Contract-Based Distributed Logical Controller Synthesis

Ashwani Anand ⓡ
MPI-SWS
Germany
ashwani@mpi-sws.org

Anne-Kathrin Schmuck ⓡ
MPI-SWS
Germany
akschmuck@mpi-sws.org

Satya Prakash Nayak ⓡ
MPI-SWS
Germany
sanayak@mpi-sws.org

ABSTRACT

We consider the problem of computing distributed logical controllers for two interacting system components via a novel sound and complete contract-based synthesis framework. Based on a discrete abstraction of component interactions as a two-player game over a finite graph and specifications for both components given as ω-regular (e.g. LTL) properties over this graph, we co-synthesize contract and controller candidates locally for each component and propose a *negotiation* mechanism which iteratively refines these candidates until a solution to the given distributed synthesis problem is found. Our framework relies on the recently introduced concept of *permissive templates* which collect an infinite number of controller candidates in a concise data structure. We utilize the efficient computability, adaptability and compositionality of such templates to obtain an efficient, yet sound and complete negotiation framework for contract-based distributed logical control. We showcase the superior performance of our approach by comparing our prototype tool CoSMo to the state-of-the-art tool on a robot motion planning benchmark suite.

ACM Reference Format:
Ashwani Anand ⓡ, Anne-Kathrin Schmuck ⓡ, and Satya Prakash Nayak ⓡ. 2024. Contract-Based Distributed Logical Controller Synthesis. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong SAR, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3641513.3650123

1 INTRODUCTION

Games on graphs provide an effective way to formalize synthesis problems in the context of correct-by-construction cyber-physical systems (CPS) design. A prime example are algorithms to synthesize *control software* that ensures the satisfaction of *logical specifications* under the presence of an external environment, which e.g., causes changed task assignments, transient operating conditions, or unavoidable interactions with other system components. The resulting *logical control software* typically forms a higher layer of the control software stack. The details of the underlying physical dynamics and actuation are then abstracted away into the structure of the game graph utilized for synthesis.

Algorithmically, the outlined control design procedure via games-on-graphs utilizes *reactive synthesis*, a well understood and highly automated design flow originating from the formal methods community rooted in computer science. The strength of reactive synthesis in logical control design is its ability to provide strong correctness guarantees by treating the environment as fully adversarial. While this view is useful if a single controller is designed for a system which needs to obey the specification in an *unknown* environment, it does not excel at synthesizing *distributed* and *interacting* logical control software. While controllers for multiple interacting systems can be obtained using existing centralized synthesis techniques, it requires sharing the specifications of all the systems with a central entity. Towards increasing privacy, a decentralized computation of the controllers is preferred, to avoid centrally handling the specifications and strategy choices of the subsystems. In the multi-system setting, each component acts as the "environment" for the other ones and controllers for components are designed concurrently. Hence, if known a-priory, the control design of one component could take the needs of other components into account and does not need to be treated fully adversarial.

Example 1. As a simple motivating example for a distributed logical control problem with this flavor, consider a fully automated factory producing pens as depicted in Fig. 1. It has a machine which takes raw materials for pens at A_1. When required, it can produce pens with erasers, for which it needs erasers from C_1. For this, it has a robot $\mathcal{R}_1$ that takes the raw materials from B_3 to the production machine at A_1. Hence, the robot $\mathcal{R}_1$ needs to visit A_1 and B_3 infinitely often, i.e. satisfy the LTL objective[1] $\varphi_1 := \Box\Diamond\mathcal{R}_1 : A_1 \wedge \Box\Diamond\mathcal{R}_1 : B_3$, where $\mathcal{R}_i : P$ denotes that $\mathcal{R}_i$ is in the cell P. For delivering the erasers to the machine, it has another robot $\mathcal{R}_2$ that takes raw material from B_3 and feeds the machines via a conveyor belt at C_1 if $\mathcal{R}_1$ feeds the raw material at A_1, i.e, the objective is to satisfy $\varphi_2 := \Box\Diamond\mathcal{R}_1 : A_1 \Rightarrow \Box\Diamond\mathcal{R}_2 : B_3 \wedge \Box\Diamond\mathcal{R}_2 : C_1$. As both robots share the same workspace, their controllers need to ensure the specification despite the movements of the other robot.

The resulting controller synthesis problem can be modeled as a game over a finite graph G – the vertices remember the current position of each robot and its edges model all possible movements between them. Each player in the game models one robot and chooses the moves of that robot along the edges of the graph. Then the specifications for each robot are an LTL objective over G, formally resulting in a *two-objective parity*[2] *game*. While we exclude continuous robot dynamics from the subsequent discussion, we note that non-trivial dynamics can be abstracted into a game graph using well established abstraction techniques (see e.g.,[3, 10, 27] for

ⓡ Authors are ordered randomly and the publicly verifiable record of the randomization is available at www.aeaweb.org. The authors are supported by the DFG projects SCHM 3541/1-1 and 389792660 TRR 248–CPEC. .

HSCC '24, May 14–16, 2024, Hong Kong SAR, China

ACM ISBN 979-8-4007-0522-9/24/05
https://doi.org/10.1145/3641513.3650123

[1] We formally introduce LTL objectives in Section 2.
[2] Parity games are formally introduced in Section 2. Their expressivity is needed to allow for the full class of LTL specifications.

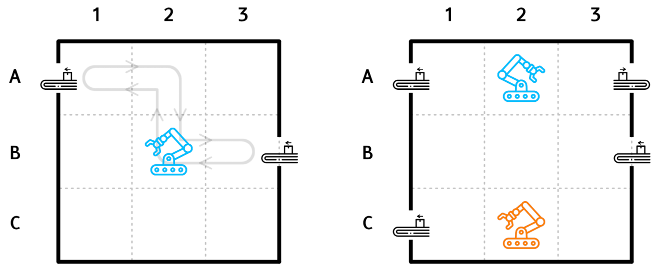


Figure 1: Illustration of a factory with two mobile robots $\mathcal{R}_1$ and $\mathcal{R}_2$, discussed in Example 1-3. Cell Γ_i is located in line $\Gamma \in \{A, B, C\}$ and row $i \in \{1, 2, 3\}$. Walls are indicated by solid lines, conveyor belts are depicted schematically.

an overview) or handled by a well-designed hand-over mechanism between continuous and logical feedback controllers (see e.g. [24]).

Technically, the main contribution of this paper is a new algorithm to solve such *two-objective parity games* arising from distributed logical control problems (as outlined in Example 1) in a *distributed fashion*, i.e., without sharing the local specifications of components with each other, and by performing most computations locally. *Assume-guarantee contracts* have proven to be very useful for such distributed synthesis problems and have been applied to various variants thereof [4, 8, 11–13, 15, 17–20, 22]. The main differences of our work compared to these works is threefold:
(i) We focus on *logical* control design but thereby for the *full class* of ω-regular specifications, making our paper most related to other A/G-based distributed *reactive synthesis* approaches [13, 15, 18, 21].
(ii) We *co-design* both *contracts and controllers*, i.e., we do not assume contracts to be given, as e.g. in [11, 19].
(iii) We develop a *sound and complete* – yet distributed – synthesis framework. Existing tools either perform synthesis centrally, e.g. [12, 15, 18], are not complete, e.g. [4, 17, 21], or cannot handle the full class of LTL specifications [8, 17].

Furthermore, the existing centralized synthesis techniques output a single strategy profile — one strategy per subsystem — which heavily depend on each other. The main advantage of our approach is the computation of decoupled strategy templates. This gives each subsystem the flexibility to independently choose any strategy from a huge class of strategies.

We achieve these new features in A/G-based synthesis by utilizing the concept of *permissive templates* recently introduced by Anand et al. [5, 7]. Such *templates* collect an infinite number of controller candidates in a concise data structure and allow for very efficient computability, adaptability and compositionality as illustrated with the next example.

Example 2. Consider again $\mathcal{R}_1$ in Fig. 1 (left) with specification φ_1 from Example 1. A classical reactive synthesis engine would return a single strategy, e.g. one which keeps cycling along the path[3] $A_1 \to A_2 \to B_2 \to B_3 \to B_2 \to A_2 \to A_1$. However, $\mathcal{R}_1$ does not really need to stick to a single path to fulfill φ_1. It only needs to *always eventually* go from A_1 to A_2 or B_1, from A_2 and B_1 to A_3 or B_2, from A_3 and B_2 to B_3, and so on. These very local liveness properties capture the essence of every correct controller for $\mathcal{R}_1$ and can be summarized as a *strategy template*, which can be extracted from a classical synthesis engine without computational overhead [7]. This controller representation has various advantages.

First, strategy templates are composable. If $\mathcal{R}_1$ is suddenly required to also collect and deposit goods from A_3, we can independently synthesize a strategy template for objective $\varphi_1' = \Box\Diamond A_3$ and both templates can be *composed* by a simple conjunction of all present liveness properties[4]. $\mathcal{R}_1$ can choose a strategy that satisfies both objectives by complying with all template properties.

Second, strategy templates keep all possible strategy choices in-tact and hence allow for a robust control implementation. If, e.g. due to the presence of other robots, A_1 is momentarily blocked, $\mathcal{R}_1$ can keep visiting B_3 and A_3 until A_1 becomes available again[5].

Third, it was recently shown by Nayak et al. [24] that the flexibility of strategy templates allows to realize logical strategy choices by continuous feedback controllers over non-linear dynamics in a provable correct way, without time and space discretizations.

While the above example illustrates the flexibility of strategy templates for a single component, Anand et al. [5, 7] consider games with only one system (and objective). The current paper novelty leverages the easy compositionality and adaptability of the templates for *contract-based distributed synthesis*. Intuitively, strategy templates collect the essence of strategic requirements for one robot in a concise data structure, which can be used to instantiate a contract. As these contracts are locally computed (one for each robot, w.r.t. its objective), they need to be synchronized, which might cause conflicts that need to be resolved. This requires multiple negotiation rounds until a realizing contract is found. The negotiation framework we present in this paper is ensured to *always terminate* and to be sound and complete, i.e., to always provide a realizable contract if the synthesis problem has a solution. The intuition behind our framework is illustrated next.

