Computing Adequately Permissive Assumptions for Synthesis *

Ashwani Anand[1], Kaushik Mallik[2], Satya Prakash Nayak[1](✉), and Anne-Kathrin Schmuck[1]

[1] Max Planck Institute for Software Systems, Kaiserslautern, Germany
{ashwani,sanayak,akschmuck}@mpi-sws.org
[2] Institute of Science and Technology Austria, Klosterneuburg, Austria
kaushik.mallik@ist.ac.at

Abstract. We automatically compute a new class of environment assumptions in two-player turn-based finite graph games which characterize an "adequate cooperation" needed from the environment to allow the system player to win. Given an ω-regular winning condition Φ for the system player, we compute an ω-regular assumption Ψ for the environment player, such that (i) every environment strategy compliant with Ψ allows the system to fulfill Φ (sufficiency), (ii) Ψ can be fulfilled by the environment for every strategy of the system (implementability), and (iii) Ψ does not prevent any cooperative strategy choice (permissiveness). For parity games, which are canonical representations of ω-regular games, we present a polynomial-time algorithm for the symbolic computation of *adequately permissive assumptions* and show that our algorithm runs faster and produces better assumptions than existing approaches—both theoretically and empirically. To the best of our knowledge, for *ω-regular* games, we provide the first algorithm to compute sufficient and implementable environment assumptions that are also *permissive*.

Keywords: Synthesis · Two-player Games · Parity · Permissiveness.

1 Introduction

Two-player ω-regular games on finite graphs are the core algorithmic components in many important problems of computer science and cyber-physical system design. Examples include the synthesis of programs which react to environment inputs, modal μ-calculus model checking, correct-by-design controller synthesis for cyber-physical systems, and supervisory control of autonomous systems.

These problems can be ultimately reduced to an abstract two-player game between an *environment player* and a *system player*, respectively capturing the external unpredictable influences and the system under design, while the game captures the non-trivial interplay between these two parts. A *solution of 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. K. Mallik is supported by the ERC project ERC-2020-AdG 101020093.

© The Author(s) 2023
S. Sankaranarayanan and N. Sharygina (Eds.): TACAS 2023, LNCS 13994, pp. 211–228, 2023.
https://doi.org/10.1007/978-3-031-30820-8_15

game is a set of decisions the system player needs to make to satisfy a given ω-regular temporal property over the states of the game, which is then used to design the sought system or its controller.

Traditionally, two-player games over graphs are solved in a zero-sum fashion, i.e., assuming that the environment will behave arbitrarily and possibly adversarially. Although this approach results in robust system designs, it usually makes the environment too powerful to allow an implementation for the system to exist. However in reality, many of the outlined application areas actually account for some cooperation of system components, especially if they are co-designed. In this scenario it is useful to understand how the environment (i.e., other processes) needs to cooperate to allow for an implementation to exist. This can be formalized by environment assumptions, which are ω-regular temporal properties that restrict the moves of the environment player in a synthesis game. Such assumptions can then be used as additional specifications in other components' synthesis problems to enforce the necessary cooperation (possibly in addition to other local requirements) or can be used to verify existing implementations.

For the reasons outlined above, the automatic computation of assumptions has received significant attention in the reactive synthesis community. It has been used in two-player games [8,6], both in the context of monolithic system design [11,19] as well as distributed system design [18,13].

All these works emphasize two desired properties of assumptions. They should be (i) *sufficient*, i.e., enable the system to win if the environment obeys its assumption and (ii) *implementable*, i.e., prevent the system from falsifying the assumption to vacuously win the game by not even respecting the original specification. In this paper, we claim that there is an important third property — *permissiveness*, i.e. the assumption retains all cooperatively winning plays in the game. This notion is crucial in the setting of distributed synthesis, as here assumptions are generated *before* the implementation of every component is fixed. Therefore, assumptions need to retain *all* feasible ways of cooperation to allow for a distributed implementation to be discovered in a decentralized manner.

While the class of assumptions considered in this paper is motivated by their use for distributed synthesis, this paper focuses only on their formalization and computation, i.e., given a two-player game over a finite graph and an ω-regular winning condition Φ for the system player, we automatically compute an *adequately permissive ω-regular assumption* Ψ for the environment player that formalizes the above intuition by being (i) sufficient, (ii) implementable, and (iii) permissive. The main observation that we exploit is that such *adequately permissive assumptions* (APA for short) can be constructed from three simple templates which can be directly extracted from a cooperative synthesis game leading to a polynomial-time algorithm for their computation. By observing page constrains, we postpone the very interesting but largely orthogonal problem of contract-based distributed synthesis using APAs to future work.

To appreciate the simplicity of the assumption templates we use, consider the game graphs depicted in Fig. 1 where the system and the environment player control the circle and square vertices, respectively. Given the specification $\Phi =$

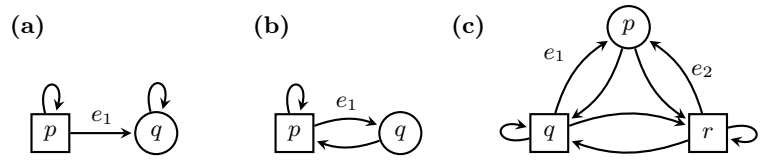

Fig. 1: Game graphs with environment (squares) and system (circles) vertices.

$\Diamond\Box\{p\}$ (which requires the play to eventually only see vertex p), the system player can win the game in Fig. 1 (a) by requiring the environment to fully disable edge e_1. This introduces the first template type—a *safety template*—on e_1. On the other hand, the game in Fig. 1 (b) only requires that e_1 is taken finitely often. This is captured by our second template type—a *co-liveness template*—on e_1. Finally, consider the game in Fig. 1 (c) with the specification $\Phi = \Box\Diamond\{p\}$, i.e. vertex p should be seen infinitely often. Here, the system player wins if whenever the source vertices of edges e_1 and e_2 are seen infinitely often, also one of these edges is taken infinitely often. This is captured by our third template type—a *live group template*—on the edge-group $\{e_1, e_2\}$.

Contribution. The main contribution of this paper is to show that APAs can always be composed from the three outlined assumption templates and can be computed in polynomial time.

Using a set of benchmark examples taken from SYNTCOMP [1] and a prototype implementation of our algorithm in our new tool SImPA, we empirically show that our algorithm is both faster and produces more desirable solutions than existing approaches. In addition, we apply SImPA to the well known 2-client arbiter synthesis benchmark from [21], which is known to only allow for an implementation of the arbiter if the clients' moves are suitably restricted. We show that applying SImPA to the unconstrained arbiter synthesis problem yields assumptions on the clients which are less restrictive but conceptually similar to the ones typically used in the literature.

