

# Symbolic construction of GR(1) contracts for synchronous systems with full information

Ioannis Filippidis      Richard M. Murray

{ifilippi,murray}@caltech.edu

Control and Dynamical Systems  
California Institute of Technology

October 5, 2018

## Abstract

This work proposes a symbolic algorithm for the construction of assume-guarantee specifications that allow multiple agents to cooperate. Each agent is assigned goals expressed in a fragment of linear temporal logic known as generalized reactivity of rank 1 (GR(1)). These goals may be unrealizable, unless additional assumptions are made by each agent about the behavior of the other agents. The proposed algorithm constructs weakly fair assumptions for each agent, to ensure that they can cooperate successfully. A necessary requirement is that the given goals be cooperatively satisfiable. We prove that there exist games for which the GR(1) fragment with liveness properties over states is not sufficient to ensure realizability from any state in the cooperatively winning set. The obstruction is due to circular dependencies of liveness goals. To prevent circularity, we introduce nested games as a formalism to express specifications with conditional assumptions. The algorithm is symbolic, with fixpoint structure similar to the GR(1) synthesis algorithm, implying time complexity polynomial in the number of states, and linear in the number of recurrence goals.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Modular design by contract</b>	<b>3</b>
<b>3</b>	<b>Games</b>	<b>4</b>
<b>4</b>	<b>Proposed approach</b>	<b>5</b>
<b>5</b>	<b>Preliminaries</b>	<b>6</b>
5.1	Turn-based synchronous games . . . . .	6
5.2	Integrals . . . . .	7
5.3	Linear temporal logic . . . . .	8
5.4	Interleaving representation of a Streett(1) game . . . . .	9
<b>6</b>	<b>Property closure</b>	<b>10</b>
6.1	Cooperative winning set . . . . .	10
6.2	Computing the closure . . . . .	12
<b>7</b>	<b>Construction of weak fairness assumptions for a single goal</b>	<b>15</b>
7.1	The role of machine closure . . . . .	15
7.2	Nonexistence of weak fairness assumptions over nodes . . . . .	16

## 1 Introduction

The design and construction of a large system relies on the ability to divide the problem into smaller ones. Each subproblem involves a subset of the system, and may itself be refined further into smaller problems. The subsystems that result from the smaller problems are considered as modules of the larger system. In many cases, the modules interact with each other, either physically, or as software, or both. For this reason, the interaction between modules needs to be constrained, in order to ensure that the modules can perform their operation as intended. For example, if we consider the fridge and a power plug as modules of a house, then the fridge can only preserve food provided that the plug provides electric power uninterruptedly. In many cases, modularization is a necessity, imposed by the topology of the design, because the system comprises of elements distributed over space. These elements control some local part of the system, but they need to communicate, in order to coordinate.

Among the benefits of modularization are the division of a complex problem into smaller ones that are computationally cheaper to solve, the localization of reasoning, which focuses the designer's attention and reduces the danger of errors, and the ability to assign the design of each subsystem to a different entity, for example a contractor specializing in that type of system. In addition, a well-defined description of each individual component enables re-using the same design in a different context, where such a component is needed. This leads to the possibility of interfacing off-the-shelf components, based on their interface description, thus reducing the need for case-by-case design and production.

In order to describe a module and its interaction with other modules, and their environment, it is necessary to represent them. A representation can range from an informal textual description, to a mathematically defined notation, with fixed syntax and semantics. The latter is desirable, because it is not ambiguous, and it enables automation of checking whether a candidate solution satisfies the requirements. Such a formal representation is usually called a *specification*.

Proving that a system will behave as intended, insofar as this is captured by a specification, is a major objective in systems that are critical for the safety of humans, or have a very high cost. These include aircraft, especially airliners, spacecraft, which is a major investment and missions are, in many cases, unique and not to be repeated, automotive subsystems, nuclear power plant controllers, and several other application areas.

We can distinguish two broad problems, at different phases of system design. The first one asks for producing formal specifications that describe the modules, with detail sufficient to allow for automated synthesis. The second problem asks for constructing an implementation of each module, and assembling the results into a complete composite system. The first problem comprises the modularization and specification step, whereas the second is the construction phase.

The specification of a system can be implemented by humans, or constructed by an algorithm. The latter approach is known as (automated) *synthesis*, and relies on notions from the theory of games [1]. Synthesis has attracted considerable interest in the past two decades, and advances both in theory and implementation have been made, as described in the following sections. In this work, we are interested in algorithmic synthesis, for both phases of system design. In particular, we aim at automatically modularizing a design that has been partially specified by a human. In other words, humans give as input a formal description about what each module is expected to accomplish. Note that this step is necessary, in one form or another, because the algorithm cannot know what the modules are intended for. We consider these as the primitive specifications, that are given by a human, and will be completed algorithmically. These specifications may be insufficient for obtaining a coherent system, but describe the goals, and provide the starting point for an algorithmic approach to complete the specifications, and then construct implementations. The automated modularization step involves *completing* the specifications, by adding more detail, in a way that ensures that there exist components satisfying the primitive specifications. Regarding the synthesis phase, we are interested in efficient and scalable synthesis algorithms that can handle specifications with many goals.

Clearly, the formal description of all the details in a given implementation is itself a specification. However, fixing a particular implementation is usually much more restrictive than needed. It is desired to describe only what is necessary of a particular module, and leave the internal details of its exact operation to be decided by the implementor of that module. The difference between an implementation, and a less restrictive

specification is *quantification*. A specification contains *existential* quantification, if it asks for some type of behavior contained in a given set, but does not describe a particular instance of that behavior. Another term that is commonly used to characterize this quality of a specification is *declarative*.

This motivates regarding synthesis as a *compiler* activity. In analogy with declarative programming languages like Haskell, a declarative specification is intended to leave unconstrained the exact imperative details of *how* the implementation will behave, step-by-step. A synthesizer from declarative specifications *compiles* them into an implementation that operates in time by reading environment inputs, and writing outputs. Reading and writing are used here in a broad sense, meaning interaction that may involve mechanical or hydraulic forces. A distinction between conventional declarative languages, and module synthesis is that the latter produces components that continue to interact with their environment without ever terminating, also known as *reactive* systems. For example, the computer that controls a nuclear reactor is not intended to terminate and produce some result, under normal operation. This is in contrast to a matrix multiplication program.

## 2 Modular design by contract

There have been several approaches to the modularization of systems. The design of each module becomes simpler, because it involves fewer elements, as counted, for example, by the number of variables used to represent it. However, the challenge is shifted, from designing a monolithic system, to putting together the pieces. In this report, we consider the problem of interfacing the modules.

Our approach constructs specifications that are partitioned into assumptions about the behavior of the world outside a component, and requirements that the component guarantees, provided its assumptions hold. This is known as assumption-commitment, or rely-guarantee paradigm for describing behaviors.

The assumption-commitment paradigm about reactive systems is an evolved instance of reasoning about conditions before, and after, a terminating behavior. A formalism for reasoning using triples of a *precondition*, a program, and a *postcondition* was introduced by Hoare [2], following the work of Floyd [3] on proving properties of elements in a flowchart, based on ideas by Perlis and Gorn [4].

Hoare's logic applies to terminating programs. However, many systems are not intended to terminate, but instead continue to operate, by reacting to their environment [5]. Francez and Pnueli [6] introduced a first generalization of Hoare-style reasoning to cyclic programs. They also considered concurrent programs. Their formalism uses explicit mention of time, and is structured into pairs of assumptions and commitments.

Lamport [7] observed that such a style of specification is essential to reason about complex systems in a modular way. Lamport and Schneider [8, 9] introduced, and related to previous approaches, what they called *generalized Hoare logic*. This is a formalism for reasoning with pre- and post-conditions, in order to prove program invariants. Misra and Chandy introduced the rely-guarantee approach for safety properties of distributed systems [10], still for safety properties. All properties up to this point were safety, and not expressed in temporal logic [11]. Two developments followed, and the work presented here is based on them.

The first was Lamport's introduction of *proof lattices* [12]. A proof lattice is a finite rooted directed acyclic graph, labeled with assertions. If  $u$  is a node labeled with property  $U$ , and  $v, w$  are its successors, labeled with properties  $V, W$ , then if  $U$  holds at any time, eventually either  $V$  or  $W$  will hold. In temporal logic, this can be expressed as  $\Box(U \rightarrow \Diamond(V \vee W))$ . Owicki and Lamport [13] revised the proof lattice approach, by labeling nodes with *temporal* properties, instead of atemporal ones (immediate assertions).

The second development was the expression by Pnueli [14] of assume-guarantee pairs in temporal logic, i.e., without reference to an explicit *time* variable. In addition, Pnueli proposed a proof method for *liveness* properties, which is based on well-founded induction. This method can be understood as starting with some temporal premises for each component, and iteratively tightening these properties into consequents that are added to the collection of available premises, for the purpose of deriving further consequents. This method enables proving liveness properties of modular systems. Informally, the requirement of well-foundedness allows using as premises only properties from an earlier stage of the deductive process. This prevents circular existential reasoning about the future, i.e., circular dependencies of liveness properties. As a simple example [15], consider Alice and Bob. Alice promises that, if she sees  $b$ , then she will do  $a$  at *some* time in the future. Reciprocally, Bob promises to eventually do  $b$ , after he sees  $a$ . As linear temporal logic (LTL) formulae, these read  $\Box(b \rightarrow \bigcirc \Diamond a)$  for Alice, and with  $a, b$  swapped, for Bob. If both Alice and Bob default

to not doing any of  $a$  or  $b$ , then they both satisfy their specifications. This problem arises, because existential quantification in *future*<sup>1</sup> time allows simultaneous antecedent failure. Otherwise, if Bob was *required* to do  $b$  for the first time, then Alice would have to do  $a$ , then Bob do  $b$  again, etc.

Compositional approaches to verification have treated the issue of circularity by using the description of the model under verification as a vehicle for carrying out the proof. In other words, the immediate behavior of the model, as captured by its transition relation, should constrain the system sufficiently much, so as to enable deducing the satisfaction of its liveness guarantees, as in the work of Abadi and Lamport [16]. This approach is suitable for verification, because the model is available at that stage. However, in the automated construction of specifications for synthesis, we prefer to quantify over time, instead of describing immediate behavior. Therefore, we desire to be able to reason about dependencies of liveness properties between modules, with minimal reliance on the implementation, i.e., on safety properties. Stark [17] proposed a proof rule for assume-guarantee reasoning about a non-circular set of liveness properties. McMillan [18] introduced a proof rule for circular reasoning about liveness. However, this proof system is intended for verification, so it relies on the availability of a model. It requires the definition of a proof lattice, and introduces graph edges that *consume* time, as a means to break simultaneity cycles. The method we propose in this work constructs specifications that can have dependencies of liveness goals, but in a way that avoids circularity. It is discussed in Section 4.

The assumption-guarantee paradigm has since evolved, and renamed several times. Meyer [19] called the paradigm *design by contract*, and supported its use for abstracting software libraries, and validating the correct operation of software. The notion of a contract generalizes assume-guarantee reasoning, because a contract can have several forms. For example, it may come in the form of an *interface automaton* [20], which offers only an implicit description of assumptions, as those environments that can be successfully connected to the interface. The interface automaton abstracts the internal details of a module, and serves as its surface appearance towards other modules.

More recently, contracts have been proposed for specifying the design of systems with both physical and computational aspects [21]. In this context, contracts are used broadly, as an umbrella term that encompasses both interface theories and assume-guarantee contracts [22, 21], with extensions to timed and probabilistic specifications. A proof system for verifying that a set of contracts refines a contract for the composite system has been proposed in [23]. A verification tool of contract refinement using an SMT solver is described in [24]. This body of work focuses mainly on using, or manipulating, existing contracts. We are interested in *constructing* contracts.

### 3 Games

In this section, we review relevant results from the literature on games of infinite duration. The literature is extensive, so we restrict to a sample that we consider representative. The problem of constructing a module that exhibits a desired set of behaviors in time can be solved with algorithms that solve games. There are different types of games, depending on:

- how many transducers are being constructed inside a single system,
- the order of player choice,
- the winning condition,
- the visibility of variables, and
- the number of players.

Games can be turn-based, where a single player moves in each time step, or concurrent [25, 26]. In synchronous games, turns are taken with a fixed schedule, whereas asynchronous games are scheduled dynamically by a dedicated player called scheduler [27].

If we want to construct a single transducer, then the synthesis problem is *centralized*. Synchronous centralized synthesis from LTL has time complexity doubly exponential in the length of the formula [5], and

---

<sup>1</sup> Compare with existential quantification in *past* time, as is the case in the past fragment of LTL. This causes no problems, because it concerns things past.

polynomial in the number of states. By restricting to a less expressive fragment of LTL, the complexity can be lowered to polynomial in the formula [28]. Asynchronous centralized synthesis does not yield to such a reduction [29]. Partial information games pose a challenge similar to full LTL properties, due to the need for a powerset-like construction [30]. To avoid this route, alternative methods have been developed [31], that use universal co-Büchi automata, instead of determinization, and antichains [32].

