Strategy Templates
– Robust Certified Interfaces for Interacting Systems

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

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

Abstract. This invited paper unifies our recent work on the automated synthesis of strategy templates as robust certified interfaces for interacting autonomous systems. Strategy templates are similar to classical strategies typically computed by reactive software synthesis but contain a huge set of relevant strategies in a succinct and simple data structure. This *permissiveness* allows efficient adaptations to new specifications as well as robustness to unexpected actuation failures during runtime. In addition to these favorable engineering properties of strategy templates for cyber-physical systems (CPS), their permissiveness, efficient computability, and adaptability enables their use for the automated synthesis of *certified interfaces* among different interacting CPS. This paper discusses the foundations and applications of this novel *control-inspired distributed reactive synthesis framework* for different cooperation and partial-observation settings common for interacting CPS.

Keywords: Rational Synthesis · Cooperative Synthesis · Strategy Templates · Negotiation · Graph Games · Robustness

1 Introduction

The holy grail of autonomy and artificial intelligence are technological systems that deploy a *reliable seamless and strategic interaction* with each other, which requires the design of *robust certified interfaces* for their interaction. In recent years, this problem has been addressed by a combination of *formal methods* (rooted in theoretical computer science) and *control theory* (rooted in engineering) [5,11–13,15], due to two main observations: first, and more obviously, the trustworthiness of software is the main object of study within the formal methods community, naturally leading to an intersection of concerns for embedded software within cyber-physical systems (CPS). The second, usually non-obvious reason, is that strategic decisions in higher control layers of autonomous CPS are largely event-based, naturally leading to discrete, transition-based models,

Authors are ordered randomly, denoted by (,) and the publicly verifiable record of the randomization is available at www.aeaweb.org/journals/policies/random-author-order/search?. The authors are supported by the DFG projects SCHM 3541/1-1 and 389792660 TRR 248-CPEC.

S. Akshay et al. (Eds.): ATVA 2024, LNCS 15054, pp. 22–41, 2025.
https://doi.org/10.1007/978-3-031-78709-6_2

such as automata, formal languages or logic. In both cases, so called *reactive* (software) systems are particularly related to feedback control. Here, the computations of a software component are influenced by inputs from the outside which might, in turn, be effected by their outputs, which forms a *logical feedback loop* between the (software) system and its (software) environment.

This connection between reactive software and feedback control is well known and existing work bridges both fields by either (i) enhancing classical feedback control with formal methods, or (ii) enhancing reactive synthesis with physical dynamics.However, methods that efficiently *integrate both* to address the challenges of *seamless strategic feedback* among autonomous CPS are still in their infancy. This invited paper outlines how our recent research addresses this gap by *rethinking distributed reactive synthesis* from a control perspective for *robust certified interface design* in higher layers of the CPS control software stack.

Distributed Reactive Synthesis is concerned with the automated computation of implementations of interacting reactive software components, given specifications for each component in linear temporal logic (LTL). While this classical synthesis problem is known to be undecidable, we consider a common variant where the strategic interaction capabilities of the components are known and modelled as a P-player game graph G. However, in contrast to most previous work, we assume that players only know their own objective Φ_p along with G and compute their own strategy π_p independently under this knowledge. In addition, during runtime, each player only observes the current moves of other players, but does not know their strategy, and hence, cannot use this knowledge to condition its strategy on. Given such an instance, this paper presents our novel techniques to *synthesize robust certified interfaces* for fully cooperating [3] and rationally interacting [10] players, which allows them to adapt their local strategies efficiently both during (offline) synthesis and during (online) deployment.

The main ingredient of our framework is the realization of these interfaces by *strategy templates*. We first introduced strategy templates as *novel robust and adaptable logical controllers* for single systems [2]. Strategy templates are similar to classical strategies typically computed by reactive synthesis but contain a huge set of relevant strategies in a succinct and simple data structure. This *permissiveness* allows efficient adaptations to new specifications as well as robustness to unexpected actuation failures during runtime. In addition, strategy template synthesis only requires mild modifications of the classical reactive synthesis algorithms and therefore enjoys identical computational complexities.

In addition to the favorable engineering properties of strategy templates for CPS control, their permissiveness, efficient computability, and adaptability enables their use for the synthesis of *certified interfaces* among different interacting CPS. Here the fundamental idea is to iteratively *co-synthesize* templates both as *assumptions* and as *guarantees* by individual players which are iteratively refined into certified interfaces. While assume-guarantee reasoning has proven useful in verification, where all component implementations are known, assume-guarantee *synthesis* usually encounters a chicken-and-egg problem – without

component implementations, no contracts can be obtained and without contracts, no respecting implementations can be synthesized. Our work fills this gap by locally computing *permissive templates* which retain as many feasible ways of cooperation as possible to allow for a distributed implementation to be eventually discovered by their iterative and distributed refinement. In particular, our frameworks does not only address safety – which naturally allows for maximal permissiveness – but to also enable *certified interactive progress*, which is known to be very challenging already for verification.

In Summary, our *iterative synthesis frameworks* are

- *expressive* – we consider parity specifications to model system objectives;
- *privacy concerned* – specifications and strategy choices of systems are not directly shared or handled centrally;
- *computationally tractable* – the efficient computability and adaptability properties of strategy templates allow for reasonable synthesis times; and
- *ensured to terminate* – due to careful under-approximations of templates which prevent game graph modifications in between iterations.

Further, the resulting *strategy templates* form a robust certified interface for interacting CPS due to their

- *full decentralization of strategy choices* – during runtime, the correct interaction of components is achieved by any locally chosen strategy, as long as each one satisfies the (previously iteratively synthesized) local template;
- *their robustness* – the full decentralization of strategy choices retains the local robustness and adaptability of strategy templates during deployment.

In addition, we have recently shown the applicability of strategy templates beyond distributed reactive synthesis. In particular, we have

- introduced a *hierarchical control framework* which combines low-level (fast) physical feedback control with high-level (slow) logical feedback control in a novel seamless manner [9] powered by the use of strategy templates as *vertical interfaces* between both control layers, and
- used strategy templates to build a *progress guided refinement framework* for the synthesis of embedded software via *infinite-state reactive program games* [14], which can efficiently handle synthesis problem which are non-tractable by any existing partial solvers for this (undecidable) synthesis problem.

Outline. This invited paper is based on our recent work [1–3,10], to which we refer for an in-depth discussion of related work, the proposed synthesis algorithms, their correctness proofs and their experimental evaluations along with many illustrative examples. In contrast, this invited paper illustrates the underlying similarities and differences of these algorithms. After recalling preliminaries in Sect. 2 and the general notion of strategy templates in Sect. 3, the discussion of the iterative synthesis frameworks in Sect. 4 provides a novel joint view on the negotiation frameworks for cooperative [3] and rational [10] players. This allows

us to extend our cooperative framework from 2 to P players, which gives additional insights about the permissiveness and termination nature of our synthesis frameworks, not contained in our previous work. In addition, Sect. 5 provides a new, unpublished algorithm for strategy template based control *under partial observation*, typically present in autonomous CPS applications.

2 Preliminaries

This section introduces notation and preliminaries used throughout the paper.

Notation. We use $\mathbb{N}$ to denote the set of natural numbers including zero. Given two natural numbers $a, b \in \mathbb{N}$ with $a \leq b$, we use $[a; b]$ to denote the set $\{n \in \mathbb{N} \mid a \leq n \leq b\}$. The notations Σ^* and Σ^ω, respectively, denote the set of finite and infinite words over the finite alphabet Σ. Given two words $u \in \Sigma^*$ and $v \in \Sigma^* \cup \Sigma^\omega$, the concatenation of u and v is written as the word uv.

2.1 Infinite Games over Finite Graphs

We use games over finite graphs as an abstract model for the interaction dynamics within in a distributed CPS. This section introduces notation and preliminaries on this modeling formalism.

Game Graphs. A *P-player (turn-based) game graph* is a tuple $G = (V, E, v_0)$ where (V, E, v_0) is a finite, directed graph with *vertices* V and *edges* E, and $v_0 \in V$ is an initial vertex. For such a game graph, let $\mathtt{P} = [1; P]$ be the set of players such that $V = \bigcup_{p \in \mathtt{P}} V_i$ is partitioned into vertices of P players in $\mathtt{P}$. We write E_p, $p \in \mathtt{P}$, to denote the edges from Player p's vertices, i.e., $E_p = E \cap (V_p \times V)$. Further, we write $V_{\neg p}$ and $E_{\neg p}$ to denote the set $\bigcup_{q \neq p} V_p$ and $\bigcup_{q \neq p} E_q$, respectively. A *play* from a vertex u_0 is a finite or infinite sequence of vertices $\rho = u_0 u_1 \ldots$ with $(u_j, u_{j+1}) \in E$ for all $j \geq 0$.