Related Work. The problem of automatically computing environment assumptions for synthesis was already addressed by Chatterjee et al. [8]. However, their class of assumptions does in general not allow to construct *permissive* assumptions. Further, computing their assumptions is an NP-hard problem, while our algorithm computes APAs in $\mathcal{O}(n^4)$-time for a parity game with n vertices. The difference in the complexity arises because Chatterjee et al. require minimality of the assumptions. On the other hand, we trade minimality for permissiveness which allows us to utilize cooperative games, which are easier to solve.

When considering cooperative solutions of non-zerosum games, related works either fix strategies for both players [7,14], assume a particularly rational behavior of the environment [4] or restrict themselves to safety assumptions [18]. In contrast, we do not make any assumption on how the environment chooses its strategy. Finally, in the context of specification-repair in zerosum games multiple automated methods for repairing environment models exist, e.g., [22,15,16,20,8]. Unfortunately, all of these methods fail to provide permissiveness. A recent work by Cavezza et al. [6] computes a minimally restrictive set of assumptions but only

for GR(1) specifications, which are a strict subclass of the problem considered in our work. To the best of our knowledge, we propose the first fully automated algorithm for computing *permissive* assumptions for general ω-regular 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 short hand for $i \in [a; b] \cap \{0, 2, 4, \ldots\}$ and $i \in [a; b] \cap \{1, 3, 5, \ldots\}$ 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 notations Σ^* and Σ^ω denote the set of finite and infinite words over Σ, respectively, 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 $G = (V, E)$ where (V, E) is a finite directed graph with *vertices* V and *edges* E, and $V = V^0 \uplus V^1$ be a partition of V. Without loss of generality, we assume that for every $v \in V$ there exists $v' \in V$ s.t. $(v, v') \in E$. For the purpose of this paper, the *system* and the *environment* players will be denoted by *Player* 0 and *Player* 1, respectively. A *play* is a finite or infinite sequence of vertices $\rho = v_0 v_1 \ldots \in V^\infty$. A *play prefix* $\mathsf{p} = v_0 v_1 \cdots v_k$ is a finite play.

Winning conditions. Given a game graph G, we consider winning conditions 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 over G, 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 an extended 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 [3].

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 partial function $\pi^i \colon V^* V^i \to V$ such that for every $\mathsf{p}v \in V^* V^i$ for which π is defined, it holds that $\pi^i(\mathsf{p}v) \in E(v)$. 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 $k \in dom(\rho)$. 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 compliant with π^i, and compliant with both π^0 and π^1 in the sets $\mathcal{L}(\pi^i)$ and $\mathcal{L}(\pi^0\pi^1)$, respectively.

Winning. Given a game $\mathcal{G} = (G, \Phi)$, a strategy π^i is (surely) *winning for Player* i if $\mathcal{L}(\pi^i) \subseteq \mathcal{L}(\Phi)$, i.e., a *Player* 0 strategy π^0 is winning if *for every Player* 1 strategy π^1 it holds that $\mathcal{L}(\pi^0\pi^1) \subseteq \mathcal{L}(\Phi)$. Similarly, a fixed strategy

profile (π^0, π^1) is *cooperatively winning* if $\mathcal{L}(\pi^0\pi^1) \subseteq \mathcal{L}(\Phi)$. We say that a vertex $v \in V$ is *winning for Player i* (resp. *cooperatively winning*) if there exists a winning strategy π^i (resp. a cooperatively winning strategy profile (π^0, π^1)) s.t. $\pi^i(v)$ is defined. We collect all winning vertices of *Player i* in the *Player i winning region* $\langle\langle i \rangle\rangle\Phi \subseteq V$ and all cooperatively winning vertices in the *cooperative winning region* $\langle\langle 0,1 \rangle\rangle\Phi$. We note that $\langle\langle i \rangle\rangle\Phi \subseteq \langle\langle 0,1 \rangle\rangle\Phi$ for both $i \in \{0,1\}$.

3 Adequately Permissive Assumptions for Synthesis

Given a two-player game $\mathcal{G}$, the goal of this paper is to compute assumptions on *Player* 1 (i.e., the environment), such that both players cooperate *just enough* to fulfill Φ while retaining all possible cooperative strategy choices. Towards a formalization of this intuition, we define winning under assumptions.

Definition 1. *Let $\mathcal{G} = (G = (V, E), \Phi)$ be a game and Ψ be an LTL formula over V. Then a Player 0 strategy π^0 is winning in $\mathcal{G}$ under assumption Ψ, if* for every *Player 1 strategy π^1 s.t. $\mathcal{L}(\pi^1) \subseteq \mathcal{L}(\Psi)$ it holds that $\mathcal{L}(\pi^0\pi^1) \subseteq \mathcal{L}(\Phi)$. We denote by $\langle\langle 0 \rangle\rangle_\Psi \Phi$ the set of vertices from which such a Player 0 strategy exists.*

We remark that the 'winning-under-assumption' strategies π^0 from Def. 1 satisfy two simple but interesting properties — *anti-monotonicity* (if π^0 is winning under an assumption, then it is so under every stronger assumption), and *conjunctivity* (if π^0 is winning under two different assumptions, then it is so under their conjunction). However, it does not satisfy *disjunctivity* (see [2, Sec. 3.1] for an example). In addition, we remark that the definition of 'winning-under-assumption' in terms of plays (rather than strategies) might seem more natural to some readers. We refer these readers to the full version of the paper [2, Sec. 3.1] for an in-depth discussion on the differences of these definitions.

We now see that the assumption Ψ introduced in Def. 1 *weakens* the strategy choices of the environment player (*Player* 1). We call assumptions *sufficient* if this weakening is strong enough to allow *Player* 0 to win from every vertex in the cooperative winning region.

Definition 2. *An assumption Ψ is* sufficient *for (G, Φ) if $\langle\langle 0 \rangle\rangle_\Psi \Phi \supseteq \langle\langle 0,1 \rangle\rangle\Phi$.*

Unfortunately, sufficient assumptions can be abused to change the given synthesis problem in an unintended way. Consider for instance the game in Fig. 2 (left) with $\Phi = \square\lozenge\{v_0\}$ and $\Psi = \square\lozenge e_1$. Here, there is no strategy π^1 for *Player* 1 such that $\mathcal{L}(\pi^1) \subseteq \mathcal{L}(\Psi)$ as the system can always falsify the assumption by simply not choosing e_1 infinitely often in v_1. Therefore, any *Player* 0 strategy is winning under assumption even if Φ is violated. The assumption Ψ, however, is trivially sufficient, as $\langle\langle 0 \rangle\rangle_\Psi \Phi = V$. In order to prevent sufficient assumptions to be falsifiable and thereby enabling vacuous winning, we define the notion of *implementability*, which ensures that Ψ solely restricts *Player* 1 moves.