Example 3. Consider the two-robot scenario in Fig. 1(right) and observe that robot $\mathcal{R}_1$ has no strategy to satisfy its specification without any assumption on $\mathcal{R}_2$'s behavior, e.g. if $\mathcal{R}_2$ always stays in B_3, $\mathcal{R}_1$ can never take raw material from B_3. However, since both robots are built by the factory designers, they can be designed to cooperate "just enough" to fulfill both objectives. We therefore assume that $\mathcal{R}_1$ can "ask" $\mathcal{R}_2$ to always eventually leave B_3, so $\mathcal{R}_1$ can collect the raw goods. Algorithmically, this is done by locally computing both a strategy template [7] for $\mathcal{R}_1$ and an *assumption template* [5] on $\mathcal{R}_2$. The latter contains local requirements on $\mathcal{R}_2$ that need to be satisfied in order for $\mathcal{R}_1$ to fulfill its objective.

When we then switch perspectives and (locally) synthesize a strategy for $\mathcal{R}_2$ for its own objective φ_2, we can force $\mathcal{R}_2$ to obey the assumption template from $\mathcal{R}_1$'s previous computation. This, however, might again put new assumptions on $\mathcal{R}_1$ for the next round. Due to the easy composability of templates, computations in each round are efficient, leading to an overall polynomial algorithm in the size of the game graph. In addition, the algorithm outputs a compatible pair of strategy templates (one for each player), which gives each player maximal freedom for realization (as outlined in Example 2) – any control realization they pick allows also the other

[3] We do not mention the not-so-relevant parts of the strategy for ease of understanding.

[4] Of course, this is not always that easy, but provably so in most practical applications. For details see the extensive evaluation in [7].

[5] Again, see [7] for an extensive case-study of this template feature.

component to fulfill their objective. In this running example, the resulting pair of compatible strategy templates require $\mathcal{R}_1$ to go to A_1 when $\mathcal{R}_2$ is at B_3, and to B_3 only when the access is granted by $\mathcal{R}_2$ (which $\mathcal{R}_2$ will grant, by following its final template), and will also let $\mathcal{R}_2$ go to B_3 always eventually when it arrives.

As these resulting strategy templates only capture the *essence* of the required cooperation, they can also be implemented under partial observation, as long as the required information about the other component are extractable from these observations. Due to page constrains, we omit the formal treatment of this case.

Outline. After presenting required preliminaries in Section 2, we formalize the considered contract-based synthesis problem in Section 3, and instantiate it via permissive templates in Section 4. We then use this instantiation to devise a negotiation algorithm for its solution in Section 5 and prove all features of the algorithm and its output illustrated in Example 3. Finally, Section 5 provides empirical evidence that our negotiation framework possess desirable computational properties by comparing our C++-based prototype tool CoSMo to state-of-the-art solvers on a benchmark suite.

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 $\{n \in \mathbb{N} \mid a \leq n \leq b\}$. Let Σ be a finite alphabet. The notations Σ^* and Σ^ω respectively denote the set of finite and infinite words over 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.

Game Graphs. A *game graph* is a tuple $G = \left(V = V^0 \cup V^1, E\right)$ where (V, E) is a finite, directed graph with *vertices* V and *edges* E, and $V_0, V_1 \subseteq V$ form a partition of V. W.l.o.g. we assume that for every $v \in V$ there exists $v' \in V$ s.t. $(v, v') \in E$. A *play* originating at a vertex v_0 is an infinite sequence of vertices $\rho = v_0 v_1 \ldots \in V^\omega$.

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. Then the set of desired infinite plays is given by the ω-regular language $\mathcal{L}(G, \Phi) \subseteq V^\omega$. When G is clear from the context, we simply write $\mathcal{L}(\Phi)$. The standard definitions of ω-regular languages and LTL are omitted for brevity and can be found in standard textbooks [9].

A *parity objective* $\Phi = Parity(\mathbb{P})$ is given by the LTL formula

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

with *priority set* $P^j = \{v : \mathbb{P}(v) = j\}$ for $0 \leq j \leq d$ of vertices for some *priority function* $\mathbb{P} : V \to [0; d]$ that assigns each vertex a *priority*. $\mathcal{L}(Parity(\mathbb{P}))$ contains all plays ρ for which the highest priority appearing infinitely often along ρ is even. We note that every game with an arbitrary ω-regular set of desired plays can be reduced to a parity game (possibly with a larger set of vertices) by standard methods [9].

Games. A *two-player (turn-based) game* is a tuple $\mathcal{G} = (G, \Phi)$, where G is a game graph, and Φ is the *winning condition* over G. A *two-player (turn-based) two-objective game* is a triple $\mathcal{G} = (G, \Phi_0, \Phi_1)$, where G is a game graph, and Φ_0 and Φ_1 are *winning conditions* over G, respectively, for Player 0 and Player 1. We call a $\mathcal{G}$ a parity game if all involved winning conditions are parity objectives.

Strategies. A strategy of Player i (for $i \in \{0, 1\}$) is a function $\pi_i : V^* V_i \to V$ such that for every $\rho v \in V^* V_i$ holds that $\pi_i(\rho v) \in E(v)$. A *strategy profile* (π_0, π_1) is a pair where π_i is a strategy for Player i. Given a strategy π_i, we say that the play $\rho = v_0 v_1 \ldots$ is *compliant* with π_i if $v_{k-1} \in V_i$ implies $v_k = \pi_i(v_0 \ldots v_{k-1})$ for all k. We refer to a play compliant with π_i and a play compliant with a strategy profile (π_0,π_1) as a π_i*-play* and a $\pi_0\pi_1$*-play*, respectively.

Winning. Given a game $\mathcal{G} = (G, \Phi)$, a play ρ in $\mathcal{G}$ is *winning* if it satisfies[6] Φ, i.e., $\rho \in \mathcal{L}(\Phi)$. A strategy π_i for Player i is *winning from a vertex* $v \in V$ if all π_i-plays from v are winning. A vertex $v \in V$ is *winning for Player* i, if there exists a Player i winning strategy π_i from v. We collect all winning vertices of Player i in the *Player* i *winning region* $\langle\langle i \rangle\rangle \Phi \subseteq V$. We say a Player i strategy is *winning* for Player i if it is winning from every vertex in $\langle\langle i \rangle\rangle \Phi$.

Furthermore, given a game (G, Φ), we say a strategy profile (π_0, π_1) is *winning from a vertex* $v \in V$ if the $\pi_0\pi_1$-play from v is winning. We say a vertex $v \in V$ is *cooperatively winning*, if there exists a winning strategy profile (π_0, π_1) from v. We collect all such vertices in the *cooperative winning region* $\langle\langle 0, 1 \rangle\rangle \Phi \subseteq V$. We say a strategy profile is *winning* if it is *winning* from every vertex in $\langle\langle 0, 1 \rangle\rangle \Phi$. Winning strategies and cooperative winning region for a two-objective game (G, Φ_0, Φ_1) are defined analogously.

3 CONTRACT-BASED SYNTHESIS

Towards a formalization of our proposed negotiation framework for distributed synthesis this section introduces the notion of assume-guarantee contracts (Section 3.1) that we build upon, the notion of iRmaC-specifications (Section 3.2) that describes our main goal, and formally states the synthesis problem we solve in this paper (Section 3.3).

3.1 Assume-Guarantee Contracts

Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$ we define an *assume-guarantee contract* over $\mathcal{G}$ — a *contract* for short — as a tuple $\mathsf{C} := ((\mathsf{A}_0, \mathsf{G}_0), (\mathsf{A}_1, \mathsf{G}_1))$ where A_i and G_i are LTL specifications over the graph G called the *assumption* and the *guarantee* for player i, respectively. It is well known that such contracts provide a *certified interface* between both players, if they are

(i) *compatible*, i.e.,

$$\mathcal{L}(\mathsf{G}_i) \subseteq \mathcal{L}(\mathsf{A}_{1-i}), \text{ and} \quad (2)$$

(ii) *realizable* by both players from at least one vertex, i.e.,

$$\exists v \in V . \forall i \in \{0, 1\} . v \in \langle\langle i \rangle\rangle(\mathsf{A}_i \Rightarrow (\mathsf{G}_i \wedge \Phi_i)). \quad (3)$$

Unfortunately, it is also well known that for the full class of ω-regular contracts, conditions (2)-(3) are not strong enough to provide a sound (and complete) proof rule for *verification*, let alone the harder problem of *synthesis*. In verification, one typically resorts to strengthening the contracts with less expressive properties [2, 23, 25, 26]. This approach was also followed by [21] for synthesis, requiring contracts to be safety formulas. This, however, always

[6] Throughout the paper, we use the terms "winning for objective Φ" and "satisfying Φ" interchangeably.

results in an unavoidable conservatism, resulting in incompleteness of the proposed approaches.

Within this paper, we take a novel approach to this problem which does not restrict the expressiveness of the formulas in $(\mathsf{A}_i, \mathsf{G}_i)$ but rather liberally changes the considered local specification in (3) to one which is "well-behaved" for contract-based distributed synthesis. We then show, that this liberty does still result in a sound and complete distributed synthesis technique by developing an algorithm to compute such "well-behaved" specifications whenever the original two-objective game has a cooperative solution. Before formalizing this problem statement in Section 3.3 we first define such "well-behaved" specifications, called iRmaC– independently Realizable and maximally Cooperative.

3.2 iRmaC-Specifications

We begin by formalizing a new "well-behaved" local specification for contract realizability.