If we want to construct several communicating transducers to obtain some collective behavior, then synthesis is called *distributed*. Of major importance in distributed synthesis is who talks to whom, and how much, called the communication architecture. A distributed game with full information is in essence a centralized synthesis problem. Distributed *synchronous* games with partial information are undecidable [33], unless we restrict the communication architecture to avoid information forks [34], or restrict the specifications to limited fragments of LTL [35]. *Bounded synthesis* circumnavigates this intractability by searching for systems with a priori bounded memory [36]. Asynchronous distributed synthesis is undecidable [27].

Besides distributed co-synthesis of fixed transducers, the more general notion of *assume-guarantee* synthesis [37] constructs transducers that can interface with a complete set of other transducers, as described by an assumption property. This is the same viewpoint with the approach proposed here. A difference is that we are interested to synthesize temporal properties with quantification (liveness), instead of directly transducers. Besides, note that distributed in the literature means constructing multiple transducers. In contrast, we are interested in distributed also in the sense that the modules will be synthesized *separately*. Thus, in the problem we consider, distributed synthesis with full information does not reduce to centralized synthesis.

Another body of work relevant to our effort is the construction of assumptions that make an unrealizable problem realizable. The methods originally developed for this purpose have been targeted at compositional verification, and use the  $L^*$  algorithm for learning deterministic automata [38], and implemented also symbolically [39]. Later work addressed synthesis, with the theory for a solution proposed in [40], on which our work builds. This approach separates the construction of assumptions into safety and liveness. The safety assumption is obtained by property *closure*, which also plays a key role in the composition theorem presented in [16].

Methods that use opponent strategies to refine the assumptions of a generalized Streett(1) specification, searching over syntactic patterns were proposed in [41, 42]. The syntactic approach of [42] was used in [43] to refine assume-guarantee specifications of coupled modules. However, that work cannot handle circularly connected modules, thus neither circular liveness dependencies. Other approaches aim at identifying the root causes of unrealizability in demanding guarantees [44]. A comprehensive survey can be found in [45].

## 4 Proposed approach

This report proposes a method for constructing assume-guarantee specifications for a set of modules. The resulting specifications must be *realizable* [46], i.e., for each module, there should exist a transducer that implements its specification. The required behavior of each module is described by a contract over a set of variables that can change values in time. We choose linear temporal logic (LTL) [11] to describe contract specifications. The specification of a module includes a partition of variables into inputs (uncontrolled by the module), and outputs (controlled by the module), as well as the primitive goals that the module must achieve, but no assumptions yet. These goals form an overall objective that the resulting contracts should satisfy. At this stage, the goals may be insufficient to ensure cooperation of the modules with each other. In other words, the specifier defines guarantees for each module, and the proposed method introduces assumptions that ensure realizability. Note that each property introduced as an assumption in the contract of some module, will also become a guarantee in the contract of some another module.

We assume that, if we were to construct a single transducer that controls the variables of all modules, then such a transducer exists. This requires that the conjoined goals be satisfiable. If the goals are unsatisfiable, then the algorithm diagnoses so, but cannot resolve the conflicts. Such a resolution would be arbitrary, because it alters the design intent that a human defined, so it should be performed by a human.

As noted in Section 3, synthesis from LTL specifications is intractable. For this reason, we restrict our effort to an LTL fragment that is less expressive, but still practically useful, while allowing synthesis in time polynomial in the number of states, and in the size of the specification formulae. The selected fragment is

known as generalized reactivity of rank 1, GR(1) [28], and describes generalized Streett games with one pair, comprised of a persistence and an acceptance property. This restriction aims at making efficient the synthesis phase, after the contracts have been constructed, as well as the construction of the contracts themselves. It is a trade-off between expressive power and complexity. It corresponds to considering the bottom level in the Borel hierarchy of sets of behaviors, as sequences [47].

We model a composite system as a game with multiple players, each representing a module. In Section 6, the winning set is computed for the case of a centralized transducer, also known as the *cooperative* winning set. This is used as a safety assumption for all modules, in order to prevent any module from forcing the system to exit the set from where another module has a winning strategy.

For each module, and each recurrence goal, the winning set in the game with that goal is computed in Section 8. If the winning region is smaller than the cooperative winning set, then weak liveness assumptions are computed for the other players, until reaching a new fixpoint. These assumptions must be *unconditionally realizable*, to prevent trivial realizability of a particular game.

The predicates in the resulting contract are represented symbolically, as binary decision diagrams (BDDs). This is in contrast with syntactic approaches for constructing assumptions. Syntactic approaches are restricted to the subset of specifications producible by the chosen grammar template, thus are incomplete. In contrast, our semantic approach always obtains a solution, if one exists. The trade-off is that the resulting properties do not have a syntactic form digestible by humans. The semantic contracts that we construct correspond to a view of contracts as an intermediate result, to be consumed by synthesis algorithms that will construct each individual module, potentially after a refinement of the contract by addition of local, internal, requirements.

As discussed in Section 2, a challenge in modular specification is reasoning about liveness. An assume-guarantee contract is intended to remain as declarative as possible. However, there are behaviors that, if specified declaratively, lead to cyclic dependency of liveness properties. For this reason, we structure the constructed specifications in a way that avoids circular dependencies of liveness requirements. This requires imposing a sequencing order on the liveness properties involved. In verification, the implementation itself is used as reference for enforcing this sequencing. In temporal logic, it is possible to achieve this purpose by explicitly introducing auxiliary variables. We avoid introducing such additional variables, because they increase the state space and can be regarded as a limited form of synthesis. Instead, we alter the specification structure, from flat to nested. For each liveness goal, nesting is introduced in the form of a stack of games. Each game in the stack has a reachability objective, and separate assumptions. Winning one of these games leads higher in the stack, until the top is reached. The top game can be won directly, and leads to the recurrence goal. The reliance on safety is in that each subgame is defined on a subset of the states. In this way, the composite system is prevented from regressing backwards, to a previous game, and progress towards the recurrence goal is ensured.

## 5 Preliminaries

### 5.1 Turn-based synchronous games

We consider turn-based synchronous games with two players [25, 26]. The results can be extended to multiple players. We do not consider concurrent games, because they are not determined, and a strategy can require an infinite amount of memory [20].

The situation in a game is represented by a number of variables. An assignment to these variables is called the *state* of the game. The game evolves by a sequence of state changes. If, in each state change, only a single player changes its own state, then the game is called *turn-based* [25]. It is synchronous if the players take turns in a fixed order.

In a game with two players, we will refer to the two players by the indices 0 and 1. In some cases, we will also use the notation of indexing the players with the letters *e* (environment) and *s* (system), instead of numbers. This is more readable when we discuss operations that consider one player as the system of interest, and lumps the remaining players as the environment of that player.

The state comprises of variables in the set  $\mathcal{V}$ . Each player can read all the variables, i.e., it has full information. Each player can write only those variables that she owns, with the exception of variable *i*.

Player  $j$  owns the set  $\mathcal{X}_j$  of variables. In addition, each player increments the auxiliary variable  $i$ , used to track turns. So,  $\mathcal{V} = \{i\} \cup \bigcup_{i=0}^{n-1} \mathcal{X}_i$ .

By  $x_i$  we will denote both the tuple of symbols in  $\mathcal{X}_i$ , as well as a tuple of values assigned to those symbols. In its own turns, player  $i$  chooses a next assignment  $x'_i \in \mathcal{X}'_i$ . The set of all states  $2^{\mathcal{V}}$  is denoted by  $S$ . For a set of variable symbols  $X$ , define the set of assignments  $\llbracket \top \rrbracket_X \triangleq 2^X$ . A predicate  $f$  indicates a set

$$\llbracket f \rrbracket_X \triangleq \{u \in 2^X \mid f(u)\}. \quad (1)$$

The game can be represented by a *game graph*, with nodes partitioned between the two players. At each node, only one of the two players moves. The game graph is bipartite, because the game is turn-based. Note that bipartiteness is necessary<sup>2</sup> later, for switching between players when constructing a nested game. The player that moves first can be selected later, after computing the winning sets, when constructing the transducers.

Each node in the game graph is represented by a tuple  $(x, i)$ , where:

- $x_i \in 2^{\mathcal{X}_i}$  is an assignment for the variables owned by player  $i$ , and the aggregate state  $x = (x_0, x_1, \dots, x_{n-1})$ .
- $i \in I \triangleq \mathbb{N}_{<n}$  is an index that signifies the player that takes a turn from  $(x, i)$ .

The transition relation of player  $i$  is  $\hat{\rho}_i(x, x'_i)$ , where  $\hat{\rho}_i$  is an action formula (a Boolean formula over primed and unprimed variables) [48]. Player  $i$  moves from the node  $(x, i)$ , by assigning values to variables in  $x_i$ . Let  $\bar{x}_j$  denote (either a tuple of, or an assignment to) variables in  $\bigcup_{i=0, i \neq j}^{n-1} \mathcal{X}_i$ . We will try to minimize use of the term state, because it can be confusing.

**Remark 1.** *A (synchronous) interleaving representation [16] is used here for the game, because it is symmetric and emphasizes the turn-based semantics. As observed in [16], an interleaving representation can be easier to reason about. In the literature about GR(1) games, typically a non-interleaving representation is used. In a non-interleaving representation, the combination of primed and unprimed variables captures whose turn it is to play (the role served by the integer variable  $i$ ). In that representation, player 0 moves from a valuation of  $(x_0, x_1)$ , and player 1 moves from  $(x'_0, x_1)$ . Note that the scheduling variable  $i$  is shared-write by all players.*

## 5.2 Integrals

In this section, we consider preimage functions induced by the transition relations  $\rho_i$ . These functions result from different quantification of the variables. Depending on the source and target set, several variants can be defined. We will refer to predicates and the sets they represent interchangeably.

**Definition 2** (Predecessors). *Given a predicate  $F$  over  $\mathcal{V}$ , the existential predecessors of  $F$  are those nodes, from where the set  $\llbracket F \rrbracket_{\mathcal{V}}$  can be reached with one transition in the game graph,*

$$\text{Pre}_j(F) \triangleq \lambda x. \lambda i. (i = j) \wedge \exists x'_j. \hat{\rho}_j(x, x'_j) \wedge F|_{x'_j/x_j}(\bar{x}_j, x'_j, j \oplus_n 1), \quad (2)$$

where  $j \oplus_n k \triangleq (j + k) \bmod n$ . Denote  $\text{Pre}(F) \triangleq \bigvee_{j \in I} \text{Pre}_j(F)$  the predecessors resulting from moves by all players.

The semantics of the least fixpoint operator  $\mu X. f(X)$  is defined as

$$\llbracket X_k \rrbracket_M^{\mathcal{E}} \triangleq \begin{cases} \emptyset, & k = 0 \\ \llbracket f(X) \rrbracket_M^{\mathcal{E}[X \leftarrow \llbracket X_{k-1} \rrbracket_M]}, & k > 0 \end{cases} \quad \llbracket \mu X. f(X) \rrbracket_M^{\mathcal{E}} \triangleq \bigcup_{k=0}^{\infty} X_k, \quad (3)$$

where  $M$  a set of variables, and  $\mathcal{E} : \{X, \dots\} \rightarrow \llbracket \top \rrbracket_M$  is an assignment that keeps track of the fixpoint iteration. The notation  $\mathcal{E}[X \leftarrow \llbracket h \rrbracket_M]$  denotes the modification of  $\mathcal{E}$  to assign the set  $\llbracket h \rrbracket_M$  to variable  $X$ .

<sup>2</sup> Any game graph can be converted to a bipartite one, by introducing intermediate nodes.

**Definition 3** (Iterated predecessors). *The iterated predecessor relation yields the nodes that can reach the set  $\llbracket F \rrbracket$  under some behavior of the players, or are already in the set  $\llbracket F \rrbracket$ , i.e.,*

$$\text{Pre}^*(F) \triangleq \mu X. F \vee \text{Pre}(X). \quad (4)$$

Note that the set  $\llbracket \text{Pre}^*(F) \rrbracket$  contains the nodes from where the players can *cooperate* to reach the set  $\llbracket F \rrbracket$ . Where clear from the context, we will call both  $\text{Pre}$  and  $\text{Pre}^*$  predecessor sets.

**Definition 4** (Controllable predecessors). *The controllable predecessors of  $F$  for player  $j$  are those nodes from where player  $j$  can force a visit to the set  $\llbracket F \rrbracket_{\mathcal{V}}$  in the next logic time step, irrespective of how the other players move, i.e.,*

$$\text{CPre}_j(F) \triangleq \lambda x. \lambda i. \neg^{i \neq j} \exists x'_i. \hat{\rho}_i(x, x'_i) \wedge \neg^{i \neq j} F|_{x'_i/x_i}(\bar{x}_i, x'_i, i \oplus_n 1). \quad (5)$$

For example, for player  $j = 0$ , it is

$$\begin{aligned} \text{CPre}_0(F) &= \lambda x. \lambda i. ((i = 0) \wedge \neg^{0 \neq 0} \exists x'_0. \hat{\rho}_0(x, x'_0) \wedge \neg^{0 \neq 0} F|_{x'_0/x_0}(\bar{x}_0, x'_0, 0 \oplus_2 1)) \vee \\ &\quad ((i = 1) \wedge \neg^{1 \neq 0} \exists x'_1. \hat{\rho}_1(x, x'_1) \wedge \neg^{1 \neq 0} F|_{x'_1/x_1}(\bar{x}_1, x'_1, 1 \oplus_2 1)) \\ &= \lambda x. \lambda i. ((i = 0) \wedge \neg^0 \exists x'_0. \hat{\rho}_0(x, x'_0) \wedge \neg^0 F|_{x'_0/x_0}(\bar{x}_0, x'_0, 1)) \vee \\ &\quad ((i = 1) \wedge \neg^1 \exists x'_1. \hat{\rho}_1(x, x'_1) \wedge \neg^1 F|_{x'_1/x_1}(\bar{x}_1, x'_1, 0)) \\ &= \lambda x. \lambda i. ((i = 0) \wedge \exists x'_0. \hat{\rho}_0(x, x'_0) \wedge F|_{x'_0/x_0}(\bar{x}_0, x'_0, 1)) \vee \\ &\quad ((i = 1) \wedge \forall x'_1. \hat{\rho}_1(x, x'_1) \rightarrow F|_{x'_1/x_1}(\bar{x}_1, x'_1, 0)). \end{aligned} \quad (6)$$

As defined here, the operator  $\text{CPre}$  is the predicate version of that defined in [1]. An attractor contains nodes from where player  $j$  can force its way to the set  $\llbracket F \rrbracket$ .