Winning Conditions. Given a game graph G, a *winning condition* (or *objective*) is a set of plays specified using a formula Φ in *linear temporal logic* (LTL) over the vertex set V, i.e., LTL formulas whose atomic propositions are sets of vertices from V. We write $\mathcal{L}(\Phi) \subseteq V^\omega$ to denote the set of plays that satisfy Φ. The standard definitions of ω-regular languages and LTL are omitted for brevity and can be found in standard textbooks [4]. In particular, we consider *parity objectives* given by the LTL formula

$$Parity(\mathbb{C}) := \bigwedge_{i \in_{\text{odd}} [0;d]} \left(\square\lozenge C^i \implies \bigvee_{j \in_{\text{even}} [i+1;d]} \square\lozenge C^j \right), \tag{1}$$

with *color set* $C^j = \{v : \mathbb{C}(v) = j\}$ for $0 \leq j \leq d$ of vertices for some *coloring function* $\mathbb{C} : V \to [0; d]$ that assigns each vertex a *color* (a.k.a. priority). Hence, $\mathcal{L}(Parity(\mathbb{C}))$ contains all plays ρ for which the highest color appearing infinitely

often is even. We note that every game with an arbitrary ω-regular winning condition can be reduced to a parity game (possibly with a larger vertex set) [4].

Games. A *P-player (single-objective) game* is a tuple $\mathcal{G} = (G, \Phi)$, where G is a game graph, and Φ is the *winning condition* over G. A *P-player multi-objective game* is a pair $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ where G is a P-player game graph and each Φ_p is an *objective* for Player p over G. We call a $\mathcal{G}$ a parity game if all involved winning conditions are parity objectives.

Strategies. A strategy of Player p, $p \in \mathtt{P}$, is a function $\pi_p \colon V^* V_p \to V$ such that for every $\rho v \in V^* V_p$, it holds that $(v, \pi_p(\rho v)) \in E$. A strategy profile for a set of players $\mathtt{P}' \subseteq \mathtt{P}$ is a tuple $(\pi_p)_{p \in \mathtt{P}'}$ of strategies, one for each player in $\mathtt{P}'$. To simplify notation, we write $\mathtt{P}_{\neg p}$ and $\pi_{\neg p}$ to denote the set $\mathtt{P} \setminus \{p\}$ and their strategy profile $(\pi_q)_{q \in \mathtt{P} \setminus \{p\}}$, respectively. Given a strategy profile $(\pi_p)_{p \in \mathtt{P}'}$, we say that a play $\rho = u_0 u_1 \ldots$ is a $(\pi_p)_{p \in \mathtt{P}'}$*-play* if for every $p \in \mathtt{P}'$ and for all $\ell \geq 1$, it holds that $u_{\ell - 1} \in V_p$ implies $u_\ell = \pi_p(u_0 \ldots u_{\ell - 1})$.

Winning. Given a game graph G, a play ρ in G is *winning* for objective Φ if it satisfies Φ, i.e., $\rho \in \mathcal{L}(\Phi)$. A strategy profile $(\pi_p)_{p \in \mathtt{P}'}$ for $\mathtt{P}' \subseteq \mathtt{P}$ is winning for objective Φ from a vertex v, denoted by $(\pi_p)_{p \in \mathtt{P}'} \vDash_v \Phi$, if every $(\pi_p)_{p \in \mathtt{P}'}$-play from v satisfies Φ. We write $(\pi_p)_{p \in \mathtt{P}'} \vDash \Phi$ if v is the initial vertex. We collect all vertices from which there exists a strategy profile for players in $\mathtt{P}'$ that satisfies Φ in the winning region[1] $\langle\!\langle \mathtt{P}' \rangle\!\rangle \Phi$.

We note two special cases. First, if $\mathtt{P}' = \{p\}$, $\pi_p \vDash_v \Phi$ requires that Player p wins from v against all strategies of players in $\mathtt{P}_{\neg p}$. If $|\mathtt{P}| = 2$, this corresponds to the classical notion of *adversarial winning* in zero-sum games. Second, given a multi-objective game $(G, (\Phi_p)_{p \in \mathtt{P}})$ and the full set of players $\mathtt{P}' = \mathtt{P}$, $(\pi_p)_{p \in \mathtt{P}} \vDash \bigwedge_{p \in \mathtt{P}} \Phi_p$ denotes a *cooperatively winning* strategy profile.

Rational Winning. To formalize a third type of winning that sits between adversarial and cooperative winning, we use the concept of *winning secure equilibria (WSE)* [7] to introduce *rationality* among players. A WSE is a Nash equilibrium where every player's main goal is to satisfy their own objective Φ_p, and, as a secondary goal, falsify the objectives of the other players. Given a P-player multi-objective game $(G, (\Phi_p)_{p \in \mathtt{P}})$, a *WSE* is a strategy profile $(\pi_p)_{p \in \mathtt{P}'}$ for $\mathtt{P}' \subseteq \mathtt{P}$ such that (i)$(\pi_p)_{p \in \mathtt{P}'} \vDash \bigwedge_{p \in \mathtt{P}'} \Phi_p$; and (ii)for every strategy π'_p of Player $p \in \mathtt{P}'$ with $\mathtt{P}'' = \mathtt{P}' \setminus \{p\}$, it holds that $(\pi'_p, (\pi_q)_{q \in \mathtt{P}''}) \nvDash \bigwedge_{q \in \mathtt{P}''} \Phi_q \Rightarrow (\pi'_p, (\pi_q)_{q \in \mathtt{P}''}) \nvDash \Phi_p$. Intuitively, item (i) ensures that the strategy profile satisfies all player's objective, whereas item (ii) ensures that no player can improve, i.e., falsify another player's objective without falsifying their own objective, by deviating from the prescribed strategy.

We call a strategy profile $(\pi_p)_{p \in \mathtt{P}}$ *rationally winning* if it is a WSE over $(G, (\Phi_p)_{p \in \mathtt{P}})$ for the full set of players $\mathtt{P}$, i.e. $\mathtt{P}' = \mathtt{P}$.

[1] Slightly abusing notation, we write $\langle\!\langle p \rangle\!\rangle \Phi$ for singleton sets of players $\mathtt{P}' = \{p\}$.

2.2 Permissive Templates

In principle, a template is simply an LTL formula Λ over a game graph G. We will, however, restrict attention to four distinct types of such formulas, and interpret them as a succinct way to represent a set of strategies for each player, in particular all strategies that *follow* Λ. Formally, a Player p strategy π_p *follows* Λ if every π_p-play belongs to $\mathcal{L}(\Lambda)$, i.e., $\pi_p \vDash_v \Lambda$ for all $v \in V$. The exposition in this section follows the presentation in [1] where more illustrative examples and intuitive explanations can be found.

Safety Templates. Given a set $S \subseteq E$ of *unsafe edges*, the safety template is defined as $\Lambda_{\text{UNSAFE}}(S) := \square \wedge_{e \in S} \neg e$, where an edge $e = (u, v)$ is equivalent to the LTL formula $u \wedge \bigcirc v$. A safety template requires that an edge to S should never be taken.

Live-Group Templates. 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 $H_\ell = \{H_i\}_{i \geq 0}$ we define a live-group template as $\Lambda_{\text{LIVE}}(H_\ell) := \bigwedge_{i \geq 0} \square\lozenge src(H_i) \Rightarrow \square\lozenge H_i$. A live-group template requires 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.

Conditional Live-Group Templates. A *conditional live-group* over G is a pair (R, H_ℓ), where $R \subseteq V$ and H_ℓ is a set of live groups. Given a set of conditional live groups $\mathcal{H}$ we define a *conditional live-group template* as $\Lambda_{\text{COND}}(\mathcal{H}) := \bigwedge_{(R,H_\ell) \in \mathcal{H}} (\square\lozenge R \Rightarrow \Lambda_{\text{LIVE}}(H_\ell))$. A conditional live-group template requires that for every pair (R, H_ℓ), if some vertex from the set R is visited infinitely often, then a following strategy must follow the live-group template $\Lambda_{\text{LIVE}}(H_\ell)$. We write $(\cdot, H_\ell)$ if there exists $R \subseteq V$, such that (R, H_ℓ) is a conditional live group.

Co-liveness Templates. Given a set of *co-live* edges D, a co-live template is defined as $\Lambda_{\text{COLIVE}}(D) := \bigwedge_{e \in D} \lozenge\square\neg e$. A co-liveness template requires that edges in D are only taken finitely often.