Definition 3. *An assumption Ψ is* implementable *for (G, Φ) if $\langle\langle 1 \rangle\rangle\Psi = V$.*

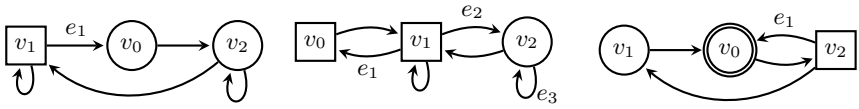

Fig. 2: Two-player games with *Player* 1 (squares) and *Player* 0 (circles) vertices.

A sufficient and implementable assumption ensures that the cooperative winning region of the original game coincides with the winning region under that assumption, i.e., $\langle\langle 0 \rangle\rangle_{\Psi} \Phi = \langle\langle 0, 1 \rangle\rangle \Phi$. However, all cooperative strategy choices of both players might still not be retained, which is ensured by the notion of *permissiveness*.

Definition 4. *An assumption* Ψ *is* permissive *for* (G, Φ) *if* $\mathcal{L}(\Phi) \subseteq \mathcal{L}(\Psi)$.

This notion of permissiveness is motivated by the intended use of assumptions for compositional synthesis. In the simplest scenario of two interacting processes, two synthesis tasks—one for each process—are considered in parallel. Here, generated assumptions in one synthesis task are used as additional specifications in the other synthesis problem. Therefore, permissiveness is crucial to not "skip" over possible cooperative solutions—each synthesis task needs to keep all allowed strategy choices for both players intact to allow for compositional reasoning. This scenario is illustrated in the following example to motivate the considered class of assumptions. Formalizing assumption-based compositional synthesis in general is however out of the scope of this paper.

Example 1. Consider the (non-zerosum) two-player game in Fig. 2 (middle) with two different specifications for both players, namely $\Phi_0 = \Diamond\Box\{v_1, v_2\}$ and $\Phi_1 = \Diamond\Box\{v_1\}$. Now consider two candidate assumptions $\Psi_0 = \Diamond\Box\neg e_1$ and $\Psi_0' = (\Box\Diamond v_1 \implies \Box\Diamond e_2)$ on *Player* 1. Notice that both assumptions are sufficient and implementable for (G, Φ_0). However, Ψ_0' does not allow the play $\{v_1\}^\omega$ and hence is not permissive whereas Ψ_0 is permissive for (G, Φ_0). As a consequence, there is no way *Player* 1 can satisfy both her objective Φ_1 and the assumption Ψ_0' even if *Player* 0 cooperates, since $\mathcal{L}(\Phi_1) \cap \mathcal{L}(\Psi_0') = \emptyset$. However, under the assumption Ψ_0 on *Player* 1 and assumption $\Psi_1 = \Diamond\Box\neg e_3$ on *Player* 0 (which is sufficient and implementable for (G, Φ_1) if we interchange the vertices of the players), they can satisfy both their own objectives and the assumptions on themselves. Therefore, they can collectively satisfy both their objectives.

We also remark that for this example, the algorithm in [9] outputs Ψ_0' as the desired assumption for game (G, Φ_0) and their used assumption formalism is not rich enough to capture assumption Ψ_0. This shows that the assumption type we are interested in is not computable by the algorithm from [9].

Definition 5. *An assumption* Ψ *is called* adequately permissive *(an APA for short) for* (G, Φ) *if it is sufficient, implementable and permissive.*

4 Computing Adequately Permissive Assumptions (APA)

In this section, we present our algorithm to compute *adequately permissive assumptions* (APA for short) for *parity games*, which are canonical representations

of ω-regular games. For a gradual exposition of the topic, we first present algorithms for simpler winning conditions, namely safety (Sec. 4.2), Büchi (Sec. 4.3), and Co-Büchi (Sec. 4.4), which are used as building blocks while presenting the algorithm for parity games (Sec. 4.5). All proofs omitted can be found in the full version [2]. Let us first introduce some preliminaries.

4.1 Preliminaries

We use symbolic fixpoint algorithms expressed in the μ-calculus [17] to compute the winning regions and to generate assumptions in simple post-processing steps.

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 we define two types of predecessor operators:

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

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

$$\mathsf{cpre}^{a,1}_G(U) = \mathsf{cpre}^a_G(U) \cup U \tag{3}$$

$$\mathsf{cpre}^{a,i}_G(U) = \mathsf{cpre}^a_G(\mathsf{cpre}^{a,i-1}_G(U)) \cup \mathsf{cpre}^{a,i-1}_G(U) \text{ with } i \geq 1 \tag{4}$$

The predecessor operator $\mathsf{pre}_G(U)$ computes the set of vertices with at least one successor in U. The controllable predecessor operators $\mathsf{cpre}^a_G(U)$ and $\mathsf{cpre}^{a,i}_G(U)$ compute the set of vertices from which *Player* a can force visiting U in *at most one* and i steps respectively. In the following, we introduce the attractor operator $\mathsf{attr}^a_G(U)$ that computes the set of vertices from which *Player* a can force at least a single visit to U in *finitely many but nonzero*[3] steps:

$$\mathsf{attr}^a_G(U) = \left(\bigcup_{i \geq 1} \mathsf{cpre}^{a,i}(U)\right) \backslash U \tag{5}$$

When clear from the context, we drop the subscript G from these operators.

Fixpoint Algorithms in the μ-calculus. μ-calculus [17] offers a succinct representation of symbolic algorithms (i.e., algorithms manipulating sets of vertices instead of individual vertices) over a game graph G. The formulas of the μ-calculus, interpreted over a 2-player game graph G, are given by the grammar

$$\phi := p \mid X \mid \phi \cup \phi \mid \phi \cap \phi \mid pre(\phi) \mid \mu X.\phi \mid \nu X.\phi$$

where p ranges over subsets of V, X ranges over a set of formal variables, pre ranges over monotone set transformers in $\{\mathsf{pre}, \mathsf{cpre}^a, \mathsf{attr}^a\}$, and μ and ν denote, respectively, the least and the greatest fixed point of the functional defined as $X \mapsto \phi(X)$. Since the operations $\cup, \cap$, and the set transformers pre are all monotonic, the fixed points are guaranteed to exist, due to the Knaster-Tarski Theorem [5]. We omit the (standard) semantics of formulas (see [17]).

A μ-calculus formula evaluates to a set of vertices over G, and the set can be computed by induction over the structure of the formula, where the fixed points are evaluated by iteration. The reader may note that pre, cpre and attr can be computed in time polynomial in number of vertices.

[3] In existing literature, usually $U \subseteq \mathsf{attr}^a(U)$, i.e., $\mathsf{attr}^a(U)$ contains vertices from which U is visited in zero steps. We exclude U from $\mathsf{attr}^a(U)$ for a technical reason.