Definition 4 (iR-Contracts). A contract $\mathsf{C} := ((\mathsf{A}_0, \mathsf{G}_0), (\mathsf{A}_1, \mathsf{G}_1))$ over a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$ is called *independently realizable* (iR) from a vertex v if (2) holds and for all $i \in \{0, 1\}$

$$v \in \langle\langle i \rangle\rangle \Phi_i^\bullet \text{ with } \Phi_i^\bullet := \mathsf{G}_i \wedge (\mathsf{A}_i \Rightarrow \Phi_i), \tag{4}$$

where $\Phi_i^\bullet$ is called a *contracted local specification*.

Intuitively, (4) requires the guarantees to be realizable by Player i without the "help" of player Player $1-i$, i.e., unconditioned from the assumption, which is in contrast to (3). It is therefore not surprising that iR-Contracts allow to solve the local contracted games $(G, \Phi_i^\bullet)$ fully independently (and in a zero-sum[7] fashion) while still ensuring that the resulting strategy profile solves the original game $\mathcal{G}$.

Proposition 5. Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$ with iR-contract $\mathsf{C} := ((\mathsf{A}_0, \mathsf{G}_0), (\mathsf{A}_1, \mathsf{G}_1))$ realizable from a vertex v, and contracted local specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$, let π_i be a winning strategy in the (zero-sum) game $(G, \Phi_i^\bullet)$.

Then the tuple (π_0, π_1) is a winning strategy profile for $\mathcal{G}$ from v.

Proof. As $v \in \langle\langle i \rangle\rangle \Phi_i^\bullet$, π_i is winning from v for game $(G, \Phi_i^\bullet)$, every π_i-play from v satisfies $\Phi_i^\bullet = \mathsf{G}_i \wedge (\mathsf{A}_i \Rightarrow \Phi_i)$. Therefore, every (π_0, π_1)-play from v satisfies both $\Phi_0^\bullet$ and $\Phi_1^\bullet$. Now, let us show that $\mathcal{L}(\Phi_0^\bullet \cap \Phi_1^\bullet) \subseteq \mathcal{L}(\Phi_0 \wedge \Phi_1)$. Using the definition of contracted local specifications and by (2), we have $\mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet) = \mathcal{L}(\mathsf{G}_0 \wedge (\mathsf{A}_0 \Rightarrow \Phi_0)) \cap \mathcal{L}(\mathsf{G}_1 \wedge (\mathsf{A}_1 \Rightarrow \Phi_1)) \subseteq_{(2)} \mathcal{L}(\mathsf{A}_1 \wedge (\mathsf{A}_0 \Rightarrow \Phi_0)) \cap \mathcal{L}(\mathsf{A}_0 \wedge (\mathsf{A}_1 \Rightarrow \Phi_1)) = \mathcal{L}(\mathsf{A}_0) \cap \mathcal{L}(\mathsf{A}_1) \cap \mathcal{L}(\mathsf{A}_0 \Rightarrow \Phi_0) \cap \mathcal{L}(\mathsf{A}_1 \Rightarrow \Phi_1) = \mathcal{L}(\Phi_0) \cap \mathcal{L}(\Phi_1) = \mathcal{L}(\Phi_0 \wedge \Phi_1)$. Therefore, every (π_0, π_1)-play from v satisfies $\Phi_0 \wedge \Phi_1$. Hence, (π_0, π_1) is a winning strategy profile for $\mathcal{G}$ from v. □

By the way they are defined, iR-Contracts can be used to simply encode a single winning strategy profile from a vertex, which essentially degrades contract-based synthesis to solving a single cooperative game with specification $\Phi_0 \cup \Phi_1$. The true potential of iR-Contracts is only reveled if they are reduced to the "essential cooperation" between both players. Then the local contracted specifications $\Phi_i^\bullet$ will give each player as much freedom as possible to choose its local strategy. This is formalized next.

[7]A zero-sum game is a two-player game where the opponent has the negated specification of the protagonist, i.e., $(G, \Phi, \neg\Phi)$, i.e., the opponent acts fully adversarially. As defined in Section 2, this (standard version of) games is denoted by tuples (G, Φ).

Definition 6 (iRmaC-Specifications). Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$, a pair of specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$ is said to be *independently realizable* and *maximally cooperative* (iRmaC) if

$$\mathcal{L}(\Phi_0 \wedge \Phi_1) = \mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet), \text{ and} \tag{5a}$$

$$\langle\langle 0, 1 \rangle\rangle (\Phi_0 \wedge \Phi_1) = \langle\langle 0 \rangle\rangle \Phi_0^\bullet \cap \langle\langle 1 \rangle\rangle \Phi_1^\bullet. \tag{5b}$$

Here (5a) ensures that the contracted local games $(G, \Phi_i^\bullet)$ do not eliminate any cooperative winning play allowed by the original specifications, while (5b) ensures that the combination of local winning regions does not restrict the cooperative winning region. These properties of iRmaC-specifications now allow each player to extract a strategy π_i *locally and fully independently* by solving the (zero-sum) game $(G, \Phi_i^\bullet)$. Then it is guaranteed that the resulting (independently chosen) strategy profile (π_0, π_1) is winning in $\mathcal{G} = (G, \Phi_0, \Phi_1)$. This is formalized next.

Proposition 7. Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$ with iRmaC specifications-$(\Phi_0^\bullet, \Phi_1^\bullet)$, the following are equivalent:

(i) there exists a winning strategy profile from v in (G, Φ_0, Φ_1),
(ii) for each $i \in \{0, 1\}$, there exists a Player i winning strategy from v in $(G, \Phi_i^\bullet)$.

Proof. (Item (i) ⇒Item (ii)) If there exists a winning strategy profile from v for the game (G, Φ_0, Φ_1), then $v \in \langle\langle 0, 1 \rangle\rangle (\Phi_0 \wedge \Phi_1)$. Then, by (5b), $v \in \langle\langle i \rangle\rangle \Phi_i^\bullet$ for each $i \in \{0, 1\}$. Hence, there exists a Player i winning strategy from v in $(G, \Phi_i^\bullet)$ for each $i \in \{0, 1\}$. (Item (i)⇐Item (ii)) Similarly, if there exists a Player i winning strategy from v for the game $(G, \Phi_i^\bullet)$ for each $i \in \{0, 1\}$, then $v \in \langle\langle i \rangle\rangle \Phi_i^\bullet$ for each $i \in \{0, 1\}$. Then, by (5b), $v \in \langle\langle 0, 1 \rangle\rangle (\Phi_0 \wedge \Phi_1)$, and hence, there exists a winning strategy profile from v for the game (G, Φ_0, Φ_1). □

With this, we argue that iRmaC-specifications indeed provide a *maximally cooperative* contract for distributed synthesis which allows to fully decentralize remaining strategy choices.

3.3 Problem Statement and Outline

Based on the desirable properties of iRmaC-specifications outlined before, the main contribution of this paper is an algorithm to compute iRmaC-specifications for *two-objective parity games*, which are a canonical representation of two-player games with an LTL objective for each player.

Problem 8. Given a two-objective parity game $\mathcal{G} = (G, \Phi_0, \Phi_1)$, compute iRmaC-specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$.

In particular, we provide an algorithm which *always* outputs an iRmaC-specification s.t. the latter only results in an empty cooperative winning region (via (5b)) *if and only if* $\mathcal{G}$ is not cooperatively solvable. Thereby, our approach constitutes a *sound and complete* approach to distributed logical controller synthesis. All existing solutions to this problem, i.e., [18, 21], only provide a sound approach. In addition, as outlined before, the computed iRmaC-specifications then allow to choose winning strategies (i.e., controllers) in a fully decentralized manner (due to Proposition 7).

Our algorithm for solving Problem 8 is introduced in Section 4 and Section 5. Conceptually, this algorithm builds upon the recently introduced formalism of *permissive templates* by Anand et al. [5, 7] and utilizes their efficient computability, adaptability and permissiveness to solve Problem 8. Interestingly, this approach does not only allow us to solve Problem 8 but also allows resulting local strategies to be easily adaptable to new local objectives and unforeseen circumstances, as illustrated in the motivating example of Section 1. We showcase the computational efficiency and the extra features of our approach by experiments with a prototype implementation on a set of control-inspired benchmarks in Section 6.

4 CONTRACTS AS TEMPLATES

This section shows how *templates* can be used to solve Problem 8 and starts with an illustrative example to convey some intuition.

Example 9. In order to appreciate the simplicity, adaptability and compositionality of templates consider the two-objective game in Fig. 2. The winning condition Φ_0 for Player 0 requires vertex c to be seen infinitely often. Intuitively, every winning strategy for Player 0 w.r.t. Φ_0 needs to eventually take the edge e_{ac} if it sees vertex a infinitely often. Furthermore, Player 0 can only win from vertex b with the help of Player 1. In particular, Player 1 needs to ensure that whenever vertex b is seen infinitely often it takes edge e_{bc} infinitely often. These two conditions can be concisely formulated via the strategy template $\Pi_0 = \Lambda_{\text{LIVE}}(\{e_{ac}\})$ and an assumption template $\Psi_0 = \Lambda_{\text{LIVE}}(\{e_{bd}\})$, both given by what we call a live-edge template – if the source is seen infinitely often, the given edge has to be taken infinitely often. It is easy to see that every Player 0 strategy that satisfies Π_0 is winning for Φ_0 under the assumption that Player 1 chooses a strategy that satisfies Ψ_0.

Now, consider the winning condition Φ_1 for Player 1 which requires the play to eventually stay in region $\{a, c, d\}$. This induces assumption Ψ_1 on Player 0 and strategy template Π_1 for Player 1 given in Fig. 2 (right). Both are *co-liveness* templates – the corresponding edge can only be taken *finitely* often. This ensures that all edges that lead to the region $\{a, c, d\}$ are taken only finitely often.

The tuples of strategy and assumption templates (Ψ_i, Π_i) we have constructed for both players in the above example will be called *contracted strategy-masks*, CSM for short. If the players now share the assumptions from their local CSMs, it is easy to see that in the above example both players can ensure the assumptions made by other player in addition to their own strategy templates, i.e., each Player i can realize $\Psi_{1-i} \wedge \Pi_i$ from all vertices. In this case, we call the CSMs (Ψ_i, Π_i) *compatible*. In such situations, the new specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$ with $\Phi_i^\bullet = \Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i)$ are directly computable from the given CSMs and indeed form an iRmaC-contract.