Composed Templates. A template $\Lambda := \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D) \wedge \Lambda_{\text{COND}}(\mathcal{H})$ will be associated with the tuple $(S, D, \mathcal{H})$, denoted by $\Lambda \lhd (S, D, \mathcal{H})$, in the rest of the paper. Similarly, $\Lambda \lhd (S, D, H_\ell)$ denotes the template $\Lambda := \Lambda_{\text{UNSAFE}}(S) \wedge \Lambda_{\text{COLIVE}}(D) \wedge \Lambda_{\text{LIVE}}(H_\ell)$. We further note that the conjunction of two templates $\Lambda \lhd (S, D, \mathcal{H})$ and $\Lambda' \lhd (S', D', \mathcal{H}')$ is equivalent to the template $(\Lambda \wedge \Lambda') \lhd (S \cup S', D \cup D', \mathcal{H} \cup \mathcal{H}')$ by the definition of conjunction of LTL formulas.

Player Specific Templates. It will be customary to separate templates to restrict single players. Hence, a template $\Lambda \lhd (S, D, \mathcal{H})$ over G is called a *p-template* for some $p \in \mathtt{P}$ if all edges used in the template are Player p's edges, i.e., $S \cup D \cup \overline{H} \subseteq E_p$, where $\overline{H} := \bigcup\{H \in H_\ell \mid (\cdot, H_\ell) \in \mathcal{H}\}$. Furthermore, given any template $\Lambda \lhd (S, D, H_\ell)$, one can extract a p-template $\Lambda|_{E_p}$ by restricting the edges used to E_p, i.e., $\Lambda|_{E_p} \lhd (S \cap E_p, D \cap E_p, H_\ell|_{E_p})$ with $H_\ell|_{E_p} = \{H \cap E_p \mid H \in H_\ell\}$. Similarly, for $\Lambda \lhd (S, D, \mathcal{H})$, we define $\Lambda|_{E_p} \lhd (S \cap E_p, D \cap E_p, \mathcal{H}|_{E_p})$ with $\mathcal{H}|_{E_p} = \big\{(R, H_\ell|_{E_p}) \mid (R, H_\ell) \in \mathcal{H}\big\}$.

3 Strategy Templates for Adversarial Winning

This section introduces the concept of strategy templates as a generalization of strategies in zero-sum games over finite graphs, i.e., in games where it is only one player's goal to satisfy the given objective Φ, while all other players try to achieve its negation (see Fig. 1 (top, right) for an illustration of this scenario). This setting reduces to a zero-sum *two-player game* between the protagonist, who must satisfy Φ and the antagonist (i.e., the collection of all other players) who try to satisfy $\neg\Phi$. The content in this section is taken from [2] and included in this paper for completeness.

*Example 1 (**Illustrative Example**).* To appreciate the simplicity and easy adaptability of our strategy templates, consider the game graph in Fig. 1 (left). The first winning condition $\Phi^0 = \Box\neg e$ requires vertex e to never be seen along a play. This can be enforced by Player 0 from vertices $\mathcal{W}_0 = \{a, b, c, d, f\}$ called the *winning region*. The safety template $\Pi_0^0 = \Lambda_{\text{UNSAFE}}(e_{ce})$ ensures that the game always stays in $\mathcal{W}_0$ by forcing the edge e_{ce} 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 $\Phi^1 = \Box\Diamond\{c, f, g\}$ which requires vertex c, f, or g to be seen infinitely often. This induces the template $\Pi_0^1 = \Lambda_{\text{LIVE}}(\{e_{bc}\}) \wedge \Lambda_{\text{LIVE}}(\{e_{ab}\})$ with two live groups which requires that whenever vertex b (resp. vertex a) is seen infinitely often, edge e_{bc}

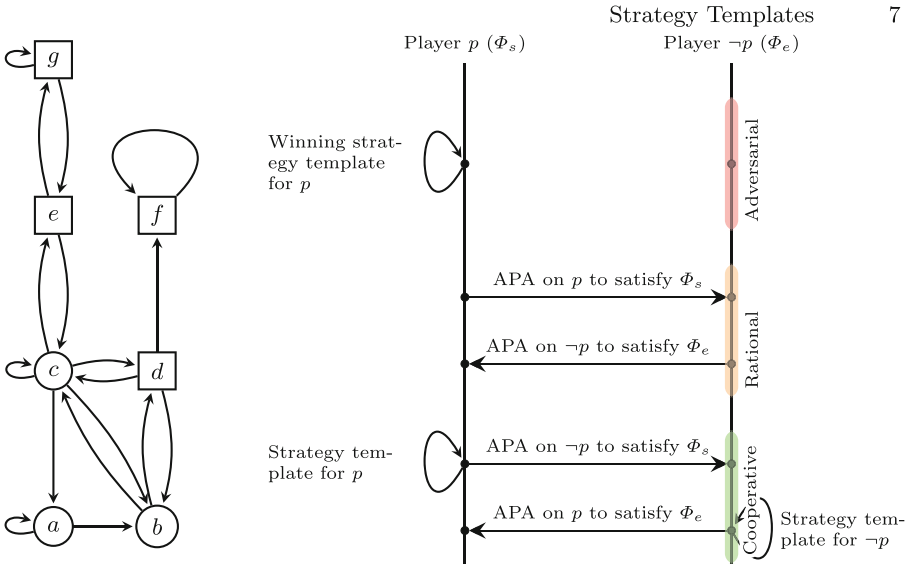

Fig. 1. (Left) A 2-player game graph with Player 0 (circle) and Player 1 (square) vertices. (Right) A graph showing the communication between Player p and Player $\neg p$ for different behaviors of Player $\neg p$. Self loop denotes the information that is computed for self and is not shared with others.

(resp. edge e_{ab}) 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 any vertex and there are infinitely many such compliant winning strategies. Finally, we consider condition $\Phi^2 = \Diamond\Box\neg b$ requiring vertex b to be seen only finitely often. This induces the co-liveness template $\Pi_0^2 = \Lambda_{\text{COLIVE}}(e_{cb}, e_{cd}, e_{ab})$ requiring that all edges from Player 0 vertices which unavoidably might lead to vertex b, i.e., e_{cb}, e_{cd}, e_{ab}, are taken only finitely often.

Winning Strategy Templates. Intuitively, a strategy template is a p-template Π_p which is used by Player p to extract a strategy for themself. We say a strategy template Π_p is *winning from a vertex* v for a game (G, Φ) if every Player p strategy following the template Π_p is winning from v. Moreover, we say a strategy template Π_p is *winning* if it is winning from every vertex in $\langle\langle p\rangle\rangle\Phi$.

The algorithm to compute a winning strategy template in a parity game lies in same time complexity class as the standard algorithm, i.e., Zielonka's algorithm [16], for solving parity games. This leads to the following result:

Proposition 1 ([2], Thm. 4). *Given a parity game with game graph G and coloring function $\mathbb{C}\colon V \to [0, d]$, a winning strategy template can be computed in $\mathcal{O}\left(|V|^{d+\mathcal{O}(1)}\right)$ time.*

Remark 1. While objectives like safety ($\Box\neg V'$), Büchi ($\Box\Diamond V'$), and co-Büchi ($\Diamond\Box V'$) are weaker than parity objectives and are subsumed by the latter, the winning strategy templates for these objectives can be computed in the matching time of their respective standard algorithms, i.e., $\mathcal{O}(m), \mathcal{O}(nm)$, and $\mathcal{O}(nm)$, respectively [2, Theorem 1–3], where $n = |V|, m = |E|$. Moreover, for safety objectives we can compute maximally permissive strategy templates (see below).

Permissiveness of Winning Strategy Templates. We call Π_p *maximally permissive* for $\mathcal{G}$, if every Player p strategy π which is winning in $\mathcal{G}$ also follows Π_p. It is easy to see that safety templates are naturally maximally permissive. However, while live-group templates capture infinitely many winning strategies in Büchi games, they *may not be* maximally permissive (see Ex. 2). As shown in the example, the strategy templates may fail to capture strategies with memory that can react to environment strategies and put less restriction on the system if the environment unexpectedly starts helping the system.

Example 2. Consider the game as in Ex. 1 with objective $\Phi^1 = \Box\Diamond\{c, f, g\}$. Our algorithm outputs the Player 0 strategy template $\Pi_0^1 = \Lambda_{\text{LIVE}}(\{e_{bc}\}) \wedge \Lambda_{\text{LIVE}}(\{e_{ab}\})$ as in Ex. 1. Now consider the winning strategy π_0 of Player 0 with memory that takes edge e_{ab} from a, takes e_{cb} from c, and takes e_{bc} for play suffix db and e_{bd} for every other play ending in b. This strategy does not follow the template—the play $(cbd)^\omega$ is in $\mathcal{L}(\pi_0)$ but not in $\mathcal{L}(\Pi_0)$.