4.2 Safety Games

A safety game is a game $\mathcal{G} = (G, \Phi)$ with $\Phi \coloneqq \Box U$ for some $U \subseteq V$, and a play fulfills Φ if it never leaves U. APAs for safety games disallow every *Player* 1 move that leaves the cooperative winning region in G w.r.t. *Safety*(U). This is formalized in the following theorem.

Theorem 1. *Let $\mathcal{G} = (G = (V, E), \Box U)$ be a safety game, $Z^* = \nu Y.U \cap \mathsf{pre}(Y)$, and $S = \{(u, v) \in E \mid (u \in V^1 \cap Z^*) \wedge (v \notin Z^*)\}$. Then $Z^* = \langle\langle 0, 1 \rangle\rangle \Box U$ and* [4]

$$\Psi_{\text{UNSAFE}}(S) \coloneqq \Box \bigwedge_{e \in S} \neg e, \tag{6}$$

is an APA for the game $\mathcal{G}$. We denote by UnsafeA(G, U) *the algorithm computing S as above, which runs in time $\mathcal{O}(n^2)$, where $n = |V|$.*

We call the LTL formula in (6) a *safety template* and assumptions that solely use this template *safety assumptions.*

4.3 Live Group Assumptions for Büchi Games

Büchi games. A Büchi game is a game $\mathcal{G} = (G, \Phi)$ where $\Phi = \Box\Diamond U$ for some $U \subseteq V$. Intuitively, a play is winning for a Büchi game if it visits the vertex set U infinitely often. We first recall that the cooperative winning region $\langle\langle 0, 1 \rangle\rangle \Box\Diamond U$ can be computed by a two-nested symbolic fixpoint algorithm [10]

$$\text{Büchi}(G, U) := \nu Y.\mu X.\ (U \cap \mathsf{pre}(Y)) \cup (\mathsf{pre}(X)). \tag{7}$$

Live group templates. Given the standard algorithm in (7), the set X^i computed in the i-th iteration of the fixpoint variable X in the last iteration of Y actually carries a lot of information to construct a very useful assumption for the Büchi game $\mathcal{G}$. To see this, recall that X^i contains all vertices which have an edge to vertices which can reach U in at most $i - 1$ steps [10, sec. 3.2]. Hence, for all *Player* 1 vertices in $X^i \setminus X^{i-1}$ we need to assume that *Player* 1 always eventually makes progress towards U by moving to X^i. This can be formalized by a so called live group template.

Definition 6. *Let $G = (V, E)$ be a game graph. Then 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*

$$\Psi_{\text{LIVE}}(H^\ell) \coloneqq \bigwedge_{i \geq 0} \Box\Diamond src(H_i) \implies \Box\Diamond H_i. \tag{8}$$

[4] We use $e = (u, v)$ in LTL formulas as a syntactic sugar for $u \wedge \bigcirc v$, where $\bigcirc$ is the LTL *next* operator. A set of edges $E' = \{e_i\}_{i \in [0;k]}$, when used as atomic proposition, is a syntactic sugar for $\bigvee_{i \in [0;k]} e_i$.

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. We will use this template to give the assumptions for Büchi games.

Remark 1. Note that the assumptions computed by Chatterjee et al. [8] uses *live edges*, i.e., singleton live groups, and hence, they are less expressive. In particular, there are instances of Büchi games, where the permissive assumptions can not be expressed using live edges but they can be using live groups, e.g., in Fig. 1 (c) the live edge assumption $\square\lozenge e_1 \wedge \square\lozenge e_2$ is sufficient but not permissive, whereas the live group assumption $\square\lozenge src(H) \implies \square\lozenge H$ with $H = \{e_1, e_2\}$ is an APA.

In the context of the fixpoint computation of (7), we can construct live groups $H^\ell = \{H_i\}_{i \geq 0}$ where each H_i contains all edges of *Player* 1 which originate in $X^i \setminus X^{i-1}$ and end in X^{i-1}. Then the live group assumption in (8) precisely captures the intuition that, in order to visit U infinitely often, *Player* 1 should take edges in H_i infinitely often if vertices in $src(H_i)$ are seen infinitely often. Unfortunately, it turns out that this live group assumption is not *permissive*. The reason is that it restricts *Player* 1 also on those vertices from which she will anyway go towards U. For example, consider the game in Fig. 2 (right). Here defining live groups through computations of (10), will mark e_1 as a live group, but then $(v_2 v_1 v_0)^\omega$ will be in $\mathcal{L}(\Phi)$ but not in the language of the assumption. Here the permissive assumption would be $\Psi = \text{TRUE}$.

Accelerated fixpoint computation. In order to provide permissiveness, we use a slightly modified fixpoint algorithm that computes the same set Z^* but allows us to extract *permissive* assumptions directly from the fixpoint computations. Towards this goal, we introduce the *together predecessor operator*.

$$\mathsf{tpre}_G(U) = \mathsf{attr}^0_G(U) \cup \mathsf{cpre}^1_G(\mathsf{attr}^0_G(U) \cup U). \tag{9}$$

Intuitively, tpre adds all vertices from which *Player* 0 does not need any cooperation to reach U in every iteration of the fixpoint computation. The interesting observation we make is that substituting the inner pre operator in (7) by tpre does not change the computed set but only accelerates the computation. This is formalized in the next proposition and visualized in Fig. 3.

Proposition 1. *Let* $\mathcal{G} = (G, \square\lozenge U)$ *be a Büchi game and*

$$\text{TBÜCHI}(G, U) = \nu Y.\mu X.\ (U \cap \mathsf{pre}(Y)) \cup (\mathsf{tpre}(X)). \tag{10}$$

Then $\text{TBÜCHI}(G, U) = \text{BÜCHI}(G, U) = \langle\!\langle 0, 1 \rangle\!\rangle \square\lozenge U$.

Prop. 1 follows from the correctness proof of (7) by using the observation that for all $U \subseteq V$ we have $\mu X.\ U \cup \mathsf{pre}(X) = \mu X.\ U \cup \mathsf{tpre}(X)$.