**Definition 5** (Attractor). *The attractor  $\text{Attr}_j(F)$  for player  $j$  is the set of all nodes, from where the system can force a future visit to the set  $\llbracket F \rrbracket$ , or is already in  $\llbracket F \rrbracket$ ,*

$$\text{Attr}_j(F) \triangleq \mu X. F \vee \text{CPre}_j(X). \quad (7)$$

As alternative notation, let  $\text{CPre}_j^*(F) \triangleq \text{Attr}_j(F)$ .

### 5.3 Linear temporal logic

Linear temporal logic [11] with past [49] is an extension of Boolean logic used to reason about temporal modalities over sequences. The temporal operators:

- next  $\bigcirc$ ,
- previous  $\ominus$ ,
- until  $\mathcal{U}$ , and
- since  $\mathcal{S}$

suffice to define the other operators [11, 50]. Let  $AP$  be a set of propositional variable symbols, with values in  $\mathbb{B} \triangleq \{\perp, \top\}$ . A well-formed LTL formula is inductively defined by

$$\begin{aligned} \varphi ::= & p \mid \neg\varphi \mid p \wedge p \\ & \mid \bigcirc\varphi \mid \varphi \mathcal{U} \varphi \\ & \mid \ominus\varphi \mid \varphi \mathcal{S} \varphi. \end{aligned} \quad (8)$$

It is modeled by a sequence (word) of variable assignments  $w : \mathbb{N} \rightarrow \mathbb{B}^{AP}$ . Here, we define informally the operators that we will use. The formula  $\Box p$  holds if  $p$  is forever true,  $\Diamond p$  if  $p$  becomes true in some non-past time. The *weak* previous formula  $\ominus p \triangleq \neg \ominus \neg p$  is true if a previous time step does not exist, or  $p$  is true in the previous time step. In contrast,  $\ominus p$  is true if a previous time step does exist, and  $p$  is true then.

## 5.4 Interleaving representation of a Streett(1) game

We will use an interleaving representation [16], with the notation defined in Section 5.1. In an interleaving representation of a turn-based game, a single player moves in each logic time step. In a synchronous game, players move in a fixed order. This order will be enforced by using the auxiliary variable  $i$ , as index of the player that should move in the current logic time step.

In a game, each player is assigned a property to realize. A game structure collects the initial conditions, actions, and liveness goals of each player. Two-player game structures in a non-interleaving representation are defined in [51]. The property to be realized by the player of interest is defined there accordingly.

In an interleaving representation, a generalized reactivity(1) property [28] to be realized by player  $j$  can be described as follows. Define

$$\begin{aligned}\rho_j(x, x'_j, i) &\triangleq \text{ite}(i \neq j, x'_j = x_j, \hat{\rho}_i \wedge (i' = i \oplus_n 1)) \\ \bar{\rho}_j(x, \bar{x}'_j, i) &\triangleq \bigwedge_{k \in J \setminus \{j\}} \rho_k(x, x'_k, i)\end{aligned}\quad (9)$$

In a two-player game, it is

$$\bar{\rho}_j(x, \bar{x}'_j, i) = \bigwedge_{k \in \{0,1\} \setminus \{j\}} \rho_k(x, \bar{x}'_k, i) = \rho_{1-j}(x, \bar{x}'_{1-j}, i). \quad (10)$$

**Definition 6** (Generalized reactivity(1)). *Assume that, for  $i \in I$ , each  $\rho_i(x, x'_i)$  is an action formula, as defined in Section 5.1. Let  $j \in I$  be the index of a player. Assume that, for  $k \in I_P \subset \mathbb{N}$ , each  $P_{j,k}(x, i)$  is an assertion (a Boolean formula over unprimed variables), and similarly for  $R_{j,r}(x, i)$ . Then, the LTL formula*

$$\varphi_{G,j} \triangleq \begin{aligned} &\bigwedge \square ((\ominus \square \bar{\rho}_j) \rightarrow \rho_j) \\ &\bigwedge (\square \bar{\rho}_j \wedge \bigwedge_k \square \diamond \neg P_{j,k}) \rightarrow \bigwedge_r \square \diamond R_{j,r}. \end{aligned} \quad (11)$$

describes a  $GR(1)$  property for player  $j$ .

For symmetry, the initial conditions have been omitted above. Initial conditions require selecting the player that moves first, and their consideration can be delayed until the phase of constructing a winning strategy. Observe that the action  $\rho_i$  can depend on the variables  $x, x'_i$ , but is independent of the variables  $\bar{x}'_i$ .

As a shorthand for the above, we define strict implication between two temporal logic formulae in a (synchronous) interleaving representation of a game.

**Definition 7** (Strict implication). *Let  $\rho_e, \rho_s, P_k, R_r$  be actions (or assertions). Define the strict implication operator  $\overset{\text{sr}}{\triangleright}$  as*

$$\underbrace{(\square \rho_e \wedge \bigwedge_k \square \diamond \neg P_k)}_{\text{assumption}} \overset{\text{sr}}{\triangleright} \underbrace{(\square \rho_s \wedge \bigwedge_r \square \diamond R_r)}_{\text{guarantee}} \triangleq \begin{aligned} &\bigwedge \square ((\ominus \square \rho_e) \rightarrow \rho_s) \\ &\bigwedge (\square \rho_e \wedge \bigwedge_k \square \diamond \neg P_k) \rightarrow \bigwedge_r \square \diamond R_r. \end{aligned} \quad (12)$$

The antecedent constrains the other players, and the consequent the player under consideration. For a non-interleaving representation, Strict implication was defined in [51]. Unless the action-fairness pairs are machine closed, and the actions are complete, the strict implication operator  $\overset{\text{sr}}{\triangleright}$  differs from the TLA while-plus operator  $\overset{\pm}{\triangleright}$  [16, 52].

With Definition 7, we can rewrite Definition 6 using strict implication

$$\varphi_{G,j} = \begin{aligned} &\bigwedge \square \bar{\rho}_j \\ &\bigwedge \bigwedge_k \square \diamond \neg P_{j,k} \end{aligned} \overset{\text{sr}}{\triangleright} \begin{aligned} &\bigwedge \square \rho_j \\ &\bigwedge \bigwedge_r \square \diamond R_{j,r}. \end{aligned} \quad (13)$$

## 6 Property closure

### 6.1 Cooperative winning set

In the following, we will present an algorithm for computing pairs of specifications that allow players to cooperate. For that purpose, some definitions are needed. Let  $\Sigma$  be a suitable alphabet, for example,  $\Sigma = 2^{\mathcal{V}}$ . The set  $\Sigma^*$  denotes finite sequences of elements in  $\Sigma$ . The set  $\Sigma^\omega$  denotes infinite sequences of elements in  $\Sigma$ . The elements of a sequence are indexed by integers, starting at 0. For a sequence  $w \in \Sigma^\omega$ , the subsequence that starts at element  $i$  and ends at element  $j$  (inclusive) is denoted by  $w[i \dots j]$ .

**Definition 8** ([53, 16]). *A behavior or property  $P \subseteq \Sigma^\omega$  is a set of infinite sequences.*

**Definition 9** (Prefix set [40]). *The prefix set of a property  $P$  is defined as*

$$\text{Pref}(P) \triangleq \{\sigma \in \Sigma^* \mid \exists w \in P. \sigma = w[0 \dots |\sigma| - 1]\}. \quad (14)$$

**Definition 10** (Limit set [40]). *Given a property  $P \subseteq \Sigma^*$ , the set of limits of property  $P$  in property  $Q$  is defined as*

$$\text{Safety}_Q(P) \triangleq \{w \in Q \mid \forall k \in \mathbb{N}. w[0 \dots k] \in P\}. \quad (15)$$

*If the subscript  $Q$  is omitted, then  $Q = \Sigma^\omega$ , i.e.,*

$$\text{Safety}(P) \triangleq \text{Safety}_{\Sigma^\omega}(P). \quad (16)$$

**Definition 11** (Relative closure [54, 16]). *The closure of a property  $P \subseteq \Sigma^\omega$  with respect to another property  $Q \subseteq \Sigma^\omega$  is defined as*

$$\mathcal{C}_Q(P) \triangleq \text{Safety}_Q(\text{Pref}(P)) = \{w \in Q \mid \forall k \in \mathbb{N}. \exists \sigma \in P. w[0 \dots k] = \sigma[0 \dots k]\} \quad (17)$$

*If the subscript  $Q$  is omitted, then  $Q = \Sigma^\omega$ , i.e.,*

$$\mathcal{C}(P) \triangleq \mathcal{C}_{\Sigma^\omega}(P). \quad (18)$$

*For brevity, define  $\bar{P} \triangleq \mathcal{C}(P)$ .*

The definition  $\mathcal{C}_{\Sigma^\omega}(P)$  corresponds to  $\mathcal{C}(P)$  in [16]. The closure of a property is with respect to the topology induced by the metric that measures similarity by the length of the longest common prefix between two sequences.

**Definition 12.** *Assume that  $P \subseteq \Sigma^\omega \cup \Sigma^*$  is a property. Define the set of letters that appear in any word in property  $P$  as*

$$\text{States}(P) \triangleq \{s \in \Sigma \mid \exists w \in P. \exists k \in \mathbb{N}. s = w[k]\}. \quad (19)$$

The definition of closure implies that  $\text{States}(P) = \text{States}(\mathcal{C}(P))$ .

**Definition 13** ([55, 40]). *Assume that  $F \subseteq \Sigma$  is a set of letters, and  $\hat{\rho}_j, j \in I$  a collection of actions (transition relations). Then, the safe words are those in the set*

$$\text{Safe}(F) \triangleq \left\{ w \in \Sigma^\omega \mid \forall k \in \mathbb{N}. w[k] \in F \wedge \bigwedge_{j \in I} \left( (w[k]|_i = j) \rightarrow \begin{array}{l} \wedge w[k+1]|_i = j \oplus_n 1 \\ \wedge w[k]|_{\bar{x}_j} = w[k+1]|_{\bar{x}_j} \\ \wedge \hat{\rho}_j(w[k], w[k+1]) \end{array} \right) \right\}. \quad (20)$$

The map  $\text{States}$  projects a sequence on the state space. In the opposite direction, the map  $\text{Safe}$  yields the largest invariant subset of a given safe set, under the transition relations.

**Definition 14** ([40]). *The cooperative winning set is the set of nodes in the game graph, from where the players can cooperate to satisfy their objectives. In a turn-based synchronous game with  $n$  players, with objectives  $\varphi_j, j \in I$  (that include the transition relations  $\rho_j$ ), it is*

$$\text{Coop} \left( \bigwedge_{j \in I} \varphi_j \right) \triangleq \left\{ u \in \Sigma = 2^{\mathcal{V}} \mid \exists w \in \mathcal{L} \left( \bigwedge_{j \in I} \varphi_j \right). w[0] = u \right\}. \quad (21)$$

In other words, the cooperative winning set is the set of nodes from where a centralized controller has a winning strategy. If the objectives  $\varphi_j$  do not include initial conditions<sup>3</sup> (i.e., are tail-closed), then

$$\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right) = \text{States}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right). \quad (22)$$

The closure of the conjoined specifications is equal to the safe words defined by the cooperative winning set. This follows from

$$\text{States}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right) = \text{States}\left(\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)\right), \quad (23)$$

which implies that

$$\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right) = \text{States}\left(\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)\right) \implies \text{Safe}\left(\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right)\right) = \text{Safe}\left(\text{States}\left(\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)\right)\right). \quad (24)$$

Observing that each  $\varphi_j$  includes  $\Box\rho_j$ , it follows that  $\text{Safe}\left(\text{States}\left(\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)\right)\right) = \mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)$ , therefore

$$\text{Safe}\left(\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right)\right) = \mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right). \quad (25)$$

Define the recurrence formulae  $\text{WF}_j \triangleq \bigwedge_r \Box \diamond G_{j,r}$ , for  $j \in I$ . For each player  $j$ , assume that it has as objective property described by the formula

$$\varphi_j \triangleq \Box\rho_j \wedge \text{WF}_j. \quad (26)$$

The property  $\varphi$  is in the GR(1) fragment of LTL, so it defines a generalized Streett game of rank 1 (unconditional, i.e., w/o assumptions). The objectives  $\varphi_j$  may be unrealizable. For each objective  $\varphi_j$ , we are interested in constructing assumptions that make it realizable. These assumptions will become objectives for the other agents. Note that, at this stage there are no persistence objectives (i.e., no recurrence assumptions yet).