Adaptability through Strategy Templates. Since the strategy templates provide structured guidelines for the systems to satisfy their tasks, we can now

combine templates for individual objectives to obtain new templates to satisfy dynamic objectives. More formally, for instance, given two objectives Φ and Φ', we can obtain a strategy template by combining her templates Π_0 and Π'_0 for the individual objectives, and one may observe that all strategies compliant with $\Pi_0 \wedge \Pi'_0$ are winning for $\Phi \wedge \Phi'$. This observation allows for more efficient recomputation of strategies when objectives for the system arrives one after another—compute the strategy template for the newest parity objective and conjunct it with the existing strategy template [2, Sec. 5.1, Alg. 4]. Consider the game as in Ex. 1 with objective $\Phi^1 = \Box\Diamond\{c, f, g\}$. The strategy template for Player 0 is Π_0^1 as in Ex. 1. Now, if another objective $\Phi^0 = \Box\neg e$ is added, for which the strategy template is $\Pi_0^0 = \Lambda_{\text{UNSAFE}}(e_{ce})$, then the template $\Pi_0^1 \wedge \Pi_0^0$—making e_{ce} unsafe—is a winning strategy template for $\Phi^1 \wedge \Phi^0$.

In addition to their compositionality, strategy templates also allow for local strategy adaptations in case of edge unavailability faults. Consider again the game in Ex. 1 with objective $\Phi^2 = \Diamond\Box\neg b$, for which the strategy template is $\Pi_0^2 = \Lambda_{\text{COLIVE}}(e_{cb}, e_{cd}, e_{ab})$. Suppose that Player 0 follows the strategy π: $c \mapsto a$, $a \mapsto b$, $b \mapsto c$, which is compliant with Π_0^2. If the edge e_{ca} becomes unavailable, we would need to re-solve the game for the modified game graph $G' = (V, E' = E \setminus \{e_{ca}\})$. However, we see that the strategy π': $c \mapsto c$, $a \mapsto b$ and $b \mapsto c$ is actually compliant with Π_0^2 over G'. This allows us to obtain a new strategy in linear time without re-solving the game if edges become temporary unavailable during deployment.

While these examples demonstrate the potential of templates for strategy adaptation (both during synthesis and during deployment), there exist scenarios where conflicts arise from these adaptations, as illustrated next.

Example 3. Consider the game in Ex. 1 with objectives Φ^1, for which the strategy template is Π_0^1. If Player 0 is now required to satisfy an additional objective $\Phi^2 = \Diamond\Box\neg b$, for which the strategy template is Π_0^2, then the composition of the templates $\Pi_0^1 \wedge \Pi_0^2$ marks the edge e_{ab} as live and co-live simultaneously. This causes a conflict at a—if the play arrives infinitely often at a, there is no edge that can be taken to follow the templates. Hence, to remove the conflicts, we must ensure that a is not visited infinitely often. So, we re-solve the game with an additional objective $\Diamond\Box\neg a$.

Conflict-Free Templates and Strategy Extraction. The previous examples demonstrated that a winning strategy can be efficiently computed from a template, if it is *conflict-free*. Formally, a template $\Lambda \lhd (S, D, \mathcal{H})$ over game graph $G = (V, E)$ is *conflict-free* if (i) every vertex v has an outgoing edge that is neither co-live nor unsafe, i.e., $E \cap (v \times V) \not\subseteq S \cup D$, and (ii) in every live-group $H \in H_\ell$ s.t. $(\cdot, H_\ell) \in \mathcal{H}$, every source vertex v has an outgoing edge in H that is neither co-live nor unsafe, i.e., $H \cap (v \times V) \not\subseteq S \cup D$.

Note that checking (i)-(ii) can be done independently for every vertex, and hence, checking whether a template is conflict-free can be done in linear time. In addition, whenever the existentially quantified edge in (i) and (ii) exists, a strategy that alternates between all these edges follows the given template. This

results in a linear time extraction of a strategy from a conflict-free template as restated below.

Theorem 1 ([3], Prop. 21). *Given a template Λ over a game graph (V, E, v_0), checking whether Λ is conflict-free can be done in $\mathcal{O}(|E|)$ time. Moreover, if Λ is conflict-free, a strategy following Λ can also be extracted in $\mathcal{O}(|E|)$ time.*

We further showed in [2, Sec.4] that strategy templates directly computed for a parity objective are always conflict free. Conflicts only arise from the above-mentioned adaptations due to changes in the specification or the game graph.

Practical Evaluation. Using our prototype tool `PeSTel`[2], we evaluated the synthesis and adaptability of strategy templates on over 200 examples of parity games adapted from the SYNTCOMP benchmark suite [8]. Our analysis shows that the cheap local adaptation of templates usually does not create conflicts, effectively avoiding the computational overhead of re-computation. In addition, our experiments show that the observed non-permissiveness of strategy templates for specifications beyond safety never yielded a reduced winning region compared to its exact computation for all 200 considered benchmark instances (see [2, Sec.6] for more details). This suggests that our technique can synthesize and adapt strategy templates for large problem class efficiently in practice.

4 Contract-Based Distributed Reactive Synthesis

While the previous section has introduced strategy templates for zero-sum games, we now consider the more common instance where every player, i.e., every component, has their own objective, leading us to *multi-objective* P-player games. For this setting, we first consider *rational players*, i.e., players whose goal is to satisfy their own objective, and only if their objective is not at risk, are adversarial to others. Thereby, rational players might still 'accidentally' help others while perusing their own objective (see Fig. 1 (middle, right) for an illustration of this scenario), and hence, do not need to be treated fully adversarially. While rational players allow for a very robust notion of interaction, the assumption of systems always ultimately being adversarial, might not always be realistic. In particular, if interacting CPS are co-synthesized the design of one component could take the needs of other components into account if these needs are known, resulting in a *cooperative* negotiation framework (see Fig. 1 (bottom, right)).

Within this section, we present the outlined synthesis frameworks of fully cooperating [3] and rationally interacting [10] players jointly. In order to do so, we first give a unified view of the underlying synthesis principles in Sect. 4.1 and then elaborate on the different algorithms for their realization in Sect. 4.2 and Sect. 4.3, respectively. In particular, this novel joint presentation extents our work on cooperative strategies from two players (as presented in [3]) to P-players – becoming en-par with our work on rationally interacting players [10]. In addition, the joint presentation of both frameworks reveals their fundamental similarities and differences, which have not been presented in this form before.

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

4.1 Contracted Specifications

Both negotiation frameworks for multi-objective games rely on the iterative computation of a new set of objectives, one for each player, such that each player can independently realize these new objectives (e.g., by a local (zero-sum) strategy template). The main intuition behind the design of this – to be computed – new specification profile, is that it should (i) ensure that any such local realization results in a global strategy profile that is winning, w.r.t. the given cooperation mode (soundness), while (ii) enable each player to realize this specification fully locally (decentralization), and (iii) not loosing any winning strategy profile on the way (maximal). This is formalized via contracted specifications next.

Definition 1 (Contracted Specifications). *Given a game* $(G, \{\Phi_p\}_{p\in\mathtt{P}})$*, a specification profile* $\{\varphi\}_{p\in\mathtt{P}}$ *is said to be a* cooperatively contracted specification CCS *(resp. a* rationally contracted specification RCS*) if it is both*

(i) sound*: any* $(\pi_p)_{p\in\mathtt{P}}$ *with* $\pi_p \vDash \varphi_p$ *is cooperatively (resp. rationally) winning;*
(ii) decentralized*: it holds that* $v_0 \in \langle\!\langle p \rangle\!\rangle \varphi_p$ *for all* $p \in \mathtt{P}$*.*

In addition, CCSs and RCSs are called maximal *if* $\mathcal{L}(\bigwedge_{p\in\mathtt{P}} \varphi_p) = \mathcal{L}(\bigwedge_{p\in\mathtt{P}} \Phi_p)$*.*

With this, the decentralized synthesis problem reduces to the computation of such contracted specifications. In both frameworks, this computation relies on the ability to iteratively refine an over-approximation of possible strategies a component might use in a (globally) winning strategy profile. The main insight that enables this efficiently, is that templates can be used for this purpose. Now, however, not strategy, but *assumption templates* $\Psi_{\mathtt{P}'}$ on players in $\mathtt{P}' \subseteq \mathtt{P}$ are used by the other players to over-approximate strategies of players $\mathtt{P}'$. In order to realize the outlined strategy over-approximation and enable an iterative framework which results in contracted specifications, *adequately permissive assumptions* are needed.

