leaving soundness and completeness guarantees of these algorithms open.

Table 1. Aggregated experimental results on generalize parity game benchmarks with objectives given *up-front* (top) and *one-by-one* (bottom). Subrows: 1st row (mean time) – average computation time (in ms); 2nd row (incomplete) – number of examples where the corresponding tool failed to compute the complete winning region; 3rd row (faster than) – number of examples where PeSTel is faster than the respective tool; 4th row (timeouts) – number of examples where the respective tool timed out (10000 ms).

		PeSTel	GenZiel [16]	GenZiel & GenBüchi [12]	GenZiel & GenGoodEp[12]	GenZiel & GenLay[12]
Benchmark A (one shot)	mean time	343	64	68	553	1224
	incomplete	0	-	3	3	2
	faster than	-	74%	75%	96%	85%
	timeouts	0	0	0	2	20
Benchmark B (one shot)	mean time	60	47	58	112	171
	incomplete	0	-	28	27	2
	faster than	-	93%	93%	97%	95%
	timeouts	1	0	2	4	18
Overall	faster than	-	90%	90%	97%	94%
Benchmark B (incremental)	mean time	91	208	215	338	394
	incomplete	0	-	24	23	2
	faster than	-	97%	97%	98%	99%
	timeouts	2	0	0	8	23

them on a large set of benchmarks. We have generated two types of benchmarks from the games used for the Reactive Synthesis Competition (SYNTCOMP) [21]. Benchmark A was generated by converting parity games into Street games using standard methods, and as each Streett pair can be represented by a $\{0,1,2\}$-priority parity game, we represented the complete Streett objective as a conjunction of multiple $\{0,1,2\}$-priority parity objectives, resulting in a generalized parity game. Benchmark B was generated by adding randomly[5] generated parity objectives to given parity games. We considered 200 examples in Benchmark A and more than 1400 examples in Benchmark B.

We summarize the complete set of results of the experiments in[6] Table 1 and Fig. 1. We performed two kinds of experiments. First, we solved every generalized parity game in Benchmark A and B in *one shot* using the different methods. The results are shown in Table 1(top) and Fig. 1(left). Although the average time taken by PeSTel is higher than GenZiel and one partial solver, it is fastest in more than 90% of the games in both benchmarks. Thus, it shows that PeSTel is as efficient as the other methods in most cases. Moreover, for every

[5] The random generator takes three parameters: game graph "G", number of objectives "k", and maximum priority "m"; and then it generates "k" random parity objectives with maximum priority "m" as follows: 50% of the vertices in "G" are selected randomly, and those vertices are assigned priorities ranging from 0 to "m" (including 0 and m) such that 1/m-th (of those 50%) vertices are assigned priority 0 and 1/m-th are assigned priority 1 and so on. The rest 50% are assigned random priorities ranging from 0 to "m". Hence, for every priority, there are at least 1/(2m)-th vertices (i.e., 1/m-th of 50% vertices) with that priority.

[6] See the full version of this paper [5, Appendix C] for a version of Fig. 1 including all solvers considered in Table 1.

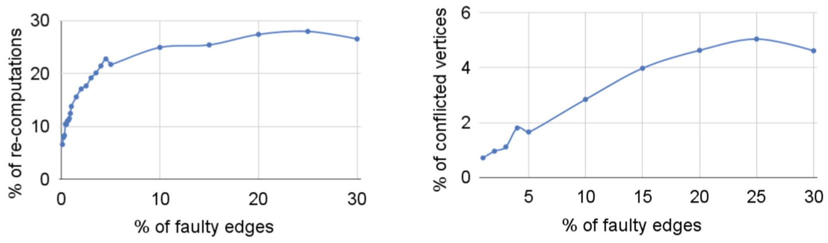


Fig. 4. Experimental results for parity games with faulty edges. Left: percentage of instances with conflicts given a certain percentage of faulty edges. Right: average percentage of vertices that created conflicts given a certain percentage of faulty edges.

game in both benchmarks, PeSTel succeeded to compute the complete winning region, whereas the partial solvers failed to do so in some cases[7]. We note that the instances which are hard for PeSTel are those where the winning region becomes empty, which is quickly detected by GenZiel but only seen by PeSTel after most objectives are (separately) considered.