Unfortunately, locally computed CSMs are not always compatible. To see this, consider the slightly modified winning condition Φ_1' for Player 1 that induces strategy template Π_1' for Player 1. This template requires the edge e_{bd} to be taken only *finitely* often. Now, Player 1 cannot realize both Ψ_0 and Π_1' as the conditions given by both templates for edge e_{bd} are *conflicting* – the same edge cannot be taken infinitely often *and* finitely often. In this case one more round of negotiation is needed to ensure that both players eventually avoid vertex d by modifying the objectives to $\Phi_i' = \Phi_i \wedge \Diamond\Box\neg d$. This will give us a new pair of CSMs that are indeed compatible, and a new pair of objectives $(\Phi_0^\bullet, \Phi_1^\bullet)$ that are now again an iRmaC specification.

In the following we formalize the notion of templates (Section 4.1) and CSMs (Section 4.2) and show that, if compatible, they indeed provide iRmaC-specifications (Section 4.3). We further show how to compute CSMs for each player (Section 4.4). The outlined negotiation for compatibility is then discussed in Section 5.

4.1 Permissive Templates

This section recalls the concept of *templates* from [5, 7]. 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 i strategy π_i *follows* Λ if every π_i-play belongs to $\mathcal{L}(\Lambda)$, i.e., strategy π_i is winning from all vertices in the game (G, Λ). The exposition in this section follows the presentation in [5] 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 $\Lambda_{\text{UNSAFE}}(S) := \Box \wedge_{e\in S} \neg e$, 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 $\Lambda_{\text{LIVE}}(H_\ell) := \bigwedge_{i\geq 0} \Box\Diamond src(H_i) \Rightarrow \Box\Diamond H_i$. 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 $\Lambda_{\text{COND}}(\mathcal{H}) := \bigwedge_{(R,H_\ell)\in\mathcal{H}} (\Box\Diamond R \Rightarrow \Lambda_{\text{LIVE}}(H_\ell))$. 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 $\Lambda_{\text{LIVE}}(H_\ell)$.

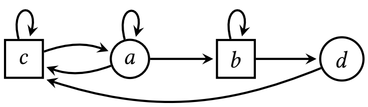
Co-liveness Templates. Given a set of *co-live* edges D a co-live template is defined as $\Lambda_{\text{COLIVE}}(D) := \bigwedge_{e\in D} \Diamond\Box\neg e$. A co-liveness template requires that edges in D are only taken finitely often.

Composed Templates. In the following, 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 $\Lambda \triangleleft (S, D, \mathcal{H})$. Similarly, $\Lambda \triangleleft (S, D, H_\ell)$ 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 $\Lambda \triangleleft (S, D, \mathcal{H})$ and $\Lambda' \triangleleft (S', D', \mathcal{H}')$ is equivalent to the template $(\Lambda \wedge \Lambda') \triangleleft (S \cup S', D \cup D', \mathcal{H} \cup \mathcal{H}')$ by the definition of conjunction of LTL formulas.

4.2 Contracted Strategy-Masks (CSMs)

Towards our goal of formalizing iRmaC-specifications via templates, this section defines *contracted strategy-masks* which contain two templates Ψ_i and Π_i, representing a set of Player $1-i$- and Player i-strategies respectively, which can be interpreted as the assumption Ψ_i on player Player $1-i$ under which Player i can win the local game (G, Φ_i) with any strategy from Π_i.

Towards this goal, we first observe that every template in Section 4.1 is defined via a set of edges that a following strategy needs to handle in a particular way. Intuitively, we can therefore "split" each template into a part restricting strategy choices for Player 0 (by only considering edges originating from V_0) and a part restricting



$$\Phi_0 = \Box\Diamond\{c\} \quad \Rightarrow \Psi_0 = \Lambda_{\text{LIVE}}(\{e_{bd}\}), \quad \Pi_0 = \Lambda_{\text{LIVE}}(\{e_{ac}\})$$
$$\Phi_1 = \Diamond\Box\{a,c,d\} \quad \Rightarrow \Psi_1 = \Lambda_{\text{COLIVE}}(e_{ab}), \quad \Pi_1 = \Lambda_{\text{COLIVE}}(e_{bb})$$
$$\Phi_1' = \Diamond\Box\{a,b,c\} \quad \Rightarrow \Psi_1' = \texttt{true}\ , \quad \Pi_1' = \Lambda_{\text{COLIVE}}(e_{bd})$$

Figure 2: A two-player game graph discussed in Example 9 with Player 1 (squares) and Player 0 (circles) vertices, different winning conditions Φ_i, and corresponding winning assumption templates Ψ_i and strategy templates Π_i for Player i.

strategy choices for Player 1 (by only considering edges originating from V_1), which then allows us to define CSM.

Definition 10. Given a game graph $G = (V, E)$, a template $\Lambda \lhd (S, D, \mathcal{H})$ over G is an *assumption template* (resp. a *strategy template*) for player i if for all edges $e \in S \cup D \cup \overline{H}$ holds that $src(e) \in V_{1-i}$ (respectively $src(e) \in V_i$) where $\overline{H} := \bigcup\{H \in H_\ell \mid (\cdot, H_\ell) \in \mathcal{H}\}$.

Definition 11. Given a game (G, Φ_i), a *contracted strategy-mask* (CSM) for player i is a tuple (Ψ_i, Π_i), such that $\Psi_i \lhd (S_i^s, D_i^s, \mathcal{H}_i^s)$ and $\Pi_i \lhd (S_i^a, D_i^a, \mathcal{H}_i^a)$ are assumption and strategy templates for player i, respectively.

We next formalize the intuition that CSMs collect winning strategies for Player i under assumptions on Player $1-i$.

Definition 12. A CSM (Ψ_i, Π_i) is *winning* for Player i in (G, Φ_i) *from vertex* v if for every Player i strategy π_i following Π_i and every Player $1-i$ strategy π_{1-i} following Ψ_i the $\pi_0\pi_1$-play originating from v is winning. Moreover, we say a CSM (Ψ_i, Π_i) is *winning* for Player i in (G, Φ_i) if it is winning from every vertex in $\langle\langle 0, 1\rangle\rangle\Phi_i$.

We denote by $\langle\langle i\rangle\rangle(\Psi_i, \Pi_i)$ the set of vertices from which (Ψ_i, Π_i) is winning for Player i in (G, Φ_i). Due to localness of our templates, the next remark follows.

Remark 13. If a CSM (Ψ_i, Π_i) is winning for Player i in (G, Φ_i) from vertex v, then every Player i strategy π_i following Π_i is winning for Player i in the game $(G, \Psi_i \Rightarrow \Phi_i)$.

4.3 Representing Contracts via CSM

The previous subsection has formalized the concept of a CSM for a local (zero-sum) game (G, Φ_i) . This section now shows how under which conditions the combination of two CSMs (Ψ_0, Π_0) and (Ψ_1, Π_1) (one for each player) allows to construct a contract

$$\mathsf{C} := ((\Psi_0, \Psi_1), (\Psi_1, \Psi_0)), \tag{6}$$

(i.e, setting $\mathsf{A}_i := \Psi_i$ and $\mathsf{G}_i := \Psi_{1-i}$), which induces iRmaC-specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$ as in (4).

The first condition we need is compatibility.

Definition 14 (Compatible CSMs)**.** Two CSMs, (Ψ_0, Π_0) for Player 0 and (Ψ_1, Π_1) for Player 1, are said to be *compatible*, if for each $i \in \{0, 1\}$, there exists a Player i strategy π_i that follows $\Pi_i \wedge \Psi_{1-i}$.

Intuitively, as Ψ_{1-i} is the assumption on Player i and Π_i represents the template that Player i will follow, we need to find a strategy that follows both templates. Before going further, let us first show a simple result that follows from Definition 14.

Proposition 15. Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$, let (Ψ_0, Π_0) and (Ψ_1, Π_1) be two compatible CSMs s.t. (Ψ_i, Π_i) is winning from a vertex v for Player i in (G, Φ_i). Then the contract C as in (6) is an iR-contract realizable from v.

Proof. We need to show that $v \in \langle\langle i\rangle\rangle(\Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i))$ for each $i = 0, 1$. Firstly, as the CSMs are compatible, for each i, there exists a Player i strategy π_i that follows $\Pi_i \wedge \Psi_{1-i}$. Hence, every π_i-play satisfies both Ψ_{1-i}. Secondly, as CSM (Ψ_i, Π_i) is winning from v for Player i in game (G, Φ_i), by Remark 13, every π_i-play from v satisfies $\Psi_i \Rightarrow \Phi_i$. Therefore, every π_i-play from v satisfies $\Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i)$, and hence, $v \in \langle\langle i\rangle\rangle(\Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i))$. □

To ensure that two compatible CSMs as in Proposition 15 are not only an iR-contract but also provide iRmaC-specifications, we utilize the main result from [5] which showed that assumption templates can be computed in an *adequately permissive*[8] way over a given parity game. This notion is translated to CSMs next.

Definition 16. Given a game (G, Φ_i) and a CSM (Ψ_i, Π_i) for Player i, we call this CSM *adequately permissive* for (G, Φ_i) if it is

(i) *sufficient*: $\langle\langle i\rangle\rangle(\Psi_i, \Pi_i) \supseteq \langle\langle 0, 1\rangle\rangle\Phi_i$,
(ii) *implementable*: $\langle\langle 1-i\rangle\rangle\Psi_i = V$ and $\langle\langle i\rangle\rangle\Pi_i = V$
(iii) *permissive*: $\mathcal{L}(\Phi_i) \subseteq \mathcal{L}(\Psi_i)$.

Note that the sufficiency condition makes the CSM winning as formalized in the next remark.