Definition 2 (APA; adapted from [1]). Given a P-player game graph $G = (V, E, v_0)$ and a specification Φ, we say that an assumption template $\Psi_{\mathtt{P}'}$ is an *adequately permissive assumption (APA)* on players $\mathtt{P}' \subseteq \mathtt{P}$ for Φ if it is

(i) *sufficient*: there exists a strategy profile $(\pi_p)_{p\notin\mathtt{P}'}$ such that for every strategy profile $(\pi_p)_{p\in\mathtt{P}'}$ of players in $\mathtt{P}'$ with $(\pi_p)_{p\in\mathtt{P}'} \vDash \Psi_{\mathtt{P}'}$, we have $(\pi_p)_{p\in\mathtt{P}} \vDash \Phi$;
(ii) *implementable*: $\langle\!\langle \mathtt{P}' \rangle\!\rangle \Psi_{\mathtt{P}'} = V$; and
(iii) *permissive*: $\mathcal{L}(\Psi_{\mathtt{P}'}) \supseteq \mathcal{L}(\Phi)$.

Intuitively, sufficiency ensures that Ψ provides a sufficient restriction on player $\mathtt{P}'$ strategies under which the others can ensure Φ. Further, implementability ensures that Ψ can be ensured by the players in $\mathtt{P}'$ without any help from the other players. Finally, permissiveness ensures that the assumption does not prevent any winning strategy profile from being realized.

Note that computing APAs on players $\mathtt{P}'$ can be reduced to computing APAs on one player in a 2-player game by considering the players in $\mathtt{P}'$ as one player and all other players as another player. We can therefore use our algorithm from [1] to compute APAs for two-player parity games in polynomial time.

Algorithm 1. ComputeRCS($\mathcal{G}$); adapted from [10, Alg.1]

Input: A P-player multi-objective parity game $\mathcal{G} = (G = (V, E, v_0), (\Phi_p)_{p \in \mathtt{P}})$.
Output: A rationally contracted specification $(\varphi_p)_{p \in \mathtt{P}}$.

1: $\Psi_p \leftarrow \mathsf{True}\ \forall p \in \mathtt{P}$
2: **return** RecursiveRCS$(\mathcal{G}, (\Psi_p)_{p \in \mathtt{P}})$

3: **procedure** RecursiveRCS$(\mathcal{G}, (\Psi_p)_{p \in \mathtt{P}})$
4: $\quad \Psi_{\neg p} \leftarrow \bigwedge_{q \neq p} \Psi_q\ \forall p \in \mathtt{P}$
5: $\quad \varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)\ \forall p \in \mathtt{P}$
6: $\quad$ **if** $v_0 \in \bigcap_{p \in \mathtt{P}} \langle\!\langle p \rangle\!\rangle \varphi_p$ **then**
7: $\quad\quad$ **return** $(\varphi_p)_{p \in \mathtt{P}}$
8: $\quad \Psi'_p \leftarrow \Psi_p \wedge$ ComputeAPA$(G, \Psi_{\neg p} \wedge \Phi_p, p)\ \forall p \in \mathtt{P}$
9: $\quad$ **return** RecursiveRCS$(\mathcal{G}, (\Psi'_p)_{p \in \mathtt{P}})$

Lemma 1 ([1, Thm. 4]). *Given a P-player game graph $G = (V, E, v_0)$ and a parity specification $\Phi = Parity(\mathbb{C})$, an APA on a set of players $\mathtt{P}' \subseteq \mathtt{P}$ for Φ can be computed, if one exists, in time $\mathcal{O}(|V|^4)$. We write* [3] ComputeAPA$(G, \Phi, \mathtt{P}')$ *to denote the procedure that returns this APA if it exists; and* False*, otherwise.*

It remains to show how contracted specifications are computed via the iterative refinement of APAs for rationally [10] and cooperatively [3] interacting players, which will be presented in Sect. 4.2 and Sect. 4.3, respectively.

4.2 Rationally Contracted Specifications

Intuitively, rationally interacting agents are intrinsically selfish. They are primed on achieving their own objective. However, through this selfishness they *might* – accidentally – help others. Towards computing a contracted specification over rational agents, we first over-approximate the way in which players (accidentally) help each other by letting each Player p compute an APA Ψ_p *on themself* by executing the procedure ComputeAPA(G, Φ_p, p). Intuitively, Ψ_p collects all restrictions on Player p strategies (i.e., moves they *choose themself*) s.t. the resulting play *can* be winning for Φ_p if others cooperate (somehow). It therefore *over-approximates* the set of all Player p strategies which could possibly form a rationally winning strategy profile with the other players. As a consequence, the intersection $\Psi_{\neg p} = \bigwedge_{q \neq p} \Psi_q$ *under-approximates* the way in which other players (accidentally) help Player p.

Computation. The main idea of algorithm 1, called ComputeRCS$(\mathcal{G})$, is therefore to iteratively refine the assumptions $(\Psi_p)_{p \in \mathtt{P}}$ on every player – via RecursiveRCS – until $(\varphi_p)_{p \in \mathtt{P}}$ with $\varphi_p := \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ (computed in line 5) is a RCS. By Def. 1 this requires that Player p is able to choose a Player p

[3] With slight abuse of notation, we also call the algorithm ComputeAPA$(G, \Phi, \mathtt{P}')$ if Φ is an arbitrary ω-regular specification. Then the game is converted into a parity game first (with a possibly exponentially larger state space) by standard methods [4].

strategy π_p s.t. $\pi_p \vDash \varphi_p$ i.e., π_p must (a) satisfy Ψ_p against all strategies of other players (which follows from Def. 2 as Ψ_p is an APA on Player p) and (b) satisfy its specification Φ_p against all other players strategies which satisfy $\Psi_{\neg p} = \bigwedge_{q \neq p} \Psi_q$. Whether (b) holds[4] is tested by intersecting all winning regions $\langle\langle p \rangle\rangle \varphi_p$ (line 6) to obtain the winning region that is achievable by any strategy profile $(\pi_p)_{p \in \mathtt{P}}$ with $\pi_p \vDash \varphi_p$. If this check fails, the assumptions $(\Psi_p)_{p \in \mathtt{P}}$ are strengthened by re-calling RecursiveRCS (line 8).

As $\Psi_p = \mathsf{True}$ for each $p \in \mathtt{P}$ in the first iteration (line 1), we see that the first time ComputeAPA is called in line 8 we have $\Psi'_p = \textsc{ComputeAPA}(G, \Phi_p, p)$. In all further iterations ComputeAPA is called with the restricted specifications $\Phi'_p := \Psi_{\neg p} \wedge \Phi_p$ for all players (line 8). This tightens the objective for Player p by incorporating the knowledge on how other players are *surely* cooperating, as $\Psi_{\neg p}$ is an *under-approximation* of their (accidental) help. As a result, the new assumptions are stricter than earlier ones, i.e., $\mathcal{L}(\Psi'_p) \subseteq \mathcal{L}(\Psi_p)$, and hence, more informative for the computation of rationally contracted strategies.

Upon termination of $\textsc{ComputeRCS}(\mathcal{G})$, the resulting specification profile is indeed a *maximal RCS*, as formalized next.

Theorem 2 ([10, Thm. 1])**.** *Given a P-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p \in \mathtt{P}}$, if the algorithm* $\textsc{ComputeRCS}(\mathcal{G})$ *terminates, then it outputs a* maximal RCS $(\varphi_p)_{p \in \mathtt{P}}$.

Example 4. Consider the game graph in Fig. 1 restricted to set of vertices $\{c, d, e, f, g\}$, with specification $\Phi_0 = \Diamond\Box\{e, f, g\}$ for Player 0 and $\Phi_1 = \Diamond\Box\{e, g\}$ for Player 1, and initial vertex c. Then first we observe that neither player can satisfy their specification by themselves. Then the APA computed by Player 0 is $\Psi_0 = \Lambda_{\textsc{colive}}(e_{cc})$, and that computed by Player 1 is $\Psi_1 = \Lambda_{\textsc{unsafe}}(e_{df}) \wedge \Lambda_{\textsc{colive}}(e_{ec})$. Intuitively, Ψ_0 ensures that the play does not get stuck in vertex c, and progress is made towards Player 0's goal set. Similarly, Ψ_1 ensures that the play does not reach a region from which satisfying Φ_0 becomes infeasible, and ensures progress towards vertex g happens. We note that Player 0 accidentally ends up helping Player 1, by trying to leave c. Hence, in the next iteration, Player p actually can realize their contract $\varphi_p = \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$, which gives us a maximal RCS.