Computing live group assumptions. Intuitively, the operator tpre_G computes the union of (i) the set of vertices from which *Player* 0 can reach U in a finite number of steps with no cooperation from *Player* 1 and (ii) the set of *Player* 1 vertices from which *Player* 0 can reach U with at most *one-time* cooperation from *Player* 1. Looking at Fig. 3, case (i) is indicated by the dotted line,

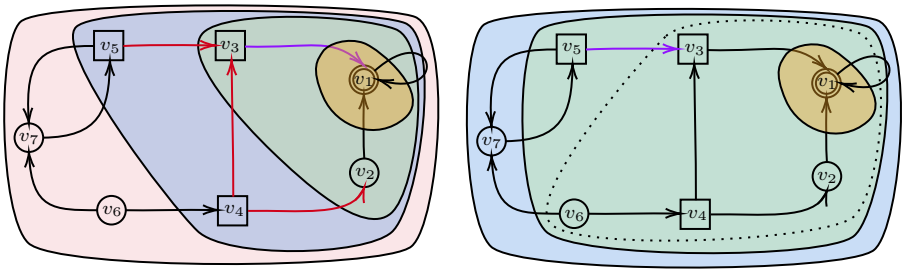

Fig. 3: Computation of $\mu X.\, U \cup \mathsf{pre}(X)$ (left) and $\mu X.\, U \cup \mathsf{tpre}(X)$ (right). Each colored region describes one iteration over X. The dotted region on the right is added by the attr part of tpre, and this allows only the vertex v_5 to be in $\mathit{front}(\{v_1\})$. Each set of the same colored edges defines a live transition group.

while case (ii) corresponds to the last added *Player* 1 vertex (e.g., v_5). Hence, we need to capture the cooperation needed by *Player* 1 only from the vertices added last, which we call the *frontier* of U in G and are formalized as follows:

$$\mathit{front}(U) := \mathsf{tpre}_G(U) \setminus \mathsf{attr}^0_G(U). \tag{11}$$

It is easy to see that, indeed $\mathit{front}(U) \subseteq V^1$, as whenever $v \in \mathit{front}(U) \cap V^0$, then it would have been the case that $v \in \mathsf{attr}^0_G(U)$ via (10).

Defining live groups based on frontiers instead of all elements in X^i indeed yields the desired permissive assumption for Büchi games. By observing that we additionally need to ensure that *Player* 1 never leaves the cooperative winning region by a simple safety assumption, we get the following result, which is the main contribution of this section.

Theorem 2. *Let* $\mathcal{G} = (G = (V, E), \Phi = \Box\Diamond U)$ *be a Büchi game with* $Z^* = \textsc{TBüchi}(G, U)$ *and* $H^\ell = \{H_i\}_{i \geq 0}$ *s.t.*

$$\emptyset \neq H_i := (\mathit{front}(X^i) \times (X^{i+1} \setminus \mathit{front}(X^i))) \cap E, \tag{12}$$

where X^i *is the set computed in the* i*-th iteration of the computation over* X *and in the last iteration of the computation over* Y *in* TBüchi. *Then* $\Psi = \Psi_{\textsc{unsafe}}(S) \wedge \Psi_{\textsc{live}}(H^\ell)$ *is an APA for* $\mathcal{G}$*, where* $S = \textsc{UnsafeA}(G, U)$. *We write* $\textsc{LiveA}(G, U)$ *to denote the algorithm to construct live groups* H^ℓ *as above, which runs in time* $\mathcal{O}(n^3)$*, where* $n = |V|$.

In fact, there is a faster algorithm for computation of APAs for Büchi games, that runs in time linear in the size of the graph, which we present in the full version [2]. We chose to present the μ-calculus based algorithm here, because it provides more insights into the nature of live groups.

4.4 Co-Liveness Assumptions in Co-Büchi Games

A co-Büchi game is the *dual* of a Büchi game, where a winning play should visit a designated set of vertices only finitely many times. Formally, a co-Büchi game

is a tuple $\mathcal{G} = (G, \Phi)$ where $\Phi = \Diamond\Box U$ for some $U \subseteq V$. The standard symbolic algorithm to compute the cooperative winning region is as follows:

$$\textsc{CoBüchi}(G, U) := \mu X.\nu Y.\ (U \cap \mathsf{pre}(Y)) \cup (\mathsf{pre}(X)). \tag{13}$$

As before, the sets X^i obtained in the i-th computation of X during the evaluation of (13) carry essential information for constructing assumptions. Intuitively, X^1 gives precisely the set of vertices from which the play can stay in U with *Player* 1's cooperation and we would like an assumption to capture the fact that we do not want *Player* 1 to go further away from X^1 infinitely often. This observation is naturally described by so called co-liveness templates.

Definition 7. *Let $G = (V, E)$ be a game graph and $D \subseteq V \times V$ a set of edges. Then a* co-liveness template *over G w.r.t. D is defined by the LTL formula*

$$\Psi_{\textsc{colive}}(D) \coloneqq \Diamond\Box \bigwedge_{e \in D} \neg e. \tag{14}$$

The assumptions employing co-liveness templates will be called co-liveness assumptions. With this, we can state the main result of this section.

Theorem 3. *Let $\mathcal{G} = (G = (V, E), \Diamond\Box U)$, $Z^* = \textsc{CoBüchi}(G, U)$ and*

$$D = \Big(\left[(X^1 \cap V^1) \times (Z^* \setminus X^1)\right] \ \cup \left[\bigcup_{i>1} (X^i \cap V^1) \times (Z^* \setminus X^{i-1})\right]\Big) \cap E, \tag{15}$$

where X^i is the set computed in the i-th iteration of fixpoint variable X in CoBüchi. *Then $\Psi = \Psi_{\textsc{unsafe}}(S) \wedge \Psi_{\textsc{colive}}(D)$ is an APA for $\mathcal{G}$, where $S =$ $\textsc{UnsafeA}(G, U)$. We write $\textsc{CoLiveA}(G, U)$ to denote the algorithm constructing co-live edges D as above which runs in time $\mathcal{O}(n^3)$, where $n = |V|$.*

We observe that X_1 is a subset of U such that if a play reaches X^1, *Player* 0 and *Player* 1 can cooperatively keep the play in X^1. To do so, we ensure via the definition of D in (15) that *Player* 1 can only leave X^1 finitely often. Moreover, with the other co-live edges in D, we ensure that *Player* 1 can only go away from X^1 finitely often, and hence if *Player* 0 plays their strategy to reach X^1 and then stay there, the play will be winning. The permissiveness of the assumption comes from the observation that if co-liveness is violated, then *Player* 1 takes a co-live edge infinitely often, and hence leaves X^1 infinitely often, implying leaving U infinitely often.

We again present a faster algorithm that runs in time linear in size of the graph for computation of APAs for co-Büchi games in the full version [2].

4.5 APA Assumptions for Parity Games

Parity games. Let $G = (V, E)$ be a game graph, and $C = \{C_0, \ldots, C_k\}$ be a set of subsets of vertices which form a partition of V. Then the game $\mathcal{G} = (G, \Phi)$ is called a *parity game* if

$$\Phi = \mathit{Parity}(C) \coloneqq \bigvee\nolimits_{i \in_{\text{odd}}[0;k]} \Box\Diamond C_i \implies \bigvee\nolimits_{j \in_{\text{even}}[i+1;k]} \Box\Diamond C_j. \tag{16}$$

The set C is called the *priority set* and a vertex v in the set C_i, for $i \in [1;k]$, is said to have *priority* i. An infinite play ρ is winning for $\Phi = Parity(C)$ if the highest priority appearing infinitely often along ρ is even.