The cooperative winning set can be computed by the fixpoint formula

$$\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right) = \nu \begin{bmatrix} Z_0 \\ Z_1 \\ \vdots \\ Z_N \end{bmatrix} \cdot \begin{bmatrix} \text{Pre}^*(G_{0,0} \wedge \text{Pre}(Z_1)) \\ \text{Pre}^*(G_{0,1} \wedge \text{Pre}(Z_2)) \\ \vdots \\ \text{Pre}^*(G_{n-1, N_{n-1}-1} \wedge \text{Pre}(Z_0)) \end{bmatrix} = \nu Z. \bigwedge_{j=0}^{n-1} \bigwedge_{r=0}^{N_j-1} \text{Pre}^*(G_{j,r} \wedge \text{Pre}(Z)). \quad (27)$$

The above computation of the fixpoint involves the recurrence goals of all players. The aim of decomposing a large system is to modularize the design effort. This motivates parallelizing the above fixpoint computation.

A slightly different arrangement is also possible. The goals of each player can be grouped into a vectorized subformula, as follows

$$\nu Z. \bigwedge_{j=0}^{n-1} \nu Z_j. Z \wedge \bigwedge_{r=0}^{N_j-1} \text{Pre}^*(G_{j,r} \wedge \text{Pre}(Z_j)) \quad (28)$$

This is expected to increase the sharing of subformulae, because of the overlap of support sets among objectives of a single player. It is motivated, in part, by the observations of Section 6.2. In Section 6.2, it is shown that the outer fixpoint will be delayed from converging only by states that are live for each objective separately, but not for all objectives jointly. By increasing coupling between goals, the rate of convergence improves, while still parallelizing the computation, with a granularity at the level of players, instead of individual recurrence goals. Regarding the variable order, postponing the interaction of BDDs for iterates associated with goals of different players is expected to reduce the coupling between variables, and thus reduce the cost and improve the effectiveness of BDD variable reordering.

<sup>3</sup> When computing the winning set in a game graph, initial conditions are neglected. They are accounted for later, during construction of a transducer.

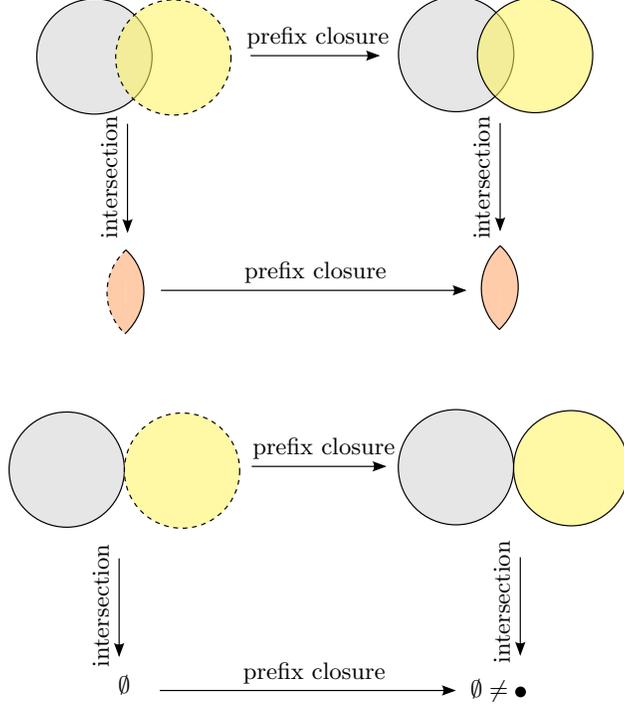


Figure 1: In general, the closure of intersection differs from the intersection of closures.

## 6.2 Computing the closure

In this section, we prove that

$$\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right) = \nu Z. \bigwedge_{j=0}^{n-1} \nu Z_j. Z \wedge \bigwedge_{r=0}^{N_j-1} \text{Pre}^*(G_{j,r} \wedge \text{Pre}(Z_j)). \quad (29)$$

This equality is a consequence of results about vectorized  $\mu$ -calculus [56]. Nonetheless, a direct proof is presented below, that gives a better picture of how the sets change during the iteration.

From Section 6.1, recall that  $\text{Safe}\left(\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right)\right) = \mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)$ . In other words, given a conjunction of properties  $\mathcal{L}(\varphi_0 \wedge \varphi_1 \wedge \dots \wedge \varphi_{n-1})$ , its closure  $\mathcal{C}(\mathcal{L}(\varphi_0 \wedge \varphi_1 \wedge \dots \wedge \varphi_{n-1}))$  is equal to the infinite words generated by the restriction of the transition relation to the cooperative winning set. For this reason, we refer to the closure  $\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)$  and the cooperative winning set  $\text{Coop}\left(\bigwedge_{j \in I} \varphi_j\right)$  interchangeably.

From Eq. (27), it suffices to prove that

$$\nu Z. \bigwedge_{j=0}^{n-1} \nu Z_j. Z \wedge \bigwedge_{r=0}^{N_j-1} \text{Pre}^*(G_{j,r} \wedge \text{Pre}(Z_j)) = \nu Z. \bigwedge_{j=0}^{n-1} \bigwedge_{r=0}^{N_j-1} \text{Pre}^*(G_{j,r} \wedge \text{Pre}(Z)). \quad (30)$$

This is equivalent to proving that  $\mathcal{C}\left(\bigcap_{j \in I} \mathcal{L}(\varphi_j)\right)$  is equal to the fixpoint iteration that alternates between taking closure and intersection.

**Proposition 15.** *For the properties defined by the formulae  $\{\varphi_i\}_{i < n}$ , the closure of the intersection is a subset of the intersection of closures, i.e.,  $\mathcal{C}\left(\bigcap_{i=0}^{n-1} \mathcal{L}(\varphi_i)\right) \subseteq \bigcap_{i=0}^{n-1} \mathcal{C}(\mathcal{L}(\varphi_i))$ .*

The obstruction in parallelizing the computation is that, in general, the opposite containment does *not* hold. In that case, the difference arises due to words on the boundary of some property, as proved by the following.

**Proposition 16.** *Assume that the closure of intersection differs from the intersection of closures, i.e.,  $\mathcal{C}(\bigcap_{i=0}^{n-1} \mathcal{L}(\varphi_i)) \neq \bigcap_{i=0}^{n-1} \mathcal{C}(\mathcal{L}(\varphi_i))$ . Then, for each word  $w \in \left(\bigcap_{i=0}^{n-1} \mathcal{C}(\mathcal{L}(\varphi_i))\right) \setminus \mathcal{C}(\bigcap_{i=0}^{n-1} \mathcal{L}(\varphi_i))$ , there exists some property  $\mathcal{L}(\varphi_j)$ , such that  $w$  is on the excluded boundary of property  $\mathcal{L}(\varphi_j)$ , i.e.,  $w \in \partial\mathcal{L}(\varphi_j) \setminus \mathcal{L}(\varphi_j)$ .*

In any ball around a word  $w$  in the boundary  $\partial\mathcal{L}(\varphi_j)$ , there exists some word  $z$  in the property  $\mathcal{L}(\varphi_j)$ . It follows that, for any prefix  $p$  of word  $w$ , there exists some word  $z \in \mathcal{L}(\varphi_j)$  that has the prefix  $p$ . As a result, the word  $w$  is safe with respect to  $\mathcal{L}(\varphi_j)$ , but not live.

Next, we define the iteration that corresponds to Eq. (29), and prove that it converges to the cooperative winning set.

**Definition 17.** *Define  $P_j \triangleq \mathcal{L}(\varphi_j)$ . Initialize  $Q^0 \triangleq \Sigma^\omega$ , and iterate for  $k \in \mathbb{N}$*

$$\begin{aligned} R_j^k &\triangleq \mathcal{C}(Q^k \cap P_j), \\ Q^{k+1} &\triangleq \bigcap_{j \in I} R_j^k. \end{aligned} \tag{31}$$

We are interested in proving that the iteration of Definition 17 reaches as fixpoint the set  $\mathcal{C}(\bigcap_{j \in I} P_j)$ . For this purpose, we will prove that

- $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^k$  remains invariant (Proposition 18), and
- if the current iterate  $Q^k$  differs from  $\mathcal{C}(\bigcap_{j \in I} P_j)$ , then  $|\text{States}(Q^{k+1})| < |\text{States}(Q^k)|$  (Proposition 19).

**Proposition 18** (Invariant). *For all  $k \in \mathbb{N}$ ,  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^k$ .*

*Proof.* By induction:

**Case  $k = 0$**  It is  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq \Sigma^\omega = Q_0$ .

**Case  $k > 0$**  Assume that  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^k$ . We will prove that  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^{k+1}$ . By definition of the iterates

$$Q^{k+1} = \bigcap_{j \in I} R_j^k = \bigcap_{j \in I} \mathcal{C}(Q^k \cap P_j). \tag{32}$$

By the induction hypothesis,

$$\mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq Q^k \implies P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq Q^k \cap P_j \implies \mathcal{C}\left(P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right)\right) \subseteq \mathcal{C}(Q^k \cap P_j). \tag{33}$$

Therefore, it suffices to prove that  $\mathcal{C}(\bigcap_{i \in I} P_i) \subseteq \mathcal{C}(P_j \cap \mathcal{C}(\bigcap_{i \in I} P_i))$ . It is

$$\bigcap_{i \in I} P_i \subseteq P_j \implies P_j \cap \bigcap_{i \in I} P_i = \bigcap_{i \in I} P_i, \tag{34}$$

so

$$\begin{aligned} P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right) &= P_j \cap \left(\left(\bigcap_{i \in I} P_i\right) \cup \left(\partial \bigcap_{i \in I} P_i\right)\right) = \left(P_j \cap \bigcap_{i \in I} P_i\right) \cup \left(P_j \cap \partial \bigcap_{i \in I} P_i\right) \\ &= \bigcap_{i \in I} P_i \cup \left(P_j \cap \partial \bigcap_{i \in I} P_i\right) \implies \\ \mathcal{C}\left(P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right)\right) &= \mathcal{C}\left(\bigcap_{i \in I} P_i \cup \left(P_j \cap \partial \bigcap_{i \in I} P_i\right)\right) = \mathcal{C}\left(\bigcap_{i \in I} P_i\right) \cup \mathcal{C}\left(P_j \cap \partial \bigcap_{i \in I} P_i\right) \implies \\ &\mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq \mathcal{C}\left(P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right)\right) \end{aligned} \tag{35}$$

By the above result, Eq. (33), and the induction hypothesis  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^k$ , it follows that

$$\forall j \in I. \mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq \mathcal{C}\left(P_j \cap \mathcal{C}\left(\bigcap_{i \in I} P_i\right)\right) \subseteq \mathcal{C}(Q^k \cap P_j) \implies \mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq \bigcap_{j \in I} \mathcal{C}(Q^k \cap P_j) \tag{36}$$

Using Eq. (32), it follows that

$$\mathcal{C}\left(\bigcap_{i \in I} P_i\right) \subseteq \bigcap_{j \in I} \mathcal{C}(Q^k \cap P_j) = Q^{k+1}. \quad (37)$$

This is the inductive claim.  $\square$

**Proposition 19** (Variant). *If  $\mathcal{C}(\bigcap_{j \in I} P_j) \neq Q^k$ , then  $|\text{States}(Q^{k+1})| < |\text{States}(Q^k)|$ .*

*Proof.* By definition of the iterates,  $Q^{k+1} = \bigcap_{j \in I} \mathcal{C}(Q^k \cap P_j)$ . The closure  $\mathcal{C}(Q^0) = \mathcal{C}(\Sigma^\omega) = \Sigma^\omega = Q^0$ , so the set  $Q^0$  is closed. As the intersection of closed sets, the set  $Q^k, k > 0$  is closed. It is

$$Q^k \cap P_j \subseteq Q^k \implies \mathcal{C}(Q^k \cap P_j) \subseteq \mathcal{C}(Q^k) = Q^k \implies Q^{k+1} = \bigcap_{j \in I} \mathcal{C}(Q^k \cap P_j) \subseteq Q^k. \quad (38)$$

It remains to prove that  $Q^{k+1} \neq Q^k$ . We will show that taking the closures  $\mathcal{C}(Q^k \cap P_j)$  will yield at least one set  $\text{States}(R_j^k) \subsetneq \text{States}(Q^k)$ . By Proposition 18,  $\mathcal{C}(\bigcap_{j \in I} P_j) \subseteq Q^k$ , and by hypothesis they are not equal. So, the difference  $K \triangleq Q^k \setminus \mathcal{C}(\bigcap_{j \in I} P_j)$  is non-empty. By induction, the containment  $Q^{k+1} \subseteq Q^k$  implies that, for any  $k > 0$ , it is  $Q^k \subseteq Q^1 = \bigcap_{j \in I} \mathcal{C}(P_j)$ . So,  $K \subseteq \bigcap_{j \in I} \mathcal{C}(P_j)$ . This result is analogous to Proposition 16, but for an arbitrary iteration along the computation.