Remark 17. If a CSM (Ψ_i, Π_i) for Player i in a game (G, Φ_i) is sufficient, then it is winning for Player i in (G, Φ_i).

With this, we are ready to prove the main theorem of this section, which shows that synthesis of iRmaC-specifications reduces to finding adequately permissive CSMs which are compatible.

Theorem 18. Given a two-objective game $\mathcal{G} = (G, \Phi_0, \Phi_1)$, let (Ψ_0, Π_0) and (Ψ_1, Π_1) be two *compatible* CSMs s.t. (Ψ_i, Π_i) is *adequately permissive* for Player i in (G, Φ_i). Then $(\Phi_0^\bullet, \Phi_1^\bullet)$ with $\Phi_i^\bullet = \Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i)$ are iRmaC-specifications.

Proof. We need to show that the pair $(\Phi_0^\bullet, \Phi_1^\bullet)$ satisfies (5a) and (5b). The proof for (5a) is completely set theoretic:
$(\subseteq)$ For each $i \in \{0, 1\}$, it holds that $\mathcal{L}(\Phi_i) \subseteq_{((\text{iii}))} \mathcal{L}(\Phi_i) \cap \mathcal{L}(\Psi_i) \subseteq \mathcal{L}(\Psi_i \Rightarrow \Phi_i) \cap \mathcal{L}(\Psi_i)$. With this we have, $\mathcal{L}(\Phi_0 \wedge \Phi_1) \subseteq \mathcal{L}(\Phi_0) \cap \mathcal{L}(\Phi_1) \subseteq \mathcal{L}(\Psi_0 \Rightarrow \Phi_0) \cap \mathcal{L}(\Psi_0) \cap \mathcal{L}(\Psi_1 \Rightarrow \Phi_1) \cap \mathcal{L}(\Psi_1)$ which simplifies to $\mathcal{L}(\Phi_0^\bullet) \cap \mathcal{L}(\Phi_1^\bullet) = \mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet)$.
$(\supseteq)$ For each $i \in \{0, 1\}$, it holds that $\mathcal{L}(\Phi_i^\bullet)$ is equivalent to $\mathcal{L}(\Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i)) = \mathcal{L}(\Psi_{1-i} \wedge (\neg\Psi_i \vee \Phi_i)) = \mathcal{L}((\Psi_{1-i} \wedge \neg\Psi_i) \vee (\Psi_{1-i} \wedge \Phi_i))$ which simplifies to $\mathcal{L}(\Psi_{1-i} \wedge \neg\Psi_i) \cup \mathcal{L}(\Psi_{1-i} \wedge \Phi_i)$. Then we have that $\mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet) = \mathcal{L}(\Phi_0^\bullet) \cap \mathcal{L}(\Phi_1^\bullet)$ reduces to $(\mathcal{L}(\Psi_1 \wedge \neg\Psi_0) \cup \mathcal{L}(\Psi_1 \wedge \Phi_0)) \cap (\mathcal{L}(\Psi_0 \wedge \neg\Psi_1) \cup \mathcal{L}(\Psi_0 \wedge \Phi_1))$ which simplifies to $\mathcal{L}(\Psi_1 \wedge \Phi_0) \cap \mathcal{L}(\Psi_0 \wedge \Phi_1) \subseteq \mathcal{L}(\Phi_0 \wedge \Phi_1)$.

Next, we show that one side of (5b) follows from (5a), whereas the other side follows from Proposition 15: $(\supseteq)$ If $v \in \langle\langle 0\rangle\rangle\Phi_0^\bullet \cap \langle\langle 1\rangle\rangle\Phi_1^\bullet$, then for each $i \in \{0, 1\}$, there exists a strategy π_i for Player i such that every π_i-play from v belongs to $\mathcal{L}(\Phi_i^\bullet)$. Hence, every $\pi_0\pi_1$-play from v belongs to $\mathcal{L}(\Phi_0^\bullet) \cap \mathcal{L}(\Phi_1^\bullet) = \mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet) =_{(5a)} \mathcal{L}(\Phi_0 \wedge \Phi_1)$. Therefore, $v \in \langle\langle 0, 1\rangle\rangle(\Phi_0 \wedge \Phi_1)$. $(\subseteq)$ If $v \in \langle\langle 0, 1\rangle\rangle(\Phi_0 \wedge$

[8] We refer to [5] for an elaborate discussion of conditions (i)-(iii) in Definition 16.

Algorithm 1 NEGOTIATE(G, Φ_0, Φ_1)

Input: $G = (V, E)$, $\Phi_0 = Parity(\mathbb{P}_0)$, $\Phi_1 = Parity(\mathbb{P}_1)$,
Output: modified specifications (Φ_0, Φ_1); CSMs (Ψ_0, Π_0), (Ψ_1, Π_1)

1: $\mathcal{W}_i, C_i, \Pi_i, \Psi_i \leftarrow$ PARITYTEMP(G, Φ_i, i), $\forall i \in \{0, 1\}$
2: **if** CHECKTEMPLATE$(G, \Psi_{1-i} \wedge \Pi_i) = \texttt{true}$, $\forall i \in \{0, 1\}$ **then**
3:     **return** $(\Phi_0, \Phi_1), (\Psi_0, \Pi_0), (\Psi_1, \Pi_1)$
4: **else**
5:     $\Phi'_i \leftarrow \Phi_i \wedge \Box(\mathcal{W}_0 \cap \mathcal{W}_1) \wedge \Diamond\Box\neg(C_0 \cup C_1)$, $\forall\, i \in \{0, 1\}$
6:     **return** NEGOTIATE(G, Φ'_0, Φ'_1)

$\Phi_1) \subseteq \langle\langle 0, 1\rangle\rangle\Phi_i$, then by Item (i), $v \in \langle\langle i\rangle\rangle(\Pi_i, \Psi_i)$. Hence, for each i, CSM (Π_i, Ψ_i) is winning for Player i from v. As the CSMs are also compatible, by Proposition 15, the contract $\mathsf{C} = (\Psi_0, \Psi_1)$ is an iR-contract realizable from v. Hence, by definition, $v \in \langle\langle i\rangle\rangle\Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i) = \langle\langle i\rangle\rangle\Phi^{\bullet}_i$. Therefore, $v \in \langle\langle 0\rangle\rangle\Phi^{\bullet}_0 \cap \langle\langle 1\rangle\rangle\Phi^{\bullet}_1$. □

4.4 Computing Adequately Permissive CSMs

As stated before, Theorem 18 shows that a solution to Problem 8 reduces to finding *adequately permissive* CSMs which are *compatible*. Due to the close connection between *adequately permissive* CSMs and *adequately permissive assumption templates* from [5], it turns out that the computation of *adequately permissive* CSMs can be done in very close analogy to the computation of adequately permissive assumption templates for parity games from [5]. In particular, we inherent (i) the observation that conjunctions of safety, co-live and conditional live-group templates are rich enough to express adequately permissive CSMs, and (ii) the existence of a polynomial time (i.e., very efficient) algorithm for their construction.

Theorem 19. Given a game graph $G = (V = V_0 \uplus V_1, E)$ with parity objective Φ_i, an adequately permissive CSM $(\Psi_i \lhd (S^s_i, D^s_i, \mathcal{H}^s_i), \Pi_i \lhd (S^a_i, D^a_i, \mathcal{H}^a_i))$ for player i in (G, Φ_i) can be computed in time $O(n^4)$, where $n = |V|$. We call the respective procedure for this computation PARITYTEMP(G, Φ_i).

For completeness, we give the full algorithm for PARITYTEMP, along with its simplified (and more efficient) versions for Safety (UNSAFE with $O(m)$, $m = |E|$), Büchi (BÜCHITEMP with $O(m)$, $m = |E|$) and co-Büchi games (COBÜCHITEMP with $O(m)$, $m = |E|$) along with additional intuition and all correctness proofs in the extended version of the paper [6]. This exposition is given in very close analogy to [5].

With Theorem 19 in place, the main algorithmic problem for solving Problem 8 is to ensure that computed CSMs are *compatible*. This is done via a negotiation algorithm, as already illustrated in the last paragraph of Example 9, formalized next.

5 NEGOTIATION FOR COMPATIBLE CSMS

This section contains the main contribution of this paper w.r.t. the algorithmic solution of Problem 8. That is, we give an algorithm to compute *adequately permissive* and *compatible* CSMs in a mostly distributed fashion.

Our algorithm, called NEGOTIATE, is depicted schematically in Fig. 3 and given formally in Algorithm 1. It uses PARITYTEMP to compute adequately permissive CSMs (Ψ_i, Π_i) for each Player i in its corresponding game (G, Φ_i) locally (Line 1 in Algorithm 1). These

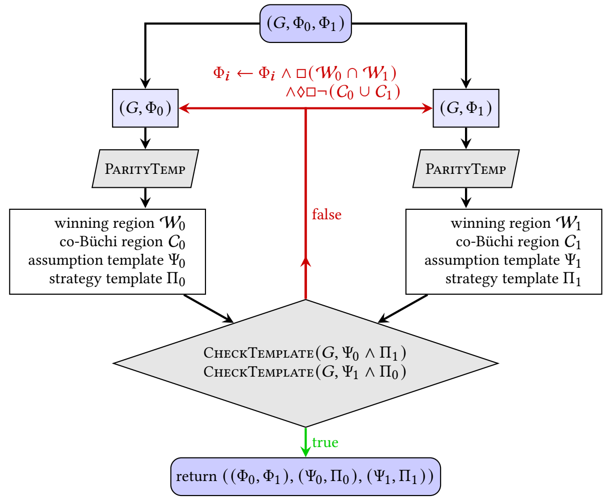


Figure 3: Flowchart illustration of NEGOTIATE (Algorithm 1).

CSMs are then checked for compatibility (as in Definition 14) via the function CHECKTEMPLATE defined in Section 5.1. If CSMs are compatible, they define iRmaC-specifications (via Theorem 18) and hence, Problem 8 is solved and NEGOTIATE terminates.