Second, we solved the examples in Benchmark B by adding the objectives *one-by-one*, i.e., we solved the game with one objective, then we added one more objective and solved it again, and so on. The results are shown in Table 1(bottom) and Fig. 1(right). As PeSTel can use the pre-computed strategy templates if we add a new objective to a game, it outperforms all the other solvers significantly as they need to re-solve the game from scratch every time.

Fault-Tolerant Control. As discussed in Sect. 5.2, strategy templates can be used to implement a fault tolerant time-dependent strategy, if the set of faulty edges F does not cause conflicts with the strategy template. We have used PeSTel on over 200 examples of parity games from SYNTCOMP [21] to evaluate the relevance of such conflicts in practice. For this, we randomly selected different percentages of edges to be faulty and checked for conflicts with the given template. The results are summarized in Fig. 4. The left plot shows the number of instances for which a conflict occurs if a certain percentage of randomly selected edges is faulty. We see that the majority of the instances never faces a conflict even when 30% of the edges are faulty. Looking more closely into the instances with conflicts, Fig. 4(right) shows the average number of conflicting vertices in these benchmarks. Here we see that conflicts occur very locally at a very small number of vertices. Our strategy templates allow for a linear-time algorithm to localize them, allowing to mitigate them in practice by additional hardware.

Remark 1. We remark again that our results are directly applicable to CPS with continuous dynamics via the paradigm of abstraction-based control design (ABCD). In particular, standard abstraction tools such as SCOTS [35],

[7] Additionally, we outperform all algorithms on the benchmarks considered by Bruyère et al. [12]. We have however chosen to not include them in our analysis as many of their generalized parity games have only one objective and are therefore trivial.

ARCS [13], MASCOT [20], P-FACES [20], or ROCS [27] automatically compute a game graph from the (stochastic) continuous dynamics that can directly be used as an input to PeSTel. The winning strategy computed by PeSTel can further be refined into a correct-by-construction continuous feedback controller for the original dynamical system using standard methods from ABCD. We leave these tool integrations to future work.

References

1. Maoz, S., Ringert, J.O.: Synthesizing a lego forklift controller in GR(1): a case study. In: Cerný, P., Kuncak, V., Madhusudan, P. (eds.) Proceedings Fourth Workshop on Synthesis, SYNT 2015, San Francisco, CA, USA, 18th July 2015. EPTCS, vol. 202, pp. 58–72 (2015). https://doi.org/10.4204/EPTCS.202.5
2. Alur, R.: Principles of Cyber-Physical Systems. MIT Press (2015)
3. Alur, R., Moarref, S., Topcu, U.: Counter-strategy guided refinement of GR(1) temporal logic specifications. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, 20–23 October 2013, pp. 26–33. IEEE (2013). https://ieeexplore.ieee.org/document/6679387/
4. Anand, A., Mallik, K., Nayak, S.P., Schmuck, A.K.: Computing adequately permissive assumptions for synthesis. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2023. Lecture Notes in Computer Science, vol. 13994, pp. 211–228. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30820-8_15
5. Anand, A., Nayak, S.P., Schmuck, A.K.: Synthesizing permissive winning strategy templates for parity games (2023)
6. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
7. Belta, C., Yordanov, B., Aydin Gol, E.: Formal Methods for Discrete-Time Dynamical Systems. SSDC, vol. 89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-50763-7
8. Bernet, J., Janin, D., Walukiewicz, I.: Permissive strategies: from parity games to safety games. RAIRO Theor. Inform. Appl. **36**(3), 261–275 (2002). https://doi.org/10.1051/ita:2002013
9. Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M.: Diagnosis and Fault-Tolerant Control. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-540-35653-0
10. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**, 911–938 (2012). https://doi.org/10.1016/j.jcss.2011.08.007
11. Bouyer, P., Markey, N., Olschewski, J., Ummels, M.: Measuring permissiveness in parity games: mean-payoff parity games revisited. In: Bultan, T., Hsiung, P. (eds.) Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis, ATVA 2011, Taipei, Taiwan, 11–14 October 2011. LNCS, vol. 6996, pp. 135–149. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_11
12. Bruyère, V., Pérez, G.A., Raskin, J., Tamines, C.: Partial solvers for generalized parity games. In: Filiot, E., Jungers, R.M., Potapov, I. (eds.) Proceedings of the 13th International Conference on Reachability Problems, RP 2019, Brussels, Belgium, 11–13 September 2019. LNCS, vol. 11674, pp. 63–78. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-30806-3_6