Consider any word  $w \in K$ . By the previous,  $w \in \bigcap_{j \in I} \mathcal{C}(P_j)$ , and  $w \notin \mathcal{C}(\bigcap_{j \in I} P_j)$ . The game graph is finite, so, by the pigeonhole principle, the word  $w$  has a finite prefix and a finite cycle as suffix. Denote by  $M$  the non-empty set of nodes in the suffix. The word  $w \in \bigcap_{j \in I} \mathcal{C}(P_j)$ , so, from each node in  $M$ , for each  $j \in I$ , a strongly connected component (SCC) that intersects all recurrence sets of  $P_j$  is reachable. The word  $w$  is not in  $\mathcal{C}(\bigcap_{j \in I} P_j)$ . So an SCC that intersects the recurrence sets of all properties is *not* reachable from any node in  $M$ .

Define  $S_j$  the SCC that intersects the recurrence sets of  $P_j$  and is reachable from  $M$ . The set  $R_j^k = \mathcal{C}(Q^k \cap P_j)$ , so nodes in  $\text{States}(R_j^k)$  can reach only the intersection  $S_j \cap \text{States}(Q^k)$ . If  $S_j \cap \text{States}(Q^k) = \emptyset$ , and there are no other SCCs that intersect a  $P_j$  and are reachable from  $M$ , then the nodes in  $M \subseteq \text{States}(Q^k)$  are not in  $\text{States}(R_j^k)$ , and the claim holds.

Suppose that  $S_j \cap \text{States}(Q^k) \neq \emptyset$ . Consider the nodes in  $S_j \cap \text{States}(Q^k)$ . These nodes are in  $\text{States}(Q^k)$ . So the same arguments apply, as those we developed for nodes in  $M$ . This leads to new SCCs, that form a directed acyclic graph (DAG). By finiteness of the game graph, the induction will terminate.

Consider a leaf of the DAG. It is an SCC terminal in  $\text{States}(Q^k)$ , that does not intersect at least one recurrence set, or of at least one property  $P_j$ . (If not, then the SCC would satisfy  $\bigcap_{j \in I} P_j$ . By construction, the leaf SCC is reachable from the nodes in  $M$  (suffix). This implies that from nodes in  $M$ , an SCC satisfying  $\bigcap_{j \in I} P_j$  is reachable. It follows that  $w$  is in  $\mathcal{C}(\bigcap_{j \in I} P_j)$ . This contradicts the definition of  $w$ , as a word not in  $\mathcal{C}(\bigcap_{j \in I} P_j)$ .) Therefore, there is at least one recurrence set of  $P_j$ , which is unreachable from the nodes in the leaf SCC, without exiting the set  $\text{States}(Q^k)$ . It follows that none of these nodes is contained in  $\text{States}(\mathcal{C}(Q^k \cap P_j))$ . These nodes are in  $\text{States}(Q^k)$ , so  $R_j^k = \mathcal{C}(Q^k \cap P_j) \subsetneq Q^k$ .  $\square$

We have proved the following.

**Theorem 20.** *The closure of intersection  $\mathcal{C}(\bigcap_{j \in I} \mathcal{L}(\varphi_j))$  is equal to the fixpoint of the iterated intersection of closures  $Q^{k+1} = \bigcap_{j \in I} \mathcal{C}(Q^k \cap \mathcal{L}(\varphi_j))$ , starting from  $Q^0 = \Sigma^\omega$ .*

After the cooperative winning set  $C = \text{Coop}(\bigwedge_{j \in I} \varphi_j)$  has been computed, each transition relation  $\rho_j$  is restricted to it, by conjoining it with  $\rho_C \triangleq C \wedge C'$ . As proved in [40] for the case of two players, the restriction to the cooperative winning set satisfies two properties:

1. it is not restrictive, because it removes edges from the transition relation  $\rho_i$  of player  $i$ , only if they lead outside the closure with respect to some other player  $\rho_j$ .
2. among all non-restrictive properties, the restriction to the cooperative winning set is minimal, as measured by the cardinality of the edges removed from the game graph.

In addition, the safety property  $\square C$  is added to the assumptions of each agent. The specifications become (redefining Eq. (26) by adding a safety assumption)

$$\varphi_j \triangleq (\square \rho_C) \rightsquigarrow (\square \rho_j \wedge \square \rho_C \wedge \text{WF}_j). \quad (39)$$

## 7 Construction of weak fairness assumptions for a single goal

In this section, we introduce the main elements for the proposed algorithm, for the case of two agents. Let us consider a single recurrence goal. More than one goals are treated by constructing a transducer that cycles through them, and communicating to other players the currently pursued goal. This is described at the end of Section 8. This need for coordination of pursued goals is unavoidable, because, otherwise, livelock arises naturally.

Our objective is to find assumptions that allow covering the cooperative winning set. This problem has been solved for a single agent, and full LTL, in [40]. Here, we are interested in assumptions restricted to the GR(1) fragment, and in multiple players. Recall that in Section 6 we conjoined the transition relations with the requirement that each player stays inside the cooperatively winning set  $C$ , similarly to [40].

Let  $G = \Box\Diamond G_{j=0,r=0}$  be the recurrence goal of interest, of player 0. Player 0 can force a visit to the set  $G$  from any node in the attractor  $A_0 \triangleq \text{Attr}_0(G)$ . But  $A_0$  may not cover the cooperative winning set  $C$ . By the definition of  $C$ , the set  $A_0$  is reachable from  $C \setminus A_0$ . Since nodes in  $C \setminus A_0$  do not belong to  $A_0$ , player 0 cannot force a transition from  $C \setminus A_0$  to  $A_0$ . By determinacy of turn-based synchronous games with full information, player 1 must be able to force such a transition. It follows that the attractor  $\text{Attr}_1(A_0)$  is non-empty. This form of argument is reminiscent of the solution of parity games [57].

We want to construct an *unconditional* assumption that player 0 makes about player 1. Unconditional means that player 1 should be able to realize the assumption, without assuming any liveness property about player 0. If it needed to assume a liveness property about player 0, that would create circularity, causing trivial realizability.

A first attempt could be  $\Box\Diamond(\text{Attr}_1(A_0) \rightarrow A_0)$ . This is insufficient, because player 0 may be able to exit the set  $\text{Attr}_1(A_0)$ , but go to  $\neg A_0$  – not to  $A_0$ . So player 0 must be able to restrict player 1 inside a subset  $K \subseteq \text{Attr}_1(A_0)$ , until player 0 forces its way to  $A_0$ , obliged by an assumption of the form  $\Box\Diamond(K \rightarrow A_0)$ . The inclusion  $K \subseteq \text{Attr}_1(A_0)$  ensures that player 1 cannot trap player 0 inside  $K$ , which would cause trivial realizability. Such an assumption may not exist, a case that is addressed later.

This exist requirement can be formalized by defining<sup>4</sup> the *controlled-escape* subset of a set  $S$ ,

$$\text{Trap}_j(S, E) \triangleq \nu X. E \vee (\text{CPre}_j(X) \wedge S). \quad (40)$$

The set  $\text{Trap}_j(S, E)$  contains those nodes, from where player  $j$  can force to either remain inside  $\text{Trap}_j(S, E)$ , or move to  $E$ , or is already in  $E$ . Note that  $\text{Trap}_j(S \vee E, \perp)$  is different, because it requires the ability to remain inside  $S \vee E$ .

Define  $B_0 \triangleq \text{Attr}_1(A_0)$ , and  $r_0 \triangleq (\text{Trap}_0(B_0, A_0) \wedge B_0) \setminus A_0$ . With this definition of a trap, we can now define the assumption of player 0 about player 1

$$\Box\Diamond(A_0 \vee \neg r_0) = \Box\Diamond(r_0 \rightarrow A_0). \quad (41)$$

This assumption extends the winning set of player 0, only if  $\llbracket r_0 \rrbracket \neq \emptyset$ . Otherwise, the assumption is not useful, and we need to either:

1. introduce a safety assumption that refers to additional variables, or
2. define the specification as a nested game.

In the following, we elaborate on these claims.

### 7.1 The role of machine closure

In Section 6, we conjoined the transition relations with a safety requirement to remain inside the cooperative winning set  $C$ . In this section, we give an example, demonstrating that absence of closure can lead to a contract unrealizable by player 1, together with a contract that is trivially realizable by player 0.

In Fig. 2, nodes from where player 0 (player 1) moves are denoted by disks (boxes). Player 0 wants  $\Box\Diamond G_{0,0}$ , and player 1  $\Box\Diamond G_{1,0}$ . The goal  $G_{1,0}$  is not reachable from nodes  $c, d$ , so these nodes are not in the cooperative winning set  $C$ . Suppose that we ignored this, and used the transition relation  $\rho_1$ , as given

<sup>4</sup> The greatest fixpoint operator  $\nu$  is defined as  $\nu X. f(X) \triangleq \neg\mu X. \neg f(X)$ .



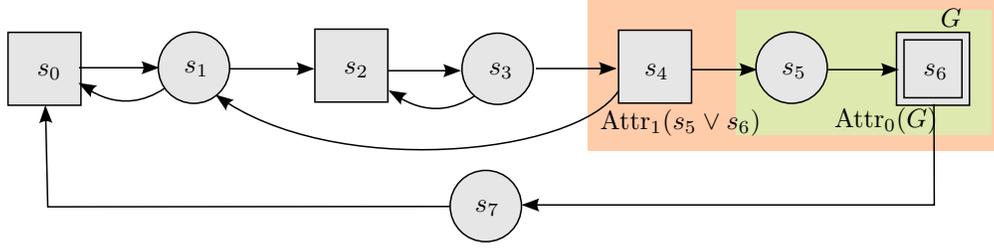


Figure 3: There does not exist a weak fairness assumption that suffices for realizability in this example. Player 0 (player 1) controls the play at disks (boxes).

(1)3. CASE:  $w[k \dots k+1] \not\models \rho_1$

(2)1.  $\forall r \in 0 \dots k-1. w[r \dots r+1] \models \rho_0 \wedge \rho_1$

PROOF: By (1)2,  $k$  is the minimal non-negative integer with this property.

(2)2.  $w, k \models \odot \Box \rho_0$

PROOF: By (2)1.

(2)3.  $w, k \not\models \rho_1$

PROOF: By (1)3.

(2)4. Q.E.D.

PROOF: By (2)2 and (2)3,

$$w, k \not\models (\odot \Box \rho_0) \rightarrow \rho_1 \implies w \not\models \Box((\odot \Box \rho_0) \rightarrow \rho_1).$$

(1)4. CASE:  $w[k \dots k+1] \not\models \rho_0$

PROOF: Similar to (1)3.

(1)5. Q.E.D.

PROOF: By (1)2, the cases (1)3 and (1)4 are exhaustive.

### Proposition 22.

ASSUME: Define the transition relations  $\rho_0, \rho_1$  by the game graph of Fig. 3. Define the set of nodes  $V \triangleq \{s_0, \dots, s_7\}$ . Define the goal  $G = \{s_6\}$  of player 0.

PROVE: There does not exist a set  $\llbracket P \rrbracket \subseteq V$ , such that:

1. the property

$$\varphi_1 \triangleq (\Box \rho_1) \stackrel{\text{sr}}{\rightarrow} (\Box \rho_0 \wedge \Box \Diamond P), \quad (42)$$

be realizable by player 1, and

2. the property

$$\varphi_0 \triangleq (\Box \rho_0 \wedge \Box \Diamond P) \stackrel{\text{sr}}{\rightarrow} (\Box \rho_1 \wedge \Box \Diamond G) \quad (43)$$

be realizable by player 0.

(1)1.  $\Box(\rho_0 \wedge \rho_1)$ .

PROOF: By Proposition 21, if  $\Box(\rho_0 \wedge \rho_1)$  is false for a play, then  $\varphi_0$  or  $\varphi_1$  is false.

(1)2. CASE:  $\llbracket P \rrbracket = \emptyset$

PROOF:  $\Box \Diamond P = \Box \Diamond \perp$  is not realizable by player 1.

(1)3. CASE:  $\llbracket P \rrbracket \neq \emptyset$

(2)1. CASE:  $\llbracket P \rrbracket \cap \{s_0, s_1\} = \emptyset$

(3)1.  $\llbracket P \rrbracket \cap (V \setminus \{s_0, s_1\}) \neq \emptyset$

PROOF: By (1)3 and (2)1.

(3)2.  $\llbracket P \rrbracket \cap \{s_2, \dots, s_7\} \neq \emptyset$

PROOF: By (3)1 and definition of node set  $V$ .

(3)3.  $\Box \Diamond P$  not realizable by player 1.

(4)1. DEFINE: Player 0 strategy

$$f \triangleq (s_3 \rightarrow (s_3 \wedge s'_4)) \wedge (s_1 \rightarrow (s_1 \wedge s'_0))$$

(4)2. If player 0 uses the strategy  $f$  of (4)1, then all plays violate  $\Box \Diamond P$ .