Conditional live group templates. As seen in the previous sections, for games with simple winning conditions which require visiting a fixed set of edges infinitely or only finitely often, a single assumption (conjoined with a simple safety assumption) suffices to characterize APAs, as there is just one way to win. However, in general parity games, there are usually multiple ways of winning: for example, in parity games with priorities $\{0, 1, 2\}$, a play will be winning if either (i) it only infinitely often sees vertices of priority 0, or (ii) it sees priority 1 infinitely often but also sees priority 2 infinitely often. Intuitively, winning option (i) requires the use of co-liveness assumptions as in Sec. 4.4. However, winning option (ii) actually requires the live group assumptions discussed in Sec. 4.3 to be *conditional* on whether certain states with priority 1 have actually been visited infinitely often. This is formalized by generalizing live group templates to *conditional live group templates.*

Definition 8. *Let $G = (V, E)$ be a game graph. Then a* conditional live group *over G is a pair (R, H^ℓ), where $R \subseteq V$ and H^ℓ is a live group. Given a set of conditional live groups $\mathcal{H}^\ell$, a* conditional live group template *is the LTL formula*

$$\Psi_{\text{COND}}(\mathcal{H}^\ell) := \bigwedge_{(R,H^\ell)\in\mathcal{H}^\ell} \left(\Box\Diamond R \implies \Psi_{\text{LIVE}}(H^\ell)\right). \tag{17}$$

Again, the assumptions employing conditional live group templates will be called conditional live group assumptions. With the generalization of live group assumptions to *conditional* live group assumptions, we actually have all the ingredients to define an APA for parity games as a conjunction

$$\Psi = \Psi_{\text{UNSAFE}}(S) \wedge \Psi_{\text{COLIVE}}(D) \wedge \Psi_{\text{COND}}(\mathcal{H}^\ell) \tag{18}$$

of a safety, a co-liveness, and a conditional live group assumptions. Intuitively, we use (i) a safety assumption to prevent *Player* 1 to leave the cooperative winning region, (ii) a co-live assumption for each winning option that requires seeing a particular *odd* priority only finitely often, and (iii) a conditional live group assumption for each winning option that requires seeing an *even* priority infinitely often if certain *odd* priority have been seen infinitely often. The remainder of this section gives an algorithm (Alg. 1) to compute the actual safety, co-live and conditional live group sets S, D and $\mathcal{H}^\ell$, respectively, and proves that the resulting assumption Ψ (as in (18)) is actually an APA for the parity game $\mathcal{G}$.

Computing APAs. The computation of unsafe, co-live, and conditional live group sets S, D, and $\mathcal{H}^\ell$ to make Ψ in (18) an APA is formalized in Alg. 1. Alg. 1 utilizes the standard fixpoint algorithm $\textsc{Parity}(G, C)$ [12] to compute the cooperative winning region for a parity game $\mathcal{G}$, defined as

$$\textsc{Parity}(G, C) := \tau X_d \cdots \nu X_2\ \mu X_1\ \nu X_0 . \bigcup_{i\in[0;d]} (C_i \cap \mathsf{pre}(X_i)), \tag{19}$$

where τ is ν if d is even, and μ otherwise. In addition, Alg. 1 involves the algorithms UnsafeA (Thm. 1), LiveA (Thm. 2), and CoLiveA (Thm. 3) to

Algorithm 1 ParityAssumption

Input: $G = (V, E)$, $C : V \to \{0, 1, \ldots\}$
Output: Ψ
1: $Z^* \leftarrow \textsc{Parity}(G, C)$
2: $S \leftarrow \textsc{UnsafeA}(G, Z^*)$
3: $G \leftarrow G|_{Z^*}$, $C \leftarrow C|_{Z^*}$
4: $(D, \mathcal{H}^\ell) \leftarrow \textsc{ComputeSets}((G, C), \emptyset, \emptyset)$
5: **return** $S, D, \mathcal{H}^\ell$

6: **procedure** $\textsc{ComputeSets}((G, C), D, \mathcal{H}^\ell)$
7: $d \leftarrow \max\{i \mid C_i \neq \emptyset\}$
8: **if** d is odd **then**
9: $W_{\neg d} \leftarrow \textsc{Parity}(G|_{V \setminus C_d}, C)$
10: $D \leftarrow D \cup \textsc{CoLiveA}(G, W_{\neg d})$
11: **else**
12: $W_d \leftarrow \textsc{Büchi}(G, C_d)$, $W_{\neg d} \leftarrow V \setminus W_d$
13: **for all** odd $i \in [0; d]$ **do**
14: $\mathcal{H}^\ell \leftarrow \mathcal{H}^\ell \cup (W_d \cap C_i, \textsc{LiveA}(G|_{W_d}, C_{i+1} \cup C_{i+3} \cdots \cup C_d))$
15: **if** $d > 0$ **then**
16: $G \leftarrow G|_{W_{\neg d}}$, $C_0 \leftarrow C_0 \cup C_d$, $C_d \leftarrow \emptyset$
17: $\textsc{ComputeSets}((G, C), D, \mathcal{H}^\ell)$
18: **else**
19: **return** $(D, \mathcal{H}^\ell)$

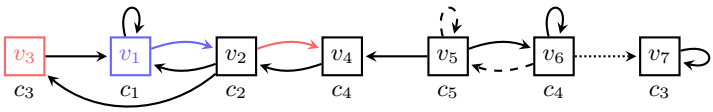

Fig. 4: A parity game, where a vertex with priority i has label c_i. The dotted edges are the unsafe edges, the dashed edges are the co-live edges, and every similarly colored vertex-edge pair forms a conditional live group.

compute safety, live group, and co-liveness assumptions in an iterative manner. In addition, $G|_U := \big(U, U^0, U^1, E'\big)$ s.t. $U^0 := V^0 \cap U$, $U^1 := V^1 \cap U$, and $E' := E \cap (U \times U)$ denotes the restriction of a game graph $G := \big(V, V^0, V^1, E\big)$ to a subset of its vertices $U \subseteq V$. Further, $C|_U$ denotes the restriction of the priority set C from V to $U \subseteq V$.

We illustrate the steps of Alg. 1 by an example depicted in Fig. 4. In line 1, we compute the cooperative winning region Z^* of the entire game, to find that the parity condition cannot be satisfied from vertex v_7 even with cooperation, i.e., $Z^* = \{v_1, \ldots, v_6\}$. So we put the edge (v_6, v_7) in a safety template, restrict the game to $G = G|_{Z^*}$ and run ComputeSets on the new restricted game.