Approximation and Termination. While ComputeRCS outputs a maximal RCS *when it terminates*, it is unfortunately not guaranteed to terminate. This is due to the fact that the procedure ComputeAPA needs a parity game as an input. While $(\Phi_p)_{p \in \mathtt{P}}$ are assumed to be parity objectives, this is general not true for $\Psi_{\neg p} \wedge \Phi_p$ in later calls to RecursiveRCS. Therefore, ComputeAPA might need to change the game graph in each iteration to reduce it to a parity game. While the *language* of the computed APA is guarantee to shrink in every iteration (due to line 8), this does not guarantee termination of Alg. 1 as such a language still contains an infinite number of words. Due to the possibly repeated

[4] Note that $\langle\langle p \rangle\rangle \Psi_p = V$ from Def. 2(ii).

changes in the game graph for APA computation, the finiteness of the underlying model can also not be used as a termination argument.

To address this problem, we provide a modified version of ComputeRCS [10, Sec.5] where the APAs computed by ComputeAPA in line 8 are over-approximated by assumptions which are only implementable and permissive, but not necessarily sufficient. This is done by restricting each assumption template to its safety and co-live part which allows to encode the resulting additional restrictions for players directly as parity specifications over the *original game graph* and thereby ensuring termination. Luckily, as assumptions are only computed on each protagonist themselves, the resulting algorithm is still sound. Moreover, this variant terminates in same time as that for solving parity games (i.e., quasi-polynomial time) and still computes maximal RSCs [10]. However, due to the over-approximation, this variant of ComputeRCS might fail to compute an RCS in every case where ComputeRCS would have succeeded.

4.3 Cooperatively Contracted Specifications

Similar to the rationally contracted specifications, we can also compute *cooperatively* contracted specifications by utilizing APAs. Here, a player can *rely* on other players to help. However, in order to give every player as much freedom as possible in (locally) choosing their strategy, we want to reduce this help to the necessary minimum. In the iterative algorithm for computing cooperatively contracted specifications every player therefore computes how *other* players *must help* her to make her own objective Φ_p realizable. Such an assumption $\Psi_{\neg p}$ on all other players $\mathtt{P}_{\neg p}$ can be computed by the procedure ComputeAPA$(G, \Phi_p, \mathtt{P}_{\neg p})$. In order to isolate the required help for each player, one needs to separate the resulting templates into Player q-templates $\Psi_{\neg p}|_{E_q}$ for each player $q \neq p$. With this, every Player p now gets new objectives from every other player in $\mathtt{P}_{\neg p}$, that specifies the help player p needs to provide for others. Hence, $\Psi'_p \leftarrow \bigwedge_{q \neq p} \Psi_{\neg q}|_{E_p}$ is an additional objective for Player p, strengthening its objective to $\varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$.

This, maybe surprisingly, shows that the computation of contracted strategies for cooperating players is actually identical to their computation for rational players (see line 5 in Alg. 1). The difference however, lies in the use of ComputeAPA to compute Ψ'_p. More importantly, also the interpretation of $(\varphi_p)_{p \in \mathtt{P}}$ being a contracted specification changes for cooperating players. Here, (b), i.e., satisfying $\Psi_{\neg p} \Rightarrow \Phi_p$ is now the easy part, as this follows directly from $\Psi_{\neg p}$ being an APA on $\mathtt{P}_{\neg p}$ for Φ_p. However, (a), i.e., ensuring to provide all requested help to others, contained in Ψ_p, might result in unrealizability of Φ_p. Then Player p must ask for more help from others, and hence, strengthen her assumption on them. The resulting refinement of assumptions requires $\Psi_{\neg p}$ to be strengthened to ComputeAPA$(G, \varphi_p, \mathtt{P}_{\neg p})$ in future iterations.

Computation. Given the above intuition, we introduce the procedure ComputeCCS(G) – which computes CCSs – by modifying the assumption computation in line 8 of Alg. 1 to $\Psi'_p \leftarrow \bigwedge_{q \neq p}$ ComputeAPA$(G, \varphi_p, \mathtt{P}_{\neg q})|_{E_p}$ for all

Algorithm 2. NEGOTIATE($\mathcal{G}$); adapted from [3, Alg.1]

Input: A P-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p\in\mathtt{P}}$.
Output: A cooperatively contracted specification $(\varphi_p)_{p\in\mathtt{P}}$ and templates $(\Lambda_p)_{p\in\mathtt{P}}$.
1: $\mathcal{W}_i, \mathcal{C}_p, \Pi_p, \Psi_{\neg p} \leftarrow$ COMPUTETEMP(G, Φ_p, p), $\forall p \in \mathtt{P}$
2: **if** $\wedge_{p\in\mathtt{P}}(\Pi_p \wedge \Psi_{\neg p})$ is conflict-free **then**
3: $\quad\Psi_p \leftarrow \bigwedge_{q\neq p} \Psi_{\neg q}|_{E_p}$ $\forall p \in \mathtt{P}$
4: $\quad\varphi_p \leftarrow \Psi_p \wedge (\Psi_{\neg p} \Rightarrow \Phi_p)$ $\forall p \in \mathtt{P}$
5: $\quad$**return** $(\varphi_p)_{p\in\mathtt{P}}, (\Psi_p \wedge \Pi_p)_{p\in\mathtt{P}}$
6: **else**
7: $\quad\Phi_p \leftarrow \Phi_p \wedge \square(\bigcap_{p\in\mathtt{P}} \mathcal{W}_i) \wedge \lozenge\square\neg(\bigcup_{p\in\mathtt{P}} \mathcal{C}_p)$, $\forall\ p \in \mathtt{P}$
8: $\quad$**return** NEGOTIATE(G, Φ'_0, Φ'_1)

$p \in \mathtt{P}$. While it can be proven that, whenever COMPUTECCS terminates, the resulting specification profile $(\varphi_p)_{p\in\mathtt{P}}$ is indeed a CCS, it is in general *not maximal*. This is due to the fact that the separation of APAs into Player p templates might make them lose their permissiveness, resulting in potentially non-maximal CCSs. This phenomenon of lacking permissiveness is similar to the observation discussed in Ex. 2, i.e. that winning strategy templates are not maximally permissive in zero-sum games. There we have seen that strategy templates exclude strategies the protagonist can only use if the antagonist (unexpectedly) helps. The same phenomenon occurs when splitting APAs: it is not always needed that all players actually help – if one player decides to help, some obligations can be lifted from other players, leading to cooperative strategies not contained in the resulting CCS.

Interestingly, this over-approximation of assumptions by their restriction to players is already enough to obtain a terminating version of COMPUTECCS. In contrast to the terminating version of COMPUTERCS, we do not need to further restrict templates only to their safety and co-live parts. Due to the now cooperative nature of the game, also live-group templates can be directly encoded as new parity objectives over the same game graph. In particular, this implies that for two-player games (addressed in [3]) which do not require the restriction of APAs (as $|\mathtt{P}_{\neg p}| = 1$ for all $p \in \mathtt{P} = \{1, 2\}$), this terminating version of COMPUTERCS (which is called NEGOTIATE in [3, Alg.1]) indeed computes *maximal CCSs*.

The NEGOTIATE Algorithm for P-Player Games. In order to complete the picture, Alg. 2 presents the formal extension of NEGOTIATE [3, Alg.1] to P-player games, following the structure of COMPUTERCS for an easier exposition in the scope of this paper. Here, the re-encoding of contracted strategies into parity games is realized by an updated version of the procedure COMPUTEAPA$(G, \Phi, \mathtt{P}_{\neg q})$, called COMPUTETEMP$(G, \Phi, q)$, which computes the winning region $\langle\langle\mathtt{P}\rangle\rangle\Phi$, the conflict region $\mathcal{C}$ that needs to be avoided eventually, and a strategy template Π_q for Player q, along with the APA on players $\mathtt{P}_{\neg q}$. With this, the realizability check of the specification profiles $(\Phi'_p)_{p\in\mathtt{P}}$ reduces to checking if all assumption and strategy templates are conflict-free. If this is not

the case, the new parity objective is obtained by adding the requirements to (i) always stay in the winning region and (ii) eventually avoid the conflict region.

Due to the cooperative nature of the game, the algorithm terminates in polynomial time. Furthermore, the use of localness of templates ensures that the algorithm does not depend on the initial vertex, and hence, for every vertex v for which there exists a cooperatively winning strategy profile, i.e., for every $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p \in \mathtt{P}} \Phi_p$ the algorithm outputs a CCS even for the modified game graph (V, E, v). This follows directly from the results in [3], and is formalized below.