- ⟨5⟩1. From the nodes  $s_2, \dots, s_7$ , the play goes to node  $s_1$ .  
 ⟨5⟩2. From node  $s_1$ , the play is  $s_1(s_0s_1)^\omega$ .  
 ⟨5⟩3. Q.E.D.  
 PROOF: By ⟨5⟩2, any play reaches, and then remains forever in, the set  $\{s_0, s_1\}$ . By ⟨2⟩1, this play does not intersect  $\llbracket P \rrbracket$ , so the play does not satisfy  $\square \diamond P$ .
- ⟨4⟩3. Q.E.D.  
 PROOF: By ⟨4⟩2, player 1 cannot realize  $\square \diamond P$ .
- ⟨3⟩4. Q.E.D.  
 PROOF: By ⟨3⟩3, the consequent of  $\varphi_1$  is false.
- ⟨2⟩2. CASE:  $\llbracket P \rrbracket \cap \{s_0, s_1\} \neq \emptyset$
- ⟨3⟩1. CASE:  $\llbracket P \rrbracket \cap \{s_2, s_3\} = \emptyset$
- ⟨4⟩1. DEFINE: Player 0 strategy
 
$$f \triangleq (s_1 \rightarrow (s_1 \wedge s'_2)) \wedge (s_3 \rightarrow (s_3 \wedge s'_2))$$
- ⟨4⟩2. If player 0 uses the strategy  $f$  of ⟨4⟩1, then all plays violate  $\square \diamond P$ .
- ⟨5⟩1. From the nodes  $s_0, s_4, \dots, s_7$ , the play goes to node  $s_1$ .  
 ⟨5⟩2. From node  $s_1$ , the play is  $s_1(s_2s_3)^\omega$ .  
 ⟨5⟩3. Q.E.D.  
 PROOF: By ⟨5⟩2, the play reaches, and then remains forever in, the set  $\{s_2, s_3\}$ . By ⟨3⟩1, this play does not intersect  $\llbracket P \rrbracket$ , so the play does not satisfy  $\square \diamond P$ .
- ⟨4⟩3. Q.E.D.  
 PROOF: By ⟨4⟩2,  $\varphi_1$  is false.
- ⟨3⟩2. CASE:  $\llbracket P \rrbracket \cap \{s_2, s_3\} \neq \emptyset$
- ⟨4⟩1. DEFINE: Player 1 strategy  $f \triangleq s_4 \rightarrow (s_4 \wedge s'_1)$ .
- ⟨4⟩2. If player 1 uses strategy  $f$  of ⟨4⟩1, and the play is in the set  $\{s_0, \dots, s_4\}$ , then the play remains in  $\{s_0, \dots, s_4\}$  in the next time step.  
 PROOF: The only edge that exits the set  $\{s_0, \dots, s_4\}$ , and satisfies both  $\rho_0$  and  $\rho_1$ , is  $s_4 \wedge s'_5$ . This player 1 edge is not in the strategy  $f$  of ⟨4⟩1.
- ⟨4⟩3. If a play starts in the set  $\{s_5, s_6, s_7\}$ , then it reaches the set  $\{s_0, \dots, s_4\}$  in a finite number of steps.  
 PROOF: By the definition of  $\rho_0, \rho_1$  and ⟨1⟩1.
- ⟨4⟩4. If player 1 uses strategy  $f$  of ⟨4⟩1, then any play enters the set  $\{s_0, \dots, s_4\}$ , and then remains in it.  
 PROOF: By ⟨4⟩2 and ⟨4⟩3.
- ⟨4⟩5. Any play where player 1 uses the strategy  $f$  of ⟨4⟩1 satisfies  $\square \diamond P$ .
- ⟨5⟩1. Any play either reaches, and remains forever in, the set  $\{s_2, s_3\}$ , or it visits node  $s_1$ .
- ⟨6⟩1. It is possible to remain forever in  $\{s_2, s_3\}$ .  
 ⟨6⟩2. If the play exits  $\{s_2, s_3\}$ , then it visits  $s_1$ .  
 PROOF: The only edge that exits  $\{s_2, s_3\}$  is  $s_3 \wedge s'_4$ . By ⟨4⟩1, the next edge is  $s_4 \wedge s'_1$ .
- ⟨6⟩3. Q.E.D.  
 PROOF: By ⟨6⟩1 and ⟨6⟩2.
- ⟨5⟩2. If the play visits node  $s_1$ , then it either visits both  $s_0$  and  $s_1$ , or both  $s_2$  and  $s_3$ .  
 PROOF: Each edge outgoing from node  $s_1$  leads to either  $s_0$  and  $s_1$ , or to  $s_2$  and  $s_3$ .
- ⟨5⟩3. Any play visits the set  $\llbracket P \rrbracket$  infinitely many times.  
 PROOF: By ⟨5⟩1, ⟨5⟩2, the play either visits both  $s_2$  and  $s_3$  infinitely many times, or it reaches  $s_1$  infinitely many times, so also either  $s_2$  and  $s_3$  infinitely many times, or  $s_0$  and  $s_1$  infinitely many times. By ⟨2⟩2 and ⟨3⟩2, the play visits the set  $\llbracket P \rrbracket$  infinitely many times.
- ⟨5⟩4. Q.E.D.  
 PROOF: By ⟨5⟩3, the play satisfies  $\square \diamond P$ .
- ⟨4⟩6. Q.E.D.  
 PROOF: By ⟨4⟩4 and ⟨4⟩5, any play where player 1 uses the strategy  $f$  satisfies  $\square \diamond P$  and violates  $\square \diamond G$ . So,  $\varphi_0$  is not true.
- ⟨3⟩3. Q.E.D.  
 PROOF: By ⟨3⟩1 and ⟨3⟩2.
- ⟨2⟩3. Q.E.D.

PROOF: By ⟨2⟩1 and ⟨2⟩2.

⟨1⟩4. Q.E.D.

PROOF: By ⟨1⟩2 and ⟨1⟩3.

We can make a number of observations. Firstly, there *does* exist a weak fairness assumption *outside* of the GR(1) fragment, such, that the game of Fig. 3 becomes non-trivially realizable. This weak fairness property is in an extension of the GR(1) fragment with action predicates in recurrence properties [58].

In particular, we have to tell player 0 that it is unfair to, forever, hide in the set  $\{s_2, s_3\}$ , i.e.,  $\diamond\Box\neg(s_3 \wedge s'_2)$ . If we add this property both as an assumption of player 1, and as a guarantee by player 0, then trivial realizability persists, because this is a liveness property (ignoring, for a moment, that this results in a Rabin(1) game). Thus, it should not be added as a guarantee for player 0.

But we can subtract this property from the assumption of player 0. Consider the desired assume-guarantee pair for player 1

$$\diamond\Box\neg(s_3 \wedge s'_2) \rightarrow \Box\diamond((s_1 \vee s_2 \vee s_3 \vee s_4) \rightarrow s_5) \quad (44)$$

Then, merge the antecedent (persistence) and consequent (recurrence) into a single recurrence property

$$\Box\diamond((s_3 \wedge s'_2) \vee s_5 \vee \neg(s_1 \vee s_2 \vee s_3 \vee s_4)). \quad (45)$$

This property is realizable by player 1, but *not* in the GR(1) fragment, because  $(s_3 \wedge s'_2)$  is an edge. It is in an extension of GR(1) with edges in liveness properties [58].

Moreover, the above property can be expressed in GR(1), by shifting the above transition formula one step into the past, as  $\Box\diamond\Box(\dots)$ . This introduces a *history variable*, for remembering the past, and a safety property about this variable's update behavior. Pnueli observes in [14] the equivalence of auxiliary variables, with allowing the past. We observe that describing in GR(1) this weak fairness property, which involves a transition relation, introduces a safety property, and increases the number of variables.

In general, a weak fairness assumption over edges (of both players) in the game graph can be computed by finding a trap set that is sufficiently large, to prevent player 1 from satisfying the assumption, by going away from the goal desired by player 0 (e.g., the edge  $s_4 \wedge s'_1$  in Fig. 3). Such a set can lead to trivial realizability. In order to prevent trivial realizability, edges of player 0 that lead away from the goal can be subtracted from the assumption, as we did above with the edge  $s_3 \wedge s'_2$ . These edges can be computed by considering consecutive iterates of a reachability computation in the cooperative winning set.

Here, we decide to use the GR(1) fragment, with recurrence properties over nodes, because recurrence assumptions that refer to edges of player 0 need to include all backward leading edges inside the trap set. Therefore, this type of assumptions explicitly refers to the transition relation, over a set of nodes. As a result, it leads to more complex and detailed formulae, which are less amenable to simplification, and are less suitable for an extension to cases with hidden variables.

Note that in a non-interleaving representation, both primed and unprimed variables are required to represent nodes from where player 1 moves. In more detail, player 1 moves from nodes of the form  $(x'_0, x_1, i)$ . Even though such a representation involves primed variables, in the game graph, these are still nodes, not edges. Therefore, in a non-interleaving representation, the propositions have the same semantics, but with different syntax.

A more direct approach is to introduce safety, by requiring that  $\Box(s_4 \rightarrow (s_4 \wedge s'_5))$ . This resolves the non-determinism, by fixing a choice (undesirable). However, such a fixed safety assumption may not exist, as proved by the following.

**Proposition 23** (Nonexistence of safety).

ASSUME: Define the transition relations  $\rho_0, \rho_1$  by the game graph of Fig. 4.

PROVE: There does not exist a set  $\llbracket\rho\rrbracket \subsetneq \llbracket\rho_1\rrbracket$ , such that player 1 chooses edges that satisfy  $\rho$ , and

$$\varphi \triangleq \Box(\rho_0 \wedge \rho) \wedge \Box\diamond G_1 \wedge \Box\diamond G_2 \quad (46)$$

is satisfiable (cooperatively by player 0 and player 1).

⟨1⟩1.  $\Box(\rho_0 \wedge \rho)$

PROOF: If  $\Box(\rho_0 \wedge \rho_1)$  is false for a play, then  $\varphi$  is false.

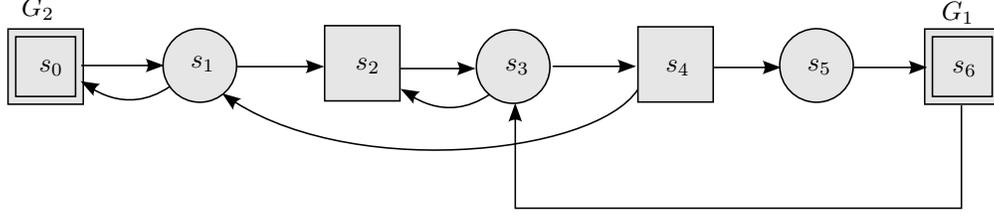


Figure 4: There does not exist a realizable GR(1) property that suffices as an assumption in this example, as proved in Lemma 25. Player 0 (player 1) controls the play at disks (boxes).

- (1)2. CASE:  $\rho$  does not include the edge  $s_0 \wedge s'_1$ .
  - (2)1. No infinite play visits  $\llbracket G_2 \rrbracket$ .
    - PROOF: By (1)2, if a play visits node  $s_0$ , then there is no next node.
  - (2)2. Q.E.D.
    - PROOF: By (2)1, no play satisfies the property  $\square \diamond G_2$ .
- (1)3. CASE:  $\rho$  does not include the edge  $s_2 \wedge s'_3$ .
  - (2)1. If an infinite play visits  $\llbracket G_2 \rrbracket$ , then it does not satisfy  $\square \diamond G_1$ .
    - PROOF: By (1)3, no path exists from the set  $\llbracket G_2 \rrbracket = \{s_0\}$ , to the set  $\llbracket G_1 \rrbracket = \{s_6\}$ .
  - (2)2. CASE: A play satisfies  $\square \diamond G_2$ .
    - PROOF: By (2)2, the play visits  $\llbracket G_2 \rrbracket$ , so by (2)1, the play violates  $\square \diamond G_1$ .
  - (2)3. CASE: A play violates  $\square \diamond G_2$ .
    - PROOF: By definition of  $\varphi$ .
  - (2)4. Q.E.D.
    - PROOF: By (2)2 and (2)3, with such a  $\rho$ , no play satisfies  $\varphi$ .
- (1)4. CASE:  $\rho$  does not include the edge  $s_4 \wedge s'_1$ .
  - (2)1. If an infinite play visits  $\llbracket G_1 \rrbracket$ , then it does not satisfy  $\square \diamond G_2$ .
    - PROOF: By (1)4, there is no path from  $\llbracket G_1 \rrbracket$  to  $\llbracket G_2 \rrbracket$ .
  - (2)2. CASE: A play satisfies  $\square \diamond G_1$ .
    - PROOF: By (2)2, the play visits  $\llbracket G_1 \rrbracket$ , so by (2)1, the play violates  $\square \diamond G_2$ .
  - (2)3. CASE: A play violates  $\square \diamond G_1$ .
    - PROOF: By definition of  $\varphi$ .
  - (2)4. Q.E.D.
    - PROOF: By (2)2 and (2)3, with such a  $\rho$ , no play satisfies  $\varphi$ .
- (1)5. CASE:  $\rho$  does not include the edge  $s_4 \wedge s'_5$ .
  - (2)1. If a play visits node  $s_6$ , then it does not revisit  $s_6$ .
    - PROOF: By (1)5, there does not exist a path from node  $s_6$  to node  $s_6$ .
  - (2)2. Q.E.D.
    - PROOF: By (2)1, no play satisfies  $\square \diamond G_1$ . By definition of  $\varphi$ , no play satisfies  $\varphi$ .
- (1)6. CASE:  $\rho$  does not include the edge  $s_6 \wedge s'_3$ .
  - PROOF: Similar to (1)2, but for the set  $\llbracket G_1 \rrbracket$ .
- (1)7. Q.E.D.
  - (2)1.  $\rho \subseteq \rho_1$  and  $\rho \neq \rho_1$ .
    - PROOF: By hypothesis.
  - (2)2. The transition relation  $\rho$  has at least one fewer edge than  $\rho_1$ .
    - PROOF: By (2)1.
  - (2)3. The cases (1)2–(1)6 are exhaustive.
    - PROOF: By (2)2, the definition, by hypothesis, of  $\rho_1$  with 5 edges, and the case statements (1)2–(1)6 for those 5 edges.
  - (2)4. Q.E.D.
    - PROOF: By (2)3.