In the new game G the highest priority is odd ($d = 5$), hence we execute lines 9-10. Now a play would be winning only if eventually the play does not see v_5 any more. Hence, in step 9, we find the region $W_{\neg 5} = \{v_1, \ldots, v_4, v_6\}$ of the restricted graph $G|_{V \setminus C_5}$ (only containing nodes v_i with priority $C(v_i) < 5$)) from where we can satisfy the parity condition without seeing v_5. We then make sure that we do not leave $W_{\neg 5}$ to visit v_5 in the game G infinitely often by executing $\textsc{CoLiveA}(G, W_{\neg 5})$ in line 10, making the edges (v_5, v_5) and (v_6, v_5) co-live.

Once we restrict a play from visiting v_5 infinitely often, we only need to focus on satisfying parity without visiting v_5 within $W_{\neg 5}$. This observation allows us

to further restrict our computation to the game $\mathcal{G} = \mathcal{G}|_{W_{\neg 5}}$ in line 16, where we also update the priorities to only range from 0 to 4. In our example this step does not change anything. We then re-execute ComputeSets on this game.

In the restricted graph, the highest priority is 4 which is even, hence we execute lines 12-14. One way of winning in this game is to visit C_4 infinitely often, so we compute the respective cooperative winning region W_4 in line 12. In our example we have $W_4 = W_{\neg 5} = \{v_1, \ldots, v_4, v_6\}$. Now, to ensure that from the vertices from which we can cooperatively see 4, we actually win, we have to make sure that every time a lower odd priority vertex is visited infinitely often, a higher priority is also visited. This can be ensured by conditional live group fairness as computed in line 14. For every odd priority $i < 4$, (i.e, for $i = 1$ and $i = 3$) we have to make sure that either 2 or 4 (if $i = 1$) or 4 (if $i = 3$) is visited infinitely often. The resulting live groups $\mathcal{H}_i^\ell = (R_i, H_i^\ell)$ collect all vertices in W_4 with priority i in R_i and all live groups allowing to see even priorities j with $i < j \leq 4$ in H_i^ℓ, where the latter is computed using the fixed-point algorithm LiveA to compute live groups. The resulting live groups for $i = 1$ (blue) and $i = 3$ (red) are depicted in Fig. 4 and given by $(\{v_1\}, \{(v_1, v_2)\})$ and $(\{v_3\}, \{(v_2, v_4)\}, \{(v_1, v_2)\})$, respectively.

At this point we have $W_{\neg 4} = \emptyset$, making the game graph computed in line 16 empty, and the algorithm eventually terminates after iteratively removing all priorities from C by running ComputeSets (without any computations, as $\mathcal{G}$ is empty) for priorities 3, 2 and 1. In a different game graph, the reasoning done for priorities 5 and 4 above can be repeated for lower priorities if there are other parts of the game graph not contained in W_4, from where the game can be won by seeing priority 2 infinitely often. The main insight into the correctness of the outlined algorithm is that all computed assumptions can be conjoined to obtain an APA for the original parity game.

With Alg. 1 in place, we now state the main result of the entire paper.

Theorem 4. *Let $\mathcal{G} = (G, Parity(C))$ be a parity game such that $(S, D, \mathcal{H}^\ell) =$ ParityAssumption(G, C). Then $\Psi = \Psi_{\text{UNSAFE}}(S) \wedge \Psi_{\text{COLIVE}}(D) \wedge \Psi_{\text{COND}}(\mathcal{H}^\ell)$ is an APA for $\mathcal{G}$. Moreover, Alg. 1 terminates in time $\mathcal{O}(n^4)$, where $n = |V|$.*

5 Experimental Evaluation

We have developed a C++-based prototype tool SImPA[5] computing **S**ufficient, **Im**plementable and **P**ermissive **A**ssumptions for Büchi, co-Büchi, and parity games. We first compare SImPA against the closest related tool GIST [9] in Sec. 5.1. We then show that SImPA gives small and meaningful assumptions for the well-known 2-client arbiter synthesis problem from [21] in Sec. 5.2.

[5] Repository URL: https://gitlab.mpi-sws.org/kmallik/simpa

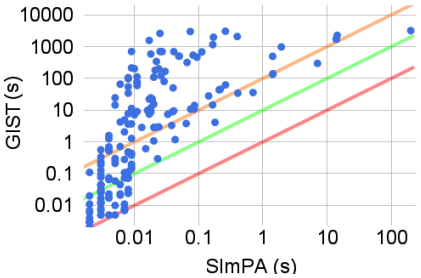

Fig. 5: Running times of SImPA vs GIST (in seconds, log-scale)

	SImPA	GIST
Mean-time	64.8s	1079.0s
Non-timeout mean-time	64.8s	209.2s
Timeouts (1hr)	0(0%)	59(26%)
No assumption generated	0(0%)	20(9%)
Faster	230(100%)	0(0%)

Table 1: Summary of the experimental results

5.1 Performance Evaluation

We compare the effectiveness of our tool against a re-implementation of GIST [9], which is not available anymore [6]. GIST originally computes assumptions only enabling a particular initial vertex to become winning for *Player* 0. However, for the experiments, we run GIST until one of the cooperatively winning vertices is not winning anymore. Since GIST starts with a maximal assumption and shrinks it until a fixed initial vertex is not winning anymore, our modification makes GIST faster as the modified termination condition is satisfied earlier. Owing to the non-dependence of our tool and dependence of GIST on a fixed vertex, this modification allows a fair comparison.

We compared the performance and the quality of the assumptions computed by SImPA and GIST on a set of parity games collected from the SYNTCOMP benchmark suite [1], with a timeout of one hour per game. All the experiments were performed on a computer equipped with Intel(R) Core(TM) i5-10600T CPU @ 2.40GHz and 32 GiB RAM.

We provide all details of the experimental results in the full version [2] and summarize them in Table 1. In addition, Fig. 5 shows a scatter plot, where every instance of the benchmarks is depicted as a point, where the X and the Y coordinates represent the running time for SImPA and GIST (in seconds), respectively. We see that SImPA is computationally much faster than GIST in every instance (all dots lie above the lower red line) – most times by one (above the middle green line) and many times even by two (above the upper orange line) orders of magnitude.

Moreover, in some experiments, GIST fails to compute a sufficient assumption (in the sense of Def. 2), whereas SImPA successfully computes an APA (see the row labeled 'no assumption generated' in Table 1). This is not surprising, as the class of assumptions used by GIST are only unsafe edges and live edges (i.e., singleton live groups) which are not expressive enough to provide sufficient assumptions for all parity games (see Fig. 1(b) for a simple example where there is no sufficient assumption that can be expressed using live edges). Furthermore,

[6] The link provided in the paper is broken, and the authors informed us that the implementation is not available.