Theorem 3. *Given a P-player game $\mathcal{G}$ with game graph $G = (V, E, v_0)$ and parity specifications $(\Phi_p)_{p \in \mathtt{P}}$, the algorithm* Negotiate($\mathcal{G}$) *terminates in $\mathcal{O}(P\,|V|^6)$ time and outputs a profile $(\varphi_p)_{p \in \mathtt{P}}$ that is CCS w.r.t. every game graph (V, E, v) with $v \in \langle\!\langle \mathtt{P} \rangle\!\rangle \bigwedge_{p \in \mathtt{P}} \Phi_p$. If $|\mathtt{P}| = 2$, $(\varphi_p)_{p \in \mathtt{P}}$ is also maximal.*

Example 5. Consider the game graph in Fig. 1 restricted to set of vertices $\{c, d, e, f, g\}$, with specification $\Phi_0 = \square\lozenge\{e\}$ for Player 0 and $\Phi_1 = \lozenge\square\{e, g\}$ for Player 1, and initial vertex c. Again, neither player can satisfy the specifications on their own. Then in Alg. 1 (to compute RCS), the APA computed by Player 0 is $\Psi_0 = \Lambda_{\text{LIVE}}(\{e_{cc}\})$, and that computed by Player 1 is $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{df}) \wedge \Lambda_{\text{COLIVE}}(e_{ec})$. In the next iteration, Player 1 can satisfy her specification $\varphi_1 := \Psi_1 \wedge (\Psi_0 \Rightarrow \Phi_1)$ by herself, but Player 0 can not satisfy $\varphi_0 := \Psi_0 \wedge (\Psi_1 \Rightarrow \Phi_0)$. Moreover, the new APAs and specifications will remain same in the next iterations, and hence, the algorithm will not terminate with a CCS. In fact, for the rational setting, no CSS exists for this game.

However, in the cooperative setting, Player 0 computes the assumption $\Psi_1 = \Lambda_{\text{UNSAFE}}(e_{df}) \wedge \Lambda_{\text{LIVE}}(\{e_{ge}\})$ on Player 1, and strategy template $\Pi_0 = \Lambda_{\text{LIVE}}(\{e_{ce}\})$. Similarly, Player 1 computes $\Psi_0 = \Lambda_{\text{COLIVE}}(e_{cc}, e_{cd})$ on Player 0, and $\Pi_1 = \Lambda_{\text{UNSAFE}}(e_{df}) \wedge \Lambda_{\text{COLIVE}}(e_{ec})$. Then since $\Psi_0 \wedge \Pi_0$ and $\Psi_1 \wedge \Pi_1$ are conflict free, the negotiation terminates with these templates for Player 0 and Player 1 respectively. Intuitively, Player 0 brings the token to e from c, and Player 1 ensures that the token does not leave $\{e, g\}$ infinitely often. Then due to Ψ_1, Player 1 also brings the token to e infinitely often. Hence, on cooperating, both agents can satisfy their respective specification.

Remark 2. In principle, conflict free strategy templates Π_p can be extracted for all players from a computed RCS or CCS $(\varphi_p)_{p \in \mathtt{P}}$ by using Prop. 1 on each φ_p. However, as already hinted at by the previous example, we show that Negotiate($\mathcal{G}$) can directly output conflict-free strategy templates without additional computational overhead [3]. As contracted specifications $(\varphi_p)_{p \in \mathtt{P}}$ fully decentralize a given multi-objective game $\mathcal{G} = (G, (\Phi_p)_{p \in \mathtt{P}})$ into locally realizable specifications φ_p, their respective winning strategy template Π_p ensures that any strategy π_p following Π_p will result in a winning strategy profile $(\Phi_p)_{p \in \mathtt{P}}$ for $\mathcal{G}$.

5 Utilizing Templates Under Partial Observation

This section will *not* discuss the *synthesis* of strategy templates under partial observation. Instead, we assume that the interaction between players is

known during synthesis, but players have to actually *play* under partial observation which requires the extraction of a *partial observation strategy* from (full-observation) strategy templates Π_p.[5]. For simplicity, we use a partial observation setting where players cannot distinguish all vertices in the graph but still take turns when playing. We use the well known knowledge-based abstraction of games under partial observation (see e.g. [6]) to build an abstraction over so-called *knowledge states* for each player, which collects all vertices of the game that a player can currently not distinguish (given the past history of the play). We then intersect all templates of vertices combined in these knowledge states. If the resulting (knowledge-based) template is non-conflicting, a strategy can be extracted in analogy to Theorem. 1.

The extraction of partial observation strategies from templates formalized in this section has not been presented before. Its soundness, however, follows directly from the known properties of knowledge-based abstractions and conflict free templates and is therefore not explicitly proven.

5.1 Preliminaries

As every player might have different partial observation settings, the knowledge abstraction is different for every player. Further, in order to build these abstractions in a sound manner, we need to make Player p choices along their moves explicit by adding a label to Player p transitions in a game graph[6].

p-Labelled Game Graphs. Let $G = (V, E, v_0)$ be a P-player game graph. Then a corresponding p-labled game graph is a tuple ${}^pG = (V, v_0\, {}^p\Sigma, {}^p\Delta)$ s.t.

- ${}^p\Sigma$ is a finite alphabet with $\bot \in {}^p\Sigma$
- $(u, \bot, v) \in {}^p\Delta$ iff $(u, v) \in E_{\neg p}$, and
- $(u, v) \in E_p$ iff there exists a *unique* $\bot \neq \sigma \in {}^p\Sigma$ s.t. $(u, \sigma, v) \in {}^p\Delta$.

Sightly abusing notation, we write ${}^p\Delta(U, \sigma)$ for some $\emptyset \neq U \subseteq V$ and $\sigma \in \Sigma$ to denote all states $v \in V$ s.t. $(u, \sigma, v) \in {}^p\Delta$ for some $u \in U$.

Vertex Covers and Partitions. Vertex subsets $\Gamma \subseteq 2^V \setminus \emptyset$ form a *cover* of V. If for all $\gamma, \gamma' \in \Gamma$ with $\gamma \neq \gamma'$ holds that $\gamma \cap \gamma' = \emptyset$, a cover is called a *partition*. Covers induce a mapping $\Gamma^{\Uparrow} : V \to 2^{\Gamma}$ s.t. $\gamma \in \Gamma^{\Uparrow}(v)$ iff $v \in \gamma$. If Γ is a partition, its holds that $|\Gamma^{\Uparrow}(v)| = 1$. Then we slightly abuse notation to write $\Gamma^{\Uparrow}(v) = \gamma$. We call covers and partitions *player-respecting* if for all $\gamma \in \Gamma$, $u, v \in \gamma$ and $q \in \mathtt{P}$ holds $u \in V_q$ iff $v \in V_q$.

Knowledge-Based Abstractions. Let ${}^pG$ be a p-labelled game graph and ${}^p\Gamma$ a player-respecting[7] *partial observation partition* which groups all states that

[5] This complies with the concept of *output-feedback control* from engineering, where control strategies are synthesized offline under full state information, but need to operate based on measurements containing only partial state information.

[6] Note that such labels are not needed in full observation settings, as discussed in previous sections, as moves are then uniquely determined by the edge successors.

[7] By assuming ${}^p\Gamma$ to be player-respecting, we assume that each player sees who is playing. This retains the turn-based nature in the abstract game. Otherwise, concurrent abstract games are obtained which are out of the scope of this paper.

are indistinguishable for Player p. Then we define a *knowledge-based abstraction* ${}^{p}\widehat{G}$ of ${}^{p}G$ under ${}^{p}\Gamma$ in the usual way, by using the interaction structure between players captured in ${}^{p}G$.

Definition 3. *A* knowledge-based abstraction $\texttt{abstract}({}^{p}G, {}^{p}\Gamma, p)$ *of a* p*-labelled game graph* ${}^{p}G = (V, v_0\, {}^{p}\Sigma, {}^{p}\Delta)$ *under the player-preserving partial observation partition* ${}^{p}\Gamma \subseteq 2^V \setminus \emptyset$ *is a tuple* ${}^{p}\widehat{G} = ({}^{p}\widehat{\Gamma}, {}^{p}\widehat{\gamma}_0, {}^{p}\Sigma, {}^{p}\widehat{\Delta})$ *s.t.*

- ${}^{p}\widehat{\Gamma} \subseteq 2^V$ *and for all* $\widehat{\gamma} \in {}^{p}\widehat{\Gamma}$ *and* $u, v \in \widehat{\gamma}$ *holds that* ${}^{p}\Gamma^{\Uparrow}(v) = {}^{p}\Gamma^{\Uparrow}(u)$*;*
- ${}^{p}\widehat{\gamma}_0 := {}^{p}\Gamma^{\Uparrow}(v_0)$*;*
- $(\widehat{v}, \sigma, \widehat{u}) \in {}^{p}\widehat{\Delta}$ *iff there exists* $\gamma \in {}^{p}\Gamma$ *s.t.* $\widehat{u} = {}^{p}\Delta(\widehat{v}, \sigma) \cap \gamma \neq \emptyset$.