In Proposition 23, we proved lack of satisfiability, not lack of mutual realizability. This condition (for safety here) is stronger than in Proposition 22 (for recurrence there). The reason is that we will need to

combine the result with Proposition 24. If a property (safety) is not realizable by player 1, then conjoining with another property (recurrence) restricts it further, so it remains unrealizable by player 1. In general, if a property does not suffice as an assumption for player 0, it is not true that restricting it will yield a property unrealizable by player 0. However, if a property  $P \wedge Q$  is unsatisfiable cooperatively by the two players, then further restriction yields an unsatisfiable property.

Suppose that the property  $P$  is realizable by player 1, and the property  $P \overset{\text{sr}}{\Rightarrow} Q$  by player 0. Use a strategy for each player to control all the variables. The composite strategy satisfies  $P \wedge (P \overset{\text{sr}}{\Rightarrow} Q)$ , so also  $P \wedge Q$ . Therefore, the property  $P \wedge Q$  is satisfiable cooperatively, a contradiction. So, the restriction of  $P$  to  $\hat{P}$  yields assume-guarantee pairs  $\hat{P}$  for player 1, and  $\hat{P} \overset{\text{sr}}{\Rightarrow} Q$ , of which at least one is not realizable.

We turn now to the nonexistence of a recurrence assumption for the conjoined goals  $(\Box \Diamond G_1) \wedge (\Box \Diamond G_2)$ . For each of the goals  $\Box \Diamond G_1$  and  $\Box \Diamond G_2$ , there exists a recurrence assumption  $\Box \Diamond P$ , such that both the formula  $(\Box \rho_0) \overset{\text{sr}}{\Rightarrow} (\Box \rho_1 \wedge \Box \Diamond P)$  is realizable by player 1, and the formula  $(\Box \rho_1 \wedge \Box \Diamond P) \overset{\text{sr}}{\Rightarrow} (\Box \rho_0 \wedge \Box \Diamond G_i)$  is realizable by player 2. In particular,

- $\Box \Diamond s_3$  for  $\Box \Diamond G_2$
- $\Box \Diamond (s_0 \vee s_2)$  for  $\Box \Diamond G_1$ .

The mutual realizability for these assumptions has been confirmed with a GR(1) synthesizer.

**Proposition 24** (Nonexistence of recurrence).

ASSUME: Define  $\rho_i$  the transition relation of player  $i$  by the game of Fig. 4. Define the set of nodes  $V \triangleq \{s_0, \dots, s_6\}$ .

PROVE: For all sets of nodes  $P \subseteq V$ , for any initial node, either

- the property

$$\varphi_1 \triangleq (\Box \rho_1) \overset{\text{sr}}{\Rightarrow} (\Box \rho_0 \wedge \Box \Diamond P), \quad (47)$$

is not realizable by player 1, or

- the property

$$\varphi_0 \triangleq (\Box \rho_0 \wedge \Box \Diamond P) \overset{\text{sr}}{\Rightarrow} (\Box \rho_1 \wedge \Box \Diamond G_1 \wedge \Box \Diamond G_2), \quad (48)$$

is not realizable by player 0.

(1)1.  $\Box(\rho_0 \wedge \rho_1)$

PROOF: By Proposition 21, if  $\Box(\rho_0 \wedge \rho_1)$  is false for a play, then  $\varphi_0$  or  $\varphi_1$  is false.

(1)2. CASE:  $\llbracket P \rrbracket = \emptyset$

PROOF: By (1)1 and (1)2,  $\varphi_1$  is false.

(1)3. CASE:  $\llbracket P \rrbracket \neq \emptyset$ .

(2)1. CASE:  $\llbracket P \rrbracket \cap \{s_2, s_3\} = \emptyset$ .

(3)1. DEFINE: Player 0 strategy

$$f \triangleq (s_1 \rightarrow (s_1 \wedge s'_2)) \wedge (s_3 \rightarrow (s_3 \wedge s'_2)).$$

(3)2. If player 0 uses strategy  $f$ , then no play satisfies  $\Box \Diamond P$ .

(4)1. From the set  $\{s_0, s_2, s_4, s_5, s_6\}$ , the play visits either node  $s_1$ , or node  $s_3$ .

(4)2. If player 0 uses strategy  $f$ , then from the nodes in  $\{s_1, s_3\}$ , the play is  $(s_2 s_3)^\omega$ .

PROOF: By the strategy  $f$  of (3)1.

(4)3. Any play is of the form  $s_i^* (s_2 s_3)^\omega$ .

PROOF: By (4)1, (4)2, and that these cases cover  $V$ .

(4)4. Q.E.D.

PROOF: By (4)3 and (2)1.

(3)3. Q.E.D.

PROOF: By (3)2, if player 0 uses the strategy  $f$  of (3)1, then player 1 cannot realize property  $\varphi_1$ .

(2)2. CASE:  $\llbracket P \rrbracket \cap \{s_2, s_3\} \neq \emptyset$ .

(3)1. CASE:  $\llbracket P \rrbracket \cap \{s_0, s_1\} \neq \emptyset$ .

(4)1. DEFINE: Player 1 strategy

$$f \triangleq s_4 \rightarrow (s_4 \wedge s'_1).$$

(4)2. If a play visits a node in  $\{s_5, s_6\}$ , then it later visits the set  $\{s_0, \dots, s_4\}$ .

- ⟨4⟩3. If player 1 uses strategy  $f$  and a play visits the set  $\{s_0, \dots, s_4\}$ , then the play remains forever in it.
- ⟨5⟩1. The only edge that exits  $\{s_0, \dots, s_4\}$  is  $s_4 \wedge s'_5$ .
- ⟨5⟩2. The edge  $s_4 \wedge s'_5$  is not in the strategy  $f$  of ⟨4⟩1.
- ⟨5⟩3. Q.E.D.
- By ⟨5⟩1 and ⟨5⟩2.
- ⟨4⟩4. If player 1 uses strategy  $f$ , then no play visits  $G_1$  an infinite number of times.
- PROOF: By ⟨4⟩2 and ⟨4⟩3.
- ⟨4⟩5. Any play that remains in  $\{s_0, \dots, s_4\}$  visits  $s_0$  and  $s_1$ , or  $s_2$  and  $s_3$ , an infinite number of times.
- ⟨4⟩6. Any play that remains in  $\{s_0, \dots, s_4\}$  satisfies  $\square \diamond P$ .
- PROOF: By ⟨4⟩5, ⟨2⟩2, and ⟨3⟩1.
- ⟨4⟩7. If player 1 uses strategy  $f$ , then all plays satisfy  $\square \diamond P$ .
- PROOF: By ⟨4⟩2, ⟨4⟩3, and ⟨4⟩6.
- ⟨4⟩8. Q.E.D.
- PROOF: By ⟨4⟩4 and ⟨4⟩7, if player 1 uses strategy  $f$ , then all plays satisfy  $\square \diamond P$  and violate  $\square \diamond G_1$ . By definition of  $\varphi_0$ , all plays violate  $\varphi_0$ . So, there does not exist a winning strategy for player 0.
- ⟨3⟩2. CASE:  $\llbracket P \rrbracket \cap \{s_0, s_1\} = \emptyset$ .
- ⟨4⟩1. CASE:  $\llbracket P \rrbracket \cap \{s_3, \dots, s_6\} = \emptyset$ .
- ⟨5⟩1. DEFINE: Player 0 strategy
- $$f \triangleq (s_1 \rightarrow (s_1 \wedge s'_0)) \wedge (s_3 \rightarrow (s_3 \wedge s'_4)).$$
- ⟨5⟩2. If player 0 uses strategy  $f$ , then no play visits node  $s_2$  infinitely many times.
- PROOF: If a play starts at node  $s_2$ , then it leaves  $s_2$ . By ⟨5⟩1, if a play is not at node  $s_2$ , then none of the edges incoming to  $s_2$  is in the strategy ⟨4⟩1.
- ⟨5⟩3.  $\llbracket P \rrbracket = \{s_2\}$
- PROOF: By ⟨2⟩2, ⟨3⟩2, and ⟨4⟩1.
- ⟨5⟩4. If player 0 uses strategy  $f$ , then no play satisfies  $\square \diamond P$ .
- PROOF: By ⟨5⟩2 and ⟨5⟩3.
- ⟨5⟩5. Q.E.D.
- PROOF: By ⟨5⟩4, if player 0 uses strategy  $f$ , then all plays violate  $\square \diamond P$ . By definition of  $\varphi_1$ , all plays violate  $\varphi_1$ . So, there does not exist a winning strategy for player 1.
- ⟨4⟩2. CASE:  $\llbracket P \rrbracket \cap \{s_3, \dots, s_6\} \neq \emptyset$ .
- ⟨5⟩1. CASE: Initial node in  $\{s_0, s_1\}$ .
- ⟨6⟩1. DEFINE: Player 0 strategy
- $$f \triangleq s_1 \rightarrow (s_1 \wedge s'_0).$$
- ⟨6⟩2. If player 0 uses strategy  $f$ , then all plays remain in the set  $\{s_0, s_1\}$ .
- PROOF: By ⟨5⟩1 and ⟨6⟩1.
- ⟨6⟩3. If player 0 uses strategy  $f$ , then all plays violate  $\square \diamond P$ .
- PROOF: By ⟨6⟩2 and ⟨3⟩2.
- ⟨6⟩4. Q.E.D.
- PROOF: By definition of  $\varphi_1$ , and ⟨6⟩3, if player 0 uses strategy  $f$ , then all plays violate  $\varphi_1$ . So, there does not exist a winning strategy for player 1.
- ⟨5⟩2. CASE: Initial node not in  $\{s_0, s_1\}$ .
- ⟨6⟩1. DEFINE: Player 1 strategy
- $$f \triangleq s_4 \rightarrow (s_4 \wedge s'_5).$$
- ⟨6⟩2. If player 1 uses strategy  $f$ , then all plays visit infinitely many times either  $s_2$  and  $s_3$ , or  $s_3, s_4, s_5$  and  $s_6$ .
- PROOF: By ⟨1⟩1, ⟨5⟩2, ⟨6⟩1, the play remains in the set  $\{s_2, \dots, s_6\}$ . The only cycles in  $\{s_2, \dots, s_6\}$  are  $s_2, s_3$  and  $s_3, s_4, s_5, s_6$ . By the pigeonhole principle, at least one of these two cycles must be visited an infinite number of times.
- ⟨6⟩3. If player 1 uses strategy  $f$ , then all plays satisfy  $\square \diamond P$ .
- PROOF: By ⟨6⟩2, ⟨2⟩2 and ⟨4⟩2.
- ⟨6⟩4. If player 1 uses strategy  $f$ , then no play visits  $\llbracket G_2 \rrbracket$ .
- ⟨7⟩1. All plays start outside  $\{s_0, s_1\}$ .
- PROOF: By ⟨5⟩2.

⟨7⟩2. No play that is outside  $\{s_0, s_1\}$ , enters  $\{s_0, s_1\}$ .  
 PROOF: By ⟨6⟩1.  
 ⟨7⟩3. No play visits  $\{s_0, s_1\}$ .  
 PROOF: By ⟨7⟩1 and ⟨7⟩2.  
 ⟨7⟩4. Q.E.D.  
 PROOF: By ⟨7⟩3 and the definition of  $G_2$ .  
 ⟨6⟩5. If player 1 uses strategy  $f$ , then no play satisfies  $\Box\Diamond G_2$ .  
 PROOF: By ⟨6⟩4.  
 ⟨6⟩6. Q.E.D.  
 PROOF: By ⟨6⟩3 and ⟨6⟩5, if player 1 uses strategy  $f$ , then all plays satisfy  $\Box\Diamond P$  and violate  $\Box\Diamond G_2$ . By definition of  $\varphi_0$ , all plays violate  $\varphi_0$ . So, there does not exist a winning strategy for player 0.  
 ⟨5⟩3. Q.E.D.  
 PROOF: By ⟨5⟩1 and ⟨5⟩2, that cover all initial nodes in  $V$ .  
 ⟨4⟩3. Q.E.D.  
 PROOF: By ⟨4⟩1 and ⟨4⟩2.  
 ⟨3⟩3. Q.E.D.  
 PROOF: By ⟨3⟩1 and ⟨3⟩2.  
 ⟨2⟩3. Q.E.D.  
 PROOF: By ⟨2⟩1 and ⟨2⟩2.  
 ⟨1⟩4. Q.E.D.  
 PROOF: By ⟨1⟩2 and ⟨1⟩3.