If they are not compatible, existing conflicts need to be resolved as formalized in Section 5.2. The required strengthening of both CSMs is again done locally by solving games with modified specifications (red arrow looping back in Fig. 3) again via PARITYTEMP.

As the resulting new CSMs might again be conflicting, this strengthening process repeats iteratively. We prove in Section 5.3 that there are always only a finite number of negotiation rounds.

5.1 Checking Compatibility Efficiently

This section discusses how the procedure CHECKTEMPLATE checks compatibility of CSMs efficiently. Based on Definition 14, checking compatibility of two CSMs reduces to checking the existence of a strategy that follows the templates $\Psi_{1-i} \wedge \Pi_i \lhd (S, D, \mathcal{H})$ for both $i \in \{0, 1\}$. As our templates are just particular LTL formulas, one can of course use automata-theoretic techniques to check this. However, given the edge sets $(S, D, \mathcal{H})$ this check can be performed more efficiently as formalized next.

Definition 20. 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., $v \times E(v) \nsubseteq D \cup S$, 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., $v \times H(v) \nsubseteq D \cup S$. The procedure of checking (i)-(ii) is called CHECKTEMPLATE(G, Λ), which returns `true` if (i)-(ii) holds, and `false` otherwise.

We note that checking (i)-(ii) can be done independently for every vertex, hence CHECKTEMPLATE(G, Λ) runs in linear time $O(n)$ for $n = |V|$. Intuitively, whenever the existentially quantified edge in (i) and (ii) of Definition 20 exists, a strategy that alternates between all these edges follows the given template. In addition, this strategy can also be extracted in linear time. This is formalized next.

Proposition 21. Given a game graph $G = (V, E)$ with conflict-free template $\Lambda \lhd (S, D, \mathcal{H})$ for Player i, a strategy π_i for Player i that follows Λ can be extracted in time $O(m)$, where m is the number of edges. This procedure is called ExtractStrategy(G, Λ).

Proof. The proof is straightforward by constructing the strategy as follows. We first remove all unsafe and co-live edges from G and then construct a strategy π_i that alternates between all remaining edges from every vertex. This strategy is well-defined as condition (i) in Definition 20 ensures that after removing all the unsafe and co-live edges a choice from every vertex remains. Moreover, if the vertex is a source of a live-group edge, condition (ii) in Definition 20 ensures that there are outgoing edges satisfying every live-group. Thereby, the constructed strategy indeed follows Λ. □

It is worth noting that ParityTemp always return conflict-free templates Ψ_i and Π_i by construction. Only when combining templates from different players into $\Pi_i \wedge \Psi_{1-i}$ conflicts may arise. However, as conflict-freeness of template $\Psi_{1-i} \wedge \Pi_i$ implies the existence of a Player i strategy following it from Proposition 21, this immediately implies that both CSMs are compatible, leading to the following corollary.

Corollary 22. Given two CSMs (Ψ_0, Π_0) and (Ψ_1, Π_1) in a game graph G, if for each $i \in \{0, 1\}$ the template $\Psi_{1-i} \wedge \Pi_i$ is conflict-free, then the two CSMs are compatible.

We note that the converse of Corollary 22 is not true, as there can be a strategy following $\Psi_{1-i} \wedge \Pi_i$ even when the corresponding CSMs are not conflict-free. However, this does not affect the completeness of our algorithm. Therefore, we focus our attention on ensuring conflict-freeness rather than compatibility. Moreover, if such a strategy exists it will be retained by the conflict resolving mechanism of Negotiate, introduced next.

5.2 Resolving Conflicts

Given a conflict in $\Psi_{1-i} \wedge \Pi_i \lhd (S, D, \mathcal{H})$ we now discuss how the modified specifications Φ'_i (as in Line 5 of Algorithm 1) allows to resolve this conflict in the next iteration.

For this, first assume that $D = \emptyset$. In this case a conflict exists because all available (live) edges are unsafe and should never be taken. Hence, an extracted strategy (via Proposition 21) is not well-defined (i.e., might get stuck in a vertex for which (i) of Definition 20 is false) or not ensured to be winning (i.e., will not be able to fulfill the liveness obligations in H_ℓ if (ii) of Definition 20 is false).

In order to ensure strategies to be winning, templates need to be re-computed over a game graph where unsafe edges $e \in S$ in $\Psi_{1-i} \wedge \Pi_i$ are removed. By looking into the details of the computation of S within ParityTemp, we see that unsafe edges always transition from the winning region $\mathcal{W}_i = \langle\langle i \rangle\rangle(\Psi_i, \Pi_i)$ to its complement $\overline{\mathcal{W}}_i = \neg\mathcal{W}_i$, i.e., every (cooperatively winning) play should never visit states in $\overline{\mathcal{W}}_i$. We therefore achieve the desired effect by adding the requirement $\square\neg(\overline{\mathcal{W}}_0 \cup \neg\overline{\mathcal{W}}_1) = \square(\mathcal{W}_0 \cap \mathcal{W}_1)$ to the specification, which obviously does not restrict the cooperative winning region, as ParityTemp is ensured to not remove any cooperative solution (due to Item (i) in Definition 16).

This intuition generalizes to the case where $D \neq \emptyset$ as follows. Here, we need to resolve the game while ensuring that co-live edges $e \in D$ are only taken finitely often. In analogy to unsafe edges, co-live edges are computed by ParityTemp s.t. they always transition to the set of vertices C_i that must only be seen finitely often along a winning play. In addition to $\mathcal{W}_i$, the set C_i can also be memorized during the computation of D within ParityTemp and hence passed to CheckTemplate in Line 1 of Algorithm 1. As for the unsafe-edge case, we can achieve the desired effect for recomputation by adding the requirement $\lozenge\square\neg(C_0 \cup C_1)$ to the specification Φ_i (see Line 5 of Algorithm 1). Again, this obviously does not alter the cooperative winning region of the game.

Remark 23. We note that Algorithm 1 is slightly simplified, as the objective Φ'_i in Line 5 of Algorithm 1 used as an input to Negotiate in latter iterations, is not a "plain" parity objective *Parity*$(\mathbb{P})$. As ParityTemp expects a parity game as an input, we need to convert (G, Φ'_0, Φ'_1) into a parity game by a simple reprocessing step. Luckily, both additional specifications can be dealt with using classical steps of Zielonka's algorithm [28], a well-known algorithm to solve parity games, which is used as the basis for ParityTemp. Concretely, we handle the $\square(\mathcal{W}_0 \cap \mathcal{W}_1)$ part of Φ'_i, by restricting the game graph G to $\mathcal{W} = \mathcal{W}_0 \cap \mathcal{W}_1$ and the $\lozenge\square\neg(C_0 \cup C_1)$ part by assigning all vertices in $C = (C_0 \cup C_1)$ the highest odd priority $2d_i + 1$. The correctness of these standard transformations follows from the same arguments as used to prove the correctness of similar steps of the ParityTemp algorithm in the extended version [6].

5.3 Properties of Negotiate

With this, we are finally ready to prove that (i) Negotiate always terminates in a finite number of steps, and (ii) upon termination, the computed CSMs indeed provide a solution to Problem 8.

Termination. Intuitively, all local synthesis problems are performed over the same (possibly shrinking) game graph G. Therefore, there exists only a finite number of templates $\Lambda \lhd (S, D, \mathcal{H})$ over G, which, in the worst case, can all be enumerated in finite time.

Theorem 24. Given a two-objective parity game $\mathcal{G} = ((V, E), \Phi_0, \Phi_1)$ with $\Phi_i = $ *Parity*$(\mathbb{P}_i)$, Algorithm 1 always terminates in $O(n^6)$ time, where $n = |V|$.

Proof. We prove termination via an induction on the lexicographically ordered sequence of pairs $(|\mathcal{W}|, |\mathcal{W} \setminus C|)$. As the base case, observe that if $|\mathcal{W}| = 0$ we have that $\mathcal{W}_i = \emptyset$ for at least on Player i, implying $\Psi_i \lhd (\emptyset, \emptyset, \emptyset)$ and $\Pi_i \lhd (\emptyset, \emptyset, \emptyset)$ for this Player i. As $\Psi_i \wedge \Pi_{1-i} = \Pi_{1-i}$ and $\Psi_{1-i} \wedge \Pi_i = \Psi_{1-i}$ in this case, and Ψ_{1-i} and Π_{1-i} are conflict-free by construction, CheckTemplate returns true and Negotiate terminates. If instead only $|\mathcal{W} \setminus C| = 0$, it follows from Remark 23 that all vertices in $\mathcal{W}$ have highest odd priority, implying that in the next iteration $\mathcal{W}_i = \emptyset$ for both players, hence all templates are empty, i.e., trivially conflict-free, hence Negotiate terminates.

Now for the induction step, suppose $|\mathcal{W}| > 0$ and $|\mathcal{W} \setminus C| > 0$ in the *previous* iteration. If $\Psi_0 \wedge \Pi_1$ and $\Psi_1 \wedge \Pi_0$ are conflict-free, Negotiate terminates. Suppose this is not the case. As G gets restricted to $\mathcal{W}$ for this iteration (see Remark 23), unsafe edges can only occur if $\mathcal{W}' \subset \mathcal{W}$ (as they are by construction from $\mathcal{W}'$ to $\neg\mathcal{W}'$ s.t. the latter is a subset of $\mathcal{W}$), where $\mathcal{W}'$ is the winning region computed in the *current* iteration. If $\mathcal{W}' = \mathcal{W}$ conflicts need to arise from colive edges. As colive edges are computed by

ParityTemp in a subgame that excludes all vertices with the highest odd priority (and therefore all vertices in C due to Remark 23), the existence of co-live edge conflicts implies the existence of co-live edges, which implies that $|\mathcal{W}' \setminus C'| < |\mathcal{W} \setminus C|$. Therefore, $(|\mathcal{W}|, |\mathcal{W} \setminus C|)$ always reduces (lexicographically) when conflicts occur. Hence, the algorithm terminates by induction hypothesis. Furthermore, as each iteration calls CheckTemplate once and *Parity* twice which runs in $O(|V|^4)$ time, Algorithm 1 terminates in $O(|V|^6)$ time. □

Soundness. While it seems to immediately follow that iRmaC-specifications can be CSMs that Negotiate outputs, as it only terminates on adequately permissive and compatible CSMs, this is only true w.r.t. the new game (G, Φ_0', Φ_1') which gets modified in every iteration. It therefore remains to show that the resulting CSMs induce iRmaC-specification for (G, Φ_0, Φ_1), which then proves that Negotiate solves Problem 8.