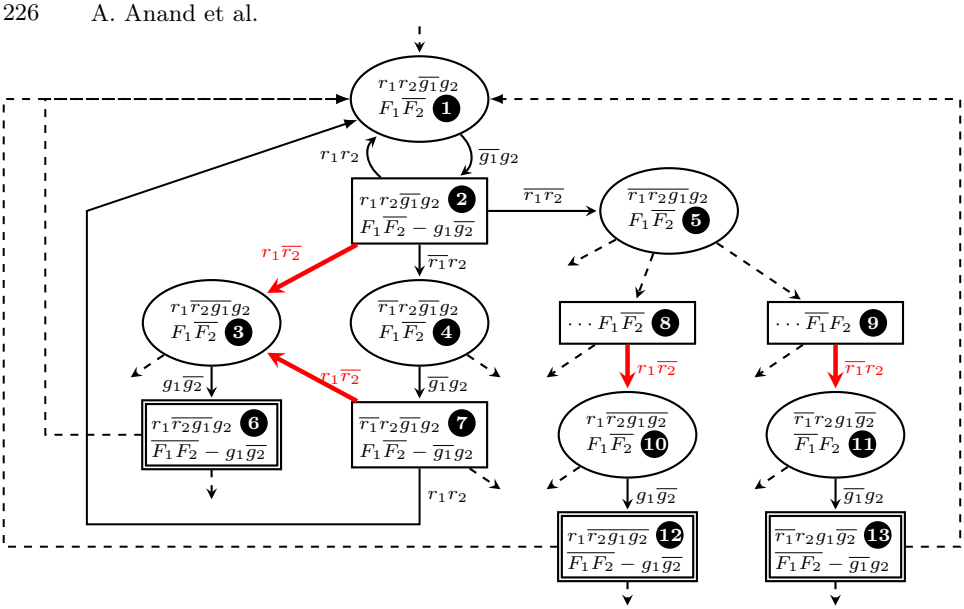

Fig. 6: Illustration of a relevant part of the game graph for the 2-client arbiter. Rectangles and circles represent *Player* 1 and *Player* 0 vertices, respectively. The labels of the *Player* 0 states indicate the current status of the request and grant bits, and in addition, remember if a request is currently *pending* using the atomic propositions F_1, F_2. The double-lined vertices are Büchi vertices, i.e., ones with no pending requests.

we note that in all cases where the assumptions computed by GIST are actually APAs, SIMPA computes the same assumptions orders of magnitudes faster.

5.2 2-Client Arbiter Example

We consider the 2-client arbiter example from the work by Piterman et al. [21], where clients $i \in \{1, 2\}$ (*Player* 1) can request or free a shared resource by setting the input variables r_i to true or false, and the arbiter (*Player* 0) can set the output variables g_i to true or false to grant or withdraw the shared resource to/from client i. The game graph for this example is implicitly given as part of the specification (as this is a GR(1) synthesis problem [21]). The goal of the arbiter is to ensure that always eventually the requests are granted. This can be depicted by a Büchi game, part of which is presented in Fig. 6. It is known that *Player* 0 can not win the game without constraining moves of *Player* 1.

Running SIMPA (took 0.01s) on this example yields two live groups (edges of one live group are indicated by thick red arrows in Fig. 6) that ensures that the play eventually moves to vertices where the *Player* 0 can force a visit to a Büchi vertex. These assumptions are similar to the ones used to restrict the clients' behavior in [21], but are more permissive. Furthermore, running GIST (took 6.44s) yields several live edges (e.g., 2 – 3, 7 – 1), which again is less permissive than ours. It turns out that an APA for this example will unavoidably require live groups — singleton live edges, as computed by GIST, will not suffice. For a detailed discussion, we refer the reader to the full version [2].

References

1. The reactive synthesis competition. http://www.syntcomp.org
2. Anand, A., Mallik, K., Nayak, S.P., Schmuck, A.K.: Computing adequately permissive assumptions for synthesis (2023), https://arxiv.org/abs/2301.07563
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Brenguier, R., Raskin, J.F., Sankur, O.: Assume-admissible synthesis. Acta Informatica (2017)
5. Bronisław Knaster, A.T.: Un théorème sur les fonctions d'ensembles. Annales de la Société polonaise de mathématique **6** (1928)
6. Cavezza, D.G., Alrajeh, D., György, A.: Minimal assumptions refinement for realizable specifications. In: Formal Methods in Software Engineering (2020)
7. Chatterjee, K., Henzinger, T.A.: Assume-guarantee synthesis. In: TACAS (2007)
8. Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Environment assumptions for synthesis. In: CONCUR (2008)
9. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Radhakrishna, A.: Gist: A solver for probabilistic games. In: CAV (2010)
10. Chatterjee, K., Henzinger, T.A., Piterman, N.: Algorithms for büchi games (2008), https://arxiv.org/abs/0805.2620
11. Chatterjee, K., Horn, F., Löding, C.: Obliging games. In: International Conference on Concurrency Theory. pp. 284–296. Springer (2010)
12. Emerson, E., Jutla, C.: Tree automata, μ-calculus and determinacy. In: FOCS (1991)
13. Finkbeiner, B., Metzger, N., Moses, Y.: Information flow guided synthesis. In: Proceedings of 34th International Conference on Computer Aided Verification (CAV 22) (2022)
14. Fisman, D., Kupferman, O., Lustig, Y.: Rational synthesis. In: TACAS (2010)
15. Gaaloul, K., Menghi, C., Nejati, S., Briand, L., Parache, Y.I.: Combining genetic programming and model checking to generate environment assumptions. TSE (2021)
16. Gaaloul, K., Menghi, C., Nejati, S., Briand, L.C., Wolfe, D.: Mining assumptions for software components using machine learning. In: ESEC/FSE (2020)
17. Kozen, D.: Results on the propositional μ-calculus. In: ICALP. Springer (1982)
18. Majumdar, R., Mallik, K., Schmuck, A.K., Zufferey, D.: Assume–guarantee distributed synthesis. IEEE TCAD (2020)
19. Majumdar, R., Piterman, N., Schmuck, A.K.: Environmentally-friendly gr (1) synthesis. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 229–246. Springer (2019)
20. Maoz, S., Ringert, J.O., Shalom, R.: Symbolic repairs for GR(1) specifications. In: ICSE (2019)
21. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation. p. 364-380. VMCAI'06, Springer-Verlag, Berlin, Heidelberg (2006)
22. Schmelter, D., Greenyer, J., Holtmann, J.: Toward learning realizable scenario-based, formal requirements specifications. In: REW (2017)

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.