13. Bulancea, O.L., Nilsson, P., Ozay, N.: Nonuniform abstractions, refinement and controller synthesis with novel BDD encodings. IFAC-PapersOnLine **51**(16), 19–24 (2018)
14. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 3rd edn. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-72274-6
15. Chatterjee, K., Henzinger, M.: Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. J. ACM **61**(3) (2014). https://doi.org/10.1145/2597631
16. Chatterjee, K., Henzinger, T.A., Piterman, N.: Generalized parity games. In: Seidl, H. (ed.) Proceedings of the 10th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007. LNCS, Braga, Portugal, 24 March–1 April 2007, vol. 4423, pp. 153–167. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71389-0_12
17. Chatterjee, K., Henzinger, T.A., Piterman, N.: Algorithms for büchi games. CoRR abs/0805.2620 (2008). https://doi.org/10.48550/ARXIV.0805.2620
18. Ehlers, R., Lafortune, S., Tripakis, S., Vardi, M.Y.: Supervisory control and reactive synthesis: a comparative introduction. Discrete Event Dyn. Syst. **27**(2), 209–260 (2016). https://doi.org/10.1007/s10626-015-0223-0
19. Fritz, R., Zhang, P.: Overview of fault-tolerant control methods for discrete event systems. IFAC-PapersOnLine **51**(24), 88–95 (2018). https://doi.org/10.1016/j.ifacol.2018.09.533
20. Hsu, K., Majumdar, R., Mallik, K., Schmuck, A.K.: Multi-layered abstraction-based controller synthesis for continuous-time systems. In: HSCC 2018, pp. 120–129. ACM (2018)
21. Jacobs, S., et al.: The reactive synthesis competition (SYNTCOMP): 2018–2021. CoRR abs/2206.00251 (2022). https://doi.org/10.48550/arXiv.2206.00251
22. Khaled, M., Zamani, M.: pFaces: an acceleration ecosystem for symbolic control. In: HSCC 2019, pp. 252–257. ACM (2019)
23. Klein, J., Baier, C., Klüppelholz, S.: Compositional construction of most general controllers. Acta Informatica **52**(4–5), 443–482 (2015). https://doi.org/10.1007/s00236-015-0239-9
24. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Where's Waldo? Sensor-based temporal logic motion planning. In: 2007 IEEE International Conference on Robotics and Automation, ICRA 2007, 10–14 April 2007, Roma, Italy, pp. 3116–3121. IEEE (2007). https://doi.org/10.1109/ROBOT.2007.363946
25. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE Trans. Robot. **25**(6), 1370–1381 (2009). https://doi.org/10.1109/TRO.2009.2030225
26. Lesi, V., Jakovljevic, Z., Pajic, M.: Towards plug-n-play numerical control for reconfigurable manufacturing systems. In: 21st IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016, Berlin, Germany, 6–9 September 2016, pp. 1–8. IEEE (2016). https://doi.org/10.1109/ETFA.2016.7733524
27. Li, Y., Liu, J.: ROCS: a robustly complete control synthesis tool for nonlinear dynamical systems. In: HSCC 2018, pp. 130–135. ACM (2018)
28. Majumdar, R., Schmuck, A.: Supervisory controller synthesis for nonterminating processes is an obliging game. IEEE Trans. Autom. Control **68**(1), 385–392 (2023). https://doi.org/10.1109/TAC.2022.3143108