Theorem 25. Let $((\Phi_0'', \Phi_1''), (\Psi_0, \Pi_0), (\Psi_1, \Pi_1))$ be the output of Negotiate(G, Φ_0, Φ_1). Then $(\Phi_0^\bullet, \Phi_1^\bullet)$ with $\Phi_i^\bullet := \Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i'')$ are iRmaC-specifications for (G, Φ_0, Φ_0).

Proof. As CSM (Ψ_i, Π_i) is adequately permissive for Player i in the game (G, Φ_i'') and as the returned CSMs are compatible, by using Theorem 18, the contracted specifications $(\Phi_0^\bullet, \Phi_1^\bullet)$ for the two-objective game (G, Φ_0'', Φ_1'') are iRmaC-specifications. Hence, $\mathcal{L}(\Phi_0'' \wedge \Phi_1'') = \mathcal{L}(\Phi_0^\bullet \wedge \Phi_1^\bullet)$ and $\langle\langle 0,1\rangle\rangle \Phi_0'' \wedge \Phi_1'' = \langle\langle 0\rangle\rangle \Phi_0^\bullet \cap \langle\langle 1\rangle\rangle \Phi_1^\bullet$. Hence, in order to prove that (5) holds, it suffices to show that $\Phi_0'' \wedge \Phi_1''$ is equivalent to $\Phi_0 \wedge \Phi_1$, i.e., $\mathcal{L}(\Phi_0'' \wedge \Phi_1'') = \mathcal{L}(\Phi_0 \wedge \Phi_1)$. This however immediately follows from the fact that ParityTempcomputes $\mathcal{W}_i$ and C_i in an adequately permissive manner, i.e., never excluding any cooperative winning play. Thereby, the addition of the terms $\square(\mathcal{W}_0 \cap \mathcal{W}_1)$ and $\lozenge\square\neg(C_0 \cup C_1)$ to the specification does not exclude cooperative winning plays either, hence keeping $\mathcal{L}(\Phi_0 \wedge \Phi_1)$ the same in each iteration. □

Decoupled Strategy Extraction. By combining the properties of iRmaC-specifications with Proposition 21, we have the following proposition which shows that by using templates to formalize iRmaC contracts, we indeed fully decouple the strategy choices for both players.

Proposition 26. In the context of Theorem 25, let π_i be a strategy of player i following $\Psi_{1-i} \wedge \Pi_i$. Then

(i) π_i is winning in $(G, \Phi_i^\bullet)$ from every $v \in \langle\langle 0,1\rangle\rangle(\Phi_0 \cap \Phi_1)$, and
(ii) the strategy profile (π_0, π_1) is winning in (G, Φ_0, Φ_1).

Proof. **(i)** As the CSM (Ψ_i, Π_i) is adequately permissive for Player i in the game (G, Φ_i''), by Remark 17, the sufficiency condition makes it winning from all vertices in $\langle\langle 0,1\rangle\rangle \Phi_i'' \supseteq \langle\langle 0,1\rangle\rangle(\Phi_0'' \wedge \Phi_1'') = \langle\langle 0,1\rangle\rangle(\Phi_0 \wedge \Phi_1)$. Moreover, as π_i follows Π_i, by using Remark 13, π_i is winning in the game $(G, \Psi_i \Rightarrow \Phi_i'')$ from $\langle\langle 0,1\rangle\rangle(\Phi_0 \wedge \Phi_1)$. Hence, every π_i-play from $\langle\langle 0,1\rangle\rangle(\Phi_0 \wedge \Phi_1)$ satisfies both Ψ_{1-i} and $\Psi_i \Rightarrow \Phi_i''$. As $\Phi_i^\bullet = \Psi_{1-i} \wedge (\Psi_i \Rightarrow \Phi_i'')$, strategy π_i is winning in the game $(G, \Phi_i^\bullet)$ from $\langle\langle 0,1\rangle\rangle(\Phi_0 \wedge \Phi_1)$.

(ii) This now follows directly from (i) and Proposition 7. □

Completeness. As our final result, we note that as a simple corollary from Proposition 21 and Proposition 26 follows that whenever a cooperative solution to the original synthesis problem (G, Φ_0, Φ_1) exists, we can extract a winning strategy profile from the CSMs computed by Negotiate.

Corollary 27. In the context of Theorem 25, for any vertex v from which there exists a winning strategy profile (π_0', π_1') for the two-objective parity game (G, Φ_0, Φ_1), there exist strategies π_i'' from v following $\Psi_{1-i} \wedge \Pi_i$ for both $i \in \{0, 1\}$.

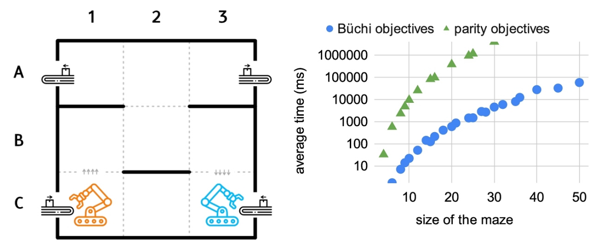


Figure 4: Left: Example of a factory benchmark with parameters $x = 3$, $y = 3$, $w = 3$, and $c = 2$. Solid lines denote walls, little up- and downward pointing arrows indicate one-way corridors. Right: Data points for factory benchmarks with Büchi objectives (blue circles) and parity objectives (green triangles) describing average execution time over all instances with the same grid size. The y-axis is given in log-scale.

6 EXPERIMENTAL EVALUATION

To demonstrate the effectiveness of our approach, we conducted experiments using a prototype tool CoSMo [1] that implements the negotiation algorithm (Algorithm 1) for solving two-objective parity games. All experiments were performed on a computer equipped with an Apple M1 Pro 8-core CPU and 16GB of RAM.

6.1 Factory Benchmark

Building upon the running example in Section 1, we generated a comprehensive set of 2357 factory benchmark instances. These instances simulate two robots, denoted as R_1 and R_2, navigating within a maze-like workspace. We used four parameters, i.e., size of the maze $x \times y$, number of walls w, and maximum number of one-way corridors c. First, we consider the Büchi objective that robots R_1 and R_2 should visit the upper-right and upper-left corners, respectively, of the maze infinitely often, while ensuring that they never occupy the same location simultaneously and do not bump into a wall. Second, we consider the parity objectives from Example 1. Further details regarding the generation of these benchmark instances can be found in the extended version [6]. An illustration of one such benchmark is depicted in Fig. 4 (left).

Experimental Results. In a first set of experiments, we ran our tool on all the factory benchmarks instances and plot all average run-times per grid-size (but with varying parameters for c and w) in Fig. 4 (right). We see that CoSMo takes significantly more time for parity objectives compared to Büchi objectives. That is because computing templates for Büchi games takes linear time in the size of the games whereas the same takes biquadratic time for parity games (see [6]). Furthermore, the templates computed for Büchi objectives do not contain co-liveness templates, and hence, they do not raise conflicts in most cases. However, templates for parity

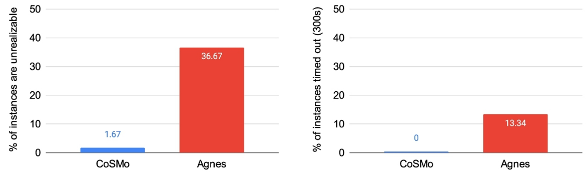


Figure 5: Left: Percentage of instances on which the respective tool reports unrealizability after termination. Right: Percentage in which the respective tool does not terminate. Both numbers are mutually exclusive.

objectives contain all types of templates and hence, typically need several rounds of negotiation.

In a second set of experiments, we compared the performance of CoSMo, with the related tool[9] agnes implementing the contract-based distributed synthesis method from [21]. Unfortunately, agnes can only handle Büchi specifications and resulted in segmentation faults for many benchmark instances. We have therefore only report computation times for all instances that have not resulted in segmentation faults.

The experimental results are summarized in Fig. 5-6. As CoSMo implements a complete algorithm, it provably only concludes that a given benchmark instance is unrealizable, if it truly is unrealizable, i.e., for 1.67% of the considered 120 instances. agnes however, concludes unrealizability in 36, 67% of its instances (see Fig. 5 (left)), resulting an many false-negatives. Similarly, as CoSMo is ensured to always terminate, we see that all considered instances have terminated in the given time bound. While, agnes typically computes a solution faster for a given instance (see Fig. 6 (left)), it enters a non-terminating negotiation loop in 13, 34% of the instances (see Fig. 5 (right)). This happens for almost all considered grid sizes, as visible from Fig. 6 (right) where all non-terminating instances are included in the average after being mapped to $300s$, which was used as a time-out for the experiments.

While our experiments show that agnes outperforms CoSMo in terms of computation times when it terminates on realizable instances (see Fig. 6 (left)), it is unable to synthesize strategies either due to conservatism or non-termination in almost 50% of the considered instances (in addition to the ones which returned segmentation faults and which are therefore not included in the results). In addition to the fact that agnes can only handle the small class of Büchi specifications while CoSMo can handle parity objectives, we conclude that CoSMo clearly solves the given synthesis task much more satisfactory.