We observe that the state space ${}^{p}\widehat{\Gamma}$ of ${}^{p}\widehat{G}$ is a player- and observation-preserving cover of V with its induced abstraction map ${}^{p}\widehat{\Gamma}^{\Uparrow}$. With this, we denote by ${}^{p}\widehat{\Gamma}_q$ the set of knowledge-states $\widehat{\gamma}$ of Player p which are owned by player q, i.e., $\widehat{\gamma} \subseteq V_q$.

5.2 Extracting Partial Observation Strategies

In this section, we show how a Player p strategy template in a p-labelled game graph ${}^{p}G$ can be abstracted into a Player p strategy template in Player p's knowledge abstraction ${}^{p}\widehat{G}$. In order to formalize this abstraction, we need to first extend strategy templates to labelled games.

Labelled Strategy Templates. Let ${}^{p}G$ be a p-labelled game graph. Then the *enabled state-action pairs* for Player p in ${}^{p}G$ are collected in the set ${}^{p}A := \{(v, \sigma) \in V_p \times {}^{p}\Sigma \mid \exists v' \in V.\ (v, \sigma, v') \in {}^{p}\Delta\}$. Then a p-labelled strategy template is a tuple ${}^{p}\Pi \lhd ({}^{p}S, {}^{p}D, {}^{p}\mathcal{H})$ with ${}^{p}S \subseteq {}^{p}A$, ${}^{p}D \subseteq {}^{p}A$, and ${}^{p}\mathcal{H}$ containing pairs of form $(R, {}^{p}H_\ell)$ for $R \subseteq V$ and ${}^{p}H_\ell \subseteq {}^{p}A$. We note that, due to the deterministic definition of transition labels in ${}^{p}G$ for Player p edges E_p in G, every Player p strategy template over G corresponds to a unique p-labelled strategy template in ${}^{p}G$.

Knowledge-Based Strategy Templates. Given a p-labelled game graph ${}^{p}G$ and a corresponding knowledge-based abstraction ${}^{p}\widehat{G}$ with state space ${}^{p}\widehat{\Gamma}$ and enabled state-action pairs ${}^{p}\widehat{A}$, we extend the partition mapping ${}^{p}\widehat{\Gamma}^{\Uparrow}$ induced by ${}^{p}\widehat{\Gamma}$ to sets of state-action pairs $\alpha \subseteq {}^{p}A$ in the obvious way, i.e., ${}^{p}\widehat{\Gamma}^{\Uparrow}(\alpha) := \{(\widehat{v}, \sigma) \in {}^{p}\widehat{A} \mid (v, \sigma) \in \alpha, \widehat{v} \in {}^{p}\widehat{\Gamma}^{\Uparrow}(v)\}$.

With this, a p-labled strategy template ${}^{p}\Pi$ in ${}^{p}G$ naturally corresponds to an abstract strategy template ${}^{p}\widehat{\Pi}$ in ${}^{p}\widehat{G}$

Definition 4. *Given a* p*-labelled game graph* ${}^{p}G$*, a corresponding knowledge-based abstraction* ${}^{p}\widehat{G}$ *with state space* ${}^{p}\widehat{\Gamma}$ *and a* p*-labelled strategy template* ${}^{p}\Pi \lhd ({}^{p}S, {}^{p}D, {}^{p}\mathcal{H})$ *over* ${}^{p}G$*, its corresponding knowledge-based strategy template* ${}^{p}\widehat{\Pi} \lhd ({}^{p}\widehat{S}, {}^{p}\widehat{D}, {}^{p}\widehat{\mathcal{H}})$ *is defined by* ${}^{p}\widehat{S} := {}^{p}\widehat{\Gamma}^{\Uparrow}({}^{p}S)$, ${}^{p}\widehat{D} := {}^{p}\widehat{\Gamma}^{\Uparrow}({}^{p}D)$, *and* ${}^{p}\widehat{\mathcal{H}} := \{(\widehat{R}, {}^{p}\widehat{H_\ell}) \mid (R, {}^{p}H_\ell) \in {}^{p}\mathcal{H}\}$ *with* $\widehat{R} = \{\widehat{\gamma} \in {}^{p}\widehat{\Gamma} \mid \widehat{\gamma} \cap R \neq \emptyset\}$ *and* ${}^{p}\widehat{H_\ell} = \{{}^{p}\widehat{\Gamma}^{\Uparrow}({}^{p}H) \mid {}^{p}H \in {}^{p}H_\ell\}$.

Given the above construction, one can use a computed strategy template Π_p in the original game graph to compute a p-labelled knowledge-based template ${}^{p}\widehat{\Pi}$ which can be used to extract a Player p strategy ${}^{p}\widehat{\pi}_p$ following ${}^{p}\widehat{\Pi}$ in ${}^{p}\widehat{G}$ via Thm. 1, if ${}^{p}\widehat{\Pi}$ is *conflict-free* in ${}^{p}\widehat{G}$. As knowledge-based abstractions ensure that every history of a play corresponds to a unique Player p state, ${}^{p}\widehat{\pi}_p$ is a well-defined observation-preserving strategy in ${}^{p}G$.

If ${}^{p}\widehat{\Pi}$ is not *conflict-free* in ${}^{p}\widehat{G}$, on can add these conflicts to the original synthesis problem by updating the parity specifications in the iterative algorithms of Sect. 4 and resolve them by another iteration of the respective synthesis algorithms before attempting a strategy extraction again. As this does not require a change in the game graph, the resulting extension of the iterative synthesis algorithms always terminates. It is however not complete, as synthesizing an observation-based winning strategy profile might require to encode the knowledge-based state structure into the negotiation algorithms.

References

1. 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, pp. 211–228. Springer, Cham (2023)
2. Anand, A., Nayak, S.P., Schmuck, A.: Synthesizing permissive winning strategy templates for parity games. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13964, pp. 436–458. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_22
3. Anand, A., Schmuck, A., Nayak, S.P.: Contract-based distributed logical controller synthesis. In: Ábrahám, E., Jr., M.M. (eds.) Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024, Hong Kong SAR, China, May 14-16, 2024, pp. 11:1–11:11. ACM (2024). https://doi.org/10.1145/3641513.3650123
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
5. Belta, C., Yordanov, B., Gol, E.: Formal Methods for Discrete-Time Dynamical Systems, Studies in Systems, Decision and Control, vol. 15. Springer (2017)
6. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for omega-regular games with imperfect information. Logical Methods Comput. Sci. **3** (2007)
7. Chatterjee, K., Henzinger, T.A., Jurdziński, M.: Games with secure equilibria. Theoret. Comput. Sci. **365**(1–2), 67–82 (2006)
8. Jacobs, S., et al.: The reactive synthesis competition (SYNTCOMP): 2018–2021. arXiv preprint arXiv:2206.00251 (2022)
9. Nayak, S.P., Egidio, L.N., Della Rossa, M., Schmuck, A.K., Jungers, R.M.: Context-triggered abstraction-based control design. IEEE Open J. Control Syst. **2**, 277–296 (2023). https://doi.org/10.1109/OJCSYS.2023.3305835

10. Nayak, S.P., Schmuck, A.: Most general winning secure equilibria synthesis in graph games. In: Finkbeiner, B., Kovács, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part III. LNCS, vol. 14572, pp. 173–193. Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_9
11. Reissig, G., Weber, A., Rungger, M.: Feedback refinement relations for the synthesis of symbolic controllers. TAC **62**(4), 1781–1796 (2017)
12. Saha, S., Julius, A.A.: An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications. In: 2016 American Control Conference (ACC), pp. 1105–1110 (2016). https://doi.org/10.1109/ACC.2016.7525063
13. Sanfelice, R.G.: Hybrid Feedback Control. Princeton University Press (2020)
14. Schmuck, A.K., Heim, P., Dimitrova, R., Nayak, S.P.: Localized attractor computations for infinite-state games. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification. CAV 2024. LNCS, vol. 14683, pp. 135–158. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-65633-0_7
15. Tabuada, P.: Verification and Control of Hybrid Systems - A Symbolic Approach. Springer (2009). http://www.springer.com/mathematics/applications/book/978-1-4419-0223-8
16. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoret. Comput. Sci. **200**(1), 135–183 (1998)