29. Meira-Góes, R., Kang, E., Lafortune, S., Tripakis, S.: On synthesizing tolerable and permissive controllers for labeled transition systems. IFAC-PapersOnLine **55**(28), 158–164 (2022). https://doi.org/10.1016/j.ifacol.2022.10.338
30. Moor, T.: A discussion of fault-tolerant supervisory control in terms of formal languages. Annu. Rev. Control. **41**, 159–169 (2016). https://doi.org/10.1016/j.arcontrol.2016.04.001
31. Neider, D., Rabinovich, R., Zimmermann, M.: Down the Borel hierarchy: solving muller games via safety games. Theor. Comput. Sci. **560**, 219–234 (2014). https://doi.org/10.1016/j.tcs.2014.01.017
32. Neider, D., Weinert, A., Zimmermann, M.: Synthesizing optimally resilient controllers. Acta Informatica **57**(1–2), 195–221 (2020). https://doi.org/10.1007/s00236-019-00345-7
33. Nilsson, P., et al.: Correct-by-construction adaptive cruise control: two approaches. IEEE Trans. Control Syst. Technol. **24**(4), 1294–1307 (2016). https://doi.org/10.1109/TCST.2015.2501351
34. Reijnen, F.F.H., Leliveld, E., van de Mortel-Fronczak, J.M., van Dinther, J., Rooda, J.E., Fokkink, W.J.: Synthesized fault-tolerant supervisory controllers, with an application to a rotating bridge. Comput. Ind. **130**, 103473 (2021). https://doi.org/10.1016/j.compind.2021.103473
35. Rungger, M., Zamani, M.: SCOTS: a tool for the synthesis of symbolic controllers. In: HSCC, pp. 99–104. ACM (2016)
36. Scher, G., Kress-Gazit, H.: Warehouse automation in a day: from model to implementation with provable guarantees. In: 16th IEEE International Conference on Automation Science and Engineering, CASE 2020, Hong Kong, 20–21 August 2020, pp. 280–287. IEEE (2020). https://doi.org/10.1109/CASE48305.2020.9217012
37. Schmuck, A.-K., Moor, T., Majumdar, R.: On the relation between reactive synthesis and supervisory control of non-terminating processes. Discrete Event Dyn. Syst. **30**(1), 81–124 (2019). https://doi.org/10.1007/s10626-019-00299-5
38. Svorenová, M., Kretínský, J., Chmelik, M., Chatterjee, K., Cerná, I., Belta, C.: Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games, pp. 259–268 (2015). https://doi.org/10.1145/2728606.2728608
39. Tabuada, P.: Verification and Control of Hybrid Systems - A Symbolic Approach. Springer, New York (2009). https://doi.org/10.1007/978-1-4419-0224-5
40. Wong, K.W., Ehlers, R., Kress-Gazit, H.: Resilient, provably-correct, and high-level robot behaviors. IEEE Trans. Robot. **34**(4), 936–952 (2018). https://doi.org/10.1109/TRO.2018.2830353
41. Wongpiromsarn, T., Topcu, U., Murray, R.M.: Receding horizon control for temporal logic specifications. In: Johansson, K.H., Yi, W. (eds.) Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, 12–15 April 2010, pp. 101–110. ACM (2010). https://doi.org/10.1145/1755952.1755968
42. Wonham, W.M., Cai, K., et al.: Supervisory control of discrete-event systems (2019)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.