6.2 Incremental Synthesis and Negotiation

While the previous section evaluates our method for a single, static synthesis task, we want to now emphasize the strength of our technique for the online adaptation of strategies. To this end, we assume that Algorithm 1 has terminated on the input (G, Φ_0, Φ_1) and compatible CSM's (Ψ_0, Π_0) and (Ψ_1, Π_1) have been obtained. Then a new parity objective Φ_i' over G arrives for component i, for which additional CSM $(\Psi_i', \Pi_i') := \textsc{ParityTemp}(G, \Phi_i')$ can be computed. It is easy to observe that if (Ψ_i', Π_i') does not introduce new conflicts, no further negotiation needs to be done and the CSM of component i can simply be updated to $(\Psi_i \wedge \Psi_i', \Pi_i \wedge \Pi_i')$. Otherwise, we simply re-negotiate by running more iterations of Algorithm 1.

[9]Unfortunately, a comparison with the only other related tool [18] which allows for parity objectives was not possible, as we were told by the authors that their tool became incompatible with the new version of BoSy and is therefore currently unusable.

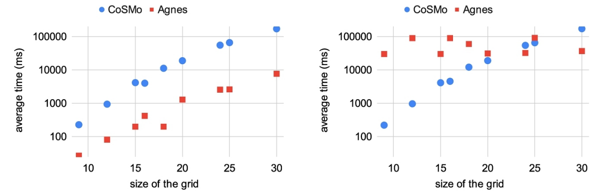


Figure 6: Average computation times over all instances with the same grid size for CoSMo (blue circles) and agnes (red squares) without timed-out instances (left) and with timed-out instances mapped to the time-out of $300s$ (right). The y-axis is given in log-scale.

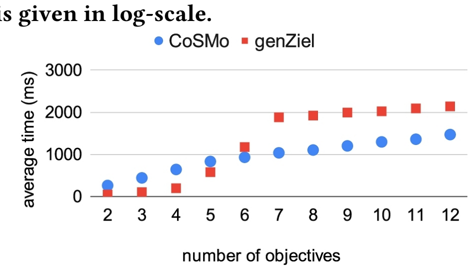


Figure 7: Experimental results over 2244 games when new parity objectives are added *incrementally one-by-one*. Data points give the average execution time (in ms) over all instances with the same number of parity objectives for CoSMo (blue circles) and genZiel [16] (red squares).

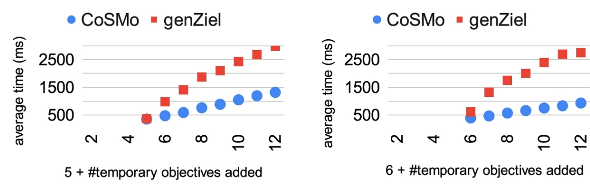


Figure 8: Variation of the experiment in Fig. 7 with either 5 (left) or 6 (right) long-term objectives.

We note that, algorithmically, this variation of the problem requires solving a chain of *generalized* parity games, i.e., a parity game with a conjunction of a finite number of parity objectives. We therefore compare the performance of CoSMo on such synthesis problems[10] to the best known solver for *generalized* parity games, i.e, genZiel from [16] (implemented by [14]). Similar to our approach, genZiel is complete and based on Zielonka's algorithm. However, it solves one *centralized* cooperative game for the conjunction of all players objectives.

Comparative evaluation. Fig. 7 shows the average computation time of genZiel and CoSMo when objectives are *incrementally one-by-one*, i.e., the game was solved with ℓ objectives, then one more objective was added and the game was solved it again. We see that for a low number of objectives, the negotiation of contracts

[10]The details about the benchmarks can be found in the extended version [6].

in a distributed fashion by CoSMo adds computational overhead, which reduces when more objectives are added. However, as more objectives are added the chance of the winning region to become empty increases. This gives genZiel an advantage, as it can detect an empty winning region very quickly (due to its centralized computation). In order to separate the effect of (i) the increased number of re-computations and (ii) the shrinking of the winning region induced by an increased number of incrementally added objectives, we conducted a section experiment where added objectives are allowed to disappear again after some time. Here, we consider benchmarks with a fixed number of long-term objectives, and iteratively add just one *temporary* objective at a time. The results are summarized in Fig. 8 when the number of long-term objectives are 5 (left) and 6 (right). We see that in this scenario CoSMo clearly outperforms genZiel, while performing computations in a distributed manner and returning strategy templates.

REFERENCES

[1] [n. d.]. CoSMo: A tool for computing assumption and strategy templates as contracted strategy masks for both players in two-objective parity games. github.com/satya2009rta/cosmo [Accessed 10-04-2024].

[2] Martín Abadi and Leslie Lamport. 1990. Composing specifications. In *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, J. W. de Bakker, W. P. de Roever, and G. Rozenberg (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–41.

[3] Rajeev Alur. 2015. *Principles of cyber-physical systems.* MIT press.

[4] R. Alur, S. Moarref, and U. Topcu. 2018. Compositional and symbolic synthesis of reactive controllers for multi-agent systems. *Inf. Comput.* 261 (2018), 616–633. https://doi.org/10.1016/j.ic.2018.02.021

[5] Ashwani Anand, Kaushik Mallik, Satya Prakash Nayak, and Anne-Kathrin Schmuck. 2023. Computing Adequately Permissive Assumptions for Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer Nature Switzerland, Cham, 211–228.

[6] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. 2023. Contract-Based Distributed Synthesis in Two-Objective Parity Games. *CoRR* abs/2307.06212 (2023). https://doi.org/10.48550/ARXIV.2307.06212 arXiv:2307.06212

[7] Ashwani Anand, Satya Prakash Nayak, and Anne-Kathrin Schmuck. 2023. Synthesizing Permissive Winning Strategy Templates for Parity Games. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13964)*, Constantin Enea and Akash Lal (Eds.). Springer, 436–458. https://doi.org/10.1007/978-3-031-37706-8_22

[8] W. Alejandro Apaza-Perez and Antoine Girard. 2021. Compositional Synthesis of Symbolic Controllers for Attractivity Specifications. In *2021 60th IEEE Conference on Decision and Control (CDC).* 2008–2013. https://doi.org/10.1109/CDC45484.2021.9682817

[9] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking.* MIT press.

[10] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. 2017. *Formal methods for discrete-time dynamical systems.* Vol. 15. Springer.

[11] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A.L. Sangiovanni-Vincentelli, W. Damm, T.A. Henzinger, and K.G. Larsen. 2018. Contracts for System Design. *Found. Trends in EDA* 12, 2-3 (2018), 124–400.

[12] R. Bloem, K. Chatterjee, S. Jacobs, and R. Könighofer. 2015. Assume-guarantee synthesis for concurrent reactive programs with partial information. In *TACAS.* Springer, 517–532.

[13] R. Brenguier, J.-F. Raskin, and O. Sankur. 2017. Assume-admissible synthesis. *Acta Informatica* 54, 1 (Feb 2017), 41–83.

[14] Véronique Bruyère, Guillermo A. Pérez, Jean-François Raskin, and Clément Tamines. 2019. Partial Solvers for Generalized Parity Games. In *Reachability Problems - 13th International Conference, RP 2019, Brussels, Belgium, September 11-13, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11674)*, Emmanuel Filiot, Raphaël M. Jungers, and Igor Potapov (Eds.). Springer, 63–78. https://doi.org/10.1007/978-3-030-30806-3_6

[15] K. Chatterjee and T.A Henzinger. 2007. Assume-guarantee synthesis. In *TACAS.* Springer, 261–275.

[16] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2007. Generalized Parity Games. In *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4423)*, Helmut Seidl (Ed.). Springer, 153–167. https://doi.org/10.1007/978-3-540-71389-0_12

[17] Eric Dallal and Paulo Tabuada. 2015. On compositional symbolic controller synthesis inspired by small-gain theorems. In *2015 54th IEEE Conference on Decision and Control (CDC).* 6133–6138. https://doi.org/10.1109/CDC.2015.7403184

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

[19] Antoine Girard, Alessio Iovine, and Sofiane Benberkane. 2022. Invariant Sets for Assume-Guarantee Contracts. In *2022 IEEE 61st Conference on Decision and Control (CDC).* 2190–2195. https://doi.org/10.1109/CDC51059.2022.9993344

[20] Eric S. Kim, Sadra Sadraddini, Calin Belta, Murat Arcak, and Sanjit A. Seshia. 2017. Dynamic contracts for distributed temporal logic control of traffic networks. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC).* 3640–3645. https://doi.org/10.1109/CDC.2017.8264194

[21] Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Damien Zufferey. 2020. Assume–Guarantee Distributed Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3215–3226. https://doi.org/10.1109/TCAD.2020.3012641

[22] Kaushik Mallik, Anne-Kathrin Schmuck, Sadegh Soudjani, and Rupak Majumdar. 2019. Compositional Synthesis of Finite-State Abstractions. *IEEE Trans. Automat. Control* 64, 6 (2019), 2629–2636. https://doi.org/10.1109/TAC.2018.2869740

[23] J. Misra and K.M. Chandy. 1981. Proofs of Networks of Processes. *IEEE Transactions on Software Engineering* SE-7, 4 (1981), 417–426. https://doi.org/10.1109/TSE.1981.230844

[24] Satya Prakash Nayak, Lucas N. Egidio, Matteo Della Rossa, Anne-Kathrin Schmuck, and Raphael M. Jungers. 2023. Context-Triggered Abstraction-Based Control Design. *IEEE Open Journal of Control Systems* 2 (2023), 277–296. https://doi.org/10.1109/OJCSYS.2023.3305835

[25] Amir Pnueli. 1985. In Transition From Global to Modular Temporal Reasoning about Programs. In *Logics and Models of Concurrent Systems*, Krzysztof R. Apt (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–144.

[26] Eugene W. Stark. 1985. A proof technique for rely/guarantee properties. In *Foundations of Software Technology and Theoretical Computer Science*, S. N. Maheshwari (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–391.

[27] Paulo Tabuada. 2009. *Verification and Control of Hybrid Systems - A Symbolic Approach.* Springer.

[28] Wieslaw Zielonka. 1998. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science* 200, 1 (1998), 135–183. https://doi.org/10.1016/S0304-3975(98)00009-7