**Lemma 25** (Nonexistence of GR(1) assumption). *Define the transition relations  $\rho_0, \rho_1$  as in the game of Fig. 4. There does not exist a property  $P$  in the GR(1) fragment, such that*

$$\varphi_1 \triangleq (\Box\rho_1) \text{ sr} \triangleright (\Box\rho_0 \wedge P) \quad (49)$$

*be realizable by player 1, and*

$$\varphi_0 \triangleq (\Box\rho_0 \wedge P) \text{ sr} \triangleright (\Box\rho_1 \wedge \Box\Diamond G_1 \wedge \Box\Diamond G_2). \quad (50)$$

*be realizable by player 0.*

PROOF: By Proposition 23 and Proposition 24.

This can be avoided, by introducing a goal counter *goal* as auxiliary variable, and switch between safety assumptions, depending on the counter, e.g.,  $\Box((s_4 \wedge \text{goal} = 1) \rightarrow (s_4 \wedge s'_5))$ .

Here, we decide to not introduce explicitly new variables in the contract, neither safety assumptions that fix choices of edges. Instead, in Section 8, we will define nested games, where the safety assumptions are introduced by partitioning the game graph into sub-games, and avoid explicit reference to extra variables inside the formula. The purpose served by those extra variables is achieved by structuring the contract into multiple games.

## 8 Nested games

A structured way of isolating conditional assumptions is by partitioning the game into smaller ones. Each smaller game has its own assumptions, independently of the other games. This prevents circularity of liveness dependencies. Each game has one reachability objective: to reach the game that contains it. Only unconditional liveness assumptions can appear inside each game. Assumptions that themselves depend on other liveness assumptions become objectives in their own game. The games partition the game graph. The approach of nested games is reminiscent of McNaughton's recursive algorithm for solving parity games [57].

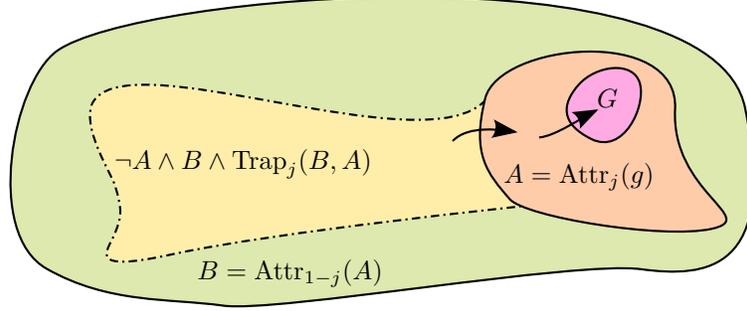


Figure 5: The sets (labeled by predicates) computed by UNCONDITIONALASSUMPTION in Algorithm 1.

---

**Algorithm 1** Construction of nested-game GR(1) specification, for each recurrence goal  $G$

---

```

1: procedure GAMESTACK( $j, G, uncovered, stack$ )
2:    $trap \leftarrow \top$ 
3:    $goal \leftarrow G$ 
4:    $stack \leftarrow \text{set}()$ 
5:   while  $\llbracket trap \rrbracket \neq \emptyset$  do ▷ Create unconditional assumptions, until stuck
6:      $attr, trap \leftarrow \text{UNCONDITIONALASSUMPTION}(j, goal)$ 
7:      $goal \leftarrow attr \cup trap$ 
8:      $assumptions.add(\Box \Diamond (trap \rightarrow attr))$ 
9:      $game \leftarrow (j, goal \wedge \neg G, G, assumptions)$ 
10:     $stack.append(game)$ 
11:     $uncovered \leftarrow uncovered \wedge \neg goal$ 
12:    if  $\llbracket uncovered \rrbracket = \emptyset$  then ▷ Covered cooperatively winning set?
13:      return
14:    GAMESTACK( $1 - j, goal, uncovered, stack$ ) ▷ Construct a nested game
15:    return
16: procedure UNCONDITIONALASSUMPTION( $j, g$ )
17:    $A \leftarrow \text{Attr}_j(g)$ 
18:    $B \leftarrow \text{Attr}_{1-j}(A)$ 
19:    $r \leftarrow \neg A \wedge B \wedge \text{Trap}_j(B, A)$ 
20:   return  $A, r$ 

```

---

Algorithm 1 computes a stack of nested games, for reaching a goal  $G$ . It covers the cooperative winning set  $C$ , so a later visit to  $G$  is always possible, from any node in  $C$ . Part of the computation is illustrated in Fig. 5.

**Proposition 26** (GAMESTACK variant). *If procedure GAMESTACK calls GAMESTACK (L14), then the set  $\llbracket uncovered \rrbracket$  after L11 in the caller has at least one more node than  $\llbracket uncovered \rrbracket$  after L11 in the callee.*

*Proof.* Consider a call to GAMESTACK by GAMESTACK (L14). Variables in the caller, and in its last call to UNCONDITIONALASSUMPTION (L6) will be indexed by 1. Variables in the callee, and in its first call to UNCONDITIONALASSUMPTION (L6) will be indexed by 2.

We will prove that, in the first call of the callee to UNCONDITIONALASSUMPTION (L6), the attractor  $A_2 = \text{Attr}_{j_2}(g_2)$  will be strictly larger than  $g_2$  (L17). We need to prove that there is a node outside  $g_2$ , from where player  $j_2$  can force a visit to  $g_2$ . It is  $g_2 = goal_2$  (L16,6) in the first iteration of the loop (L5). First iteration implies  $goal_2 = G_2$  (L3). In the caller,  $goal_1 = G_2$  (L14,1), so  $g_2 = goal_2 = G_2 = goal_1$ . The value of  $j_2$  (L17) is  $1 - j_1$  in the caller (L16,6,1,14).

In the caller, L14 was reached. So the loop terminated, implying  $\llbracket trap_1 \rrbracket = \emptyset$  (L5). In the last loop iteration,  $\llbracket trap_1 \rrbracket = \emptyset$  implies that  $\llbracket goal_1 \rrbracket = \llbracket attr_1 \rrbracket$  (L7). The return statement (L12) was not executed, so  $\llbracket uncovered_{1,L12} \rrbracket \neq \emptyset$ . By L11,  $\llbracket uncovered_{1,L12} \rrbracket \neq \emptyset$  implies that  $goal_1$  does not cover  $uncovered_{1,L1}$  at L1. By  $\llbracket goal_1 \rrbracket = \llbracket attr_1 \rrbracket$ , it follows that  $\llbracket attr_1 \rrbracket$  does not cover  $uncovered_{1,L1}$ .

It is  $A_1 = attr_1$  (L6,20) from the last call to UNCONDITIONALASSUMPTION. So  $A_1 = Attr_{j_1}(g_1)$  does not cover  $uncovered_{1,L1}$ . By definition,  $uncovered_{1,L1}$  is a subset of the cooperative winning set, and goal  $g_1$  is contained in  $A_1$ . So, any node in  $uncovered_{1,L1}$  can reach  $g_1$ , thus also  $A_1$ .

Suppose that no node of player  $(1-j_1)$  in  $uncovered_{1,L12} = uncovered_{1,L1} \wedge \neg goal_1 = uncovered_{1,L1} \wedge \neg A_1$  has an edge that leads to  $A_1$ . Then,  $uncovered_{1,L1} \wedge \neg A_1$  (non-empty) must contain a node of player  $j_1$  that has an edge to  $A_1$ . This node must<sup>5</sup> be in  $A_1$ , because  $A_1$  is an attractor for player  $j_1$ . This is a contradiction. We conclude that at least one node of player  $j_2 = 1-j_1$  is in  $uncovered_{1,L1} \wedge \neg A_1$  and has an edge to  $A_1$ . This node is outside  $A_1 = attr_1 = goal_1 = G_2$ , and will be in  $A_2 = Attr_{j_2}(g_2) = Attr_{1-j_1}(A_1)$  in the first call to UNCONDITIONALASSUMPTION by the callee. This proves the claim.  $\square$

**Proposition 27** (GAMESTACK Termination). *If the game graph is finite, then any call to procedure GAMESTACK of Algorithm 1 terminates.*

*Proof.* A call to GAMESTACK may not terminate for two reasons: the loop or the recursion never terminate. Suppose that the loop never terminates, so  $\llbracket trap \rrbracket \neq \emptyset$ . It is  $goal = g$  (L6,16) and  $\llbracket g \rrbracket \subseteq \llbracket Attr_j(g) \rrbracket$  (attractor def) and  $A = Attr_j(g)$  (L17), so  $\llbracket goal \rrbracket \subseteq \llbracket A \rrbracket$ .

The set  $\llbracket trap \rrbracket = \llbracket r \rrbracket$  (L6,20) and  $\llbracket r \rrbracket \cap \llbracket A \rrbracket = \emptyset$  (L19), so  $\llbracket trap \rrbracket \cap \llbracket A \rrbracket \neq \emptyset$ . We supposed that  $\llbracket trap \rrbracket \neq \emptyset$ , so the set  $\llbracket trap \rrbracket$  contains nodes outside  $\llbracket A \rrbracket$ . By  $\llbracket goal \rrbracket \subseteq \llbracket A \rrbracket$ , it follows that  $\llbracket trap \rrbracket$  contains nodes outside  $\llbracket goal \rrbracket$ .

So the set  $\llbracket goal \rrbracket$  increases strictly in each iteration. By hypothesis, the game graph has a finite number of nodes, so  $goal$  will eventually cover the graph, implying that  $\llbracket B \rrbracket = \llbracket A \rrbracket$  (L18), thus  $\llbracket trap \rrbracket = \llbracket r \rrbracket \subseteq \llbracket \neg A \wedge B \rrbracket = \llbracket \neg A \wedge A \rrbracket = \emptyset$ . This contradicts the supposition  $\llbracket trap \rrbracket = \emptyset$ . So the loop at L5 terminates.

Suppose that the number of recursive calls to GAMESTACK is infinite. By Proposition 26, with each recursive call to GAMESTACK, the cardinality of the set  $\llbracket uncovered \rrbracket$  decreases by at least one. We supposed an infinite number of recursive calls, so in some recursive call to GAMESTACK,  $\llbracket uncovered \rrbracket = \emptyset$ . So the guard of L12 becomes true, and that call returns, without any further recursion, a contradiction. Therefore, the number of recursive calls is finite.  $\square$

Upon termination, the algorithm has computed a stack of games, each game is in effect in a subset of the game graph.

The time complexity is at most quadratic in the number of nodes, with time measured by  $CPre_j$  calls. This complexity follows because of single alternation of least and greatest fixpoints (L17–19). For each call to UNCONDITIONALASSUMPTION either  $\llbracket trap \rrbracket = \emptyset$ , so by Proposition 26 the next call to UNCONDITIONALASSUMPTION will remove a node from the uncovered ones, or  $\llbracket trap \rrbracket \neq \emptyset$  so by Proposition 27, the current call removes a node from the uncovered ones. Therefore, UNCONDITIONALASSUMPTION is called at most  $2|\Sigma|$  times.

Each call to UNCONDITIONALASSUMPTION contains two chained attractor computations, and a trap computation. Each of these can invoke  $CPre_j$  at most  $|\Sigma|$  times. The previous two statements imply that the time complexity is at most quadratic in the number of game graph nodes.

Note that searching for fewer assumptions, inducing a smaller winning set, can be exponentially expensive, as proved for syntactic recurrence formulae in [43]. Conceptually, the nesting of games has common elements with modular game graphs [59] and open temporal logic [60].

Let us revisit the example of Fig. 3, to observe the algorithm's execution. Player 0 wants  $\square \diamond G$ . The first call to GAMESTACK will call UNCONDITIONALASSUMPTION. Player 0 can force a visit to  $s_6$  from the attractor  $A = Attr_0(s_6) = s_5 \vee s_6$ . Player 1 can force  $A$  from  $B = Attr_1(A) = s_4 \vee s_5 \vee s_6$ . But  $r = \perp$ , because player 1 can escape to  $s_1$ .

So, a nested game is constructed over  $s_0 \vee s_1 \vee s_2 \vee s_3 \vee s_4$ , with player 1 wanting  $\diamond(s_5 \vee s_6)$ . In the nested game,  $A = Attr_1(s_5 \vee s_6) = s_4 \vee s_5 \vee s_6$ . The attractor  $B = Attr_0(s_4 \vee s_5 \vee s_6) = \top$ , and player 0 can keep player 1 in there, until player 0 visits  $s_4 \vee s_5 \vee s_6$ . So, in the nested game, player 1 makes the assumption  $\square \diamond((s_0 \vee s_1 \vee s_2 \vee s_3) \rightarrow (s_4 \vee s_5 \vee s_6)) = \square \diamond \neg(s_0 \vee s_1 \vee s_2 \vee s_3)$ . This covers the cooperative winning set, in this example the entire game graph.

In implementation, the players need to communicate, and select a leader in a cyclic order. Each player becomes a leader in turn. Each time a player becomes a leader, it selects its next recurrence goal, in cyclic

<sup>5</sup> In a turn-based game, from each node, a single player controls all edges. This argument would not hold in a concurrent game, a consequence of lacking determinacy [25, 20].

order. It announces the current goal, by using an auxiliary integer variable dedicated to this purpose. Note that this operation is analogous to centralized transducer construction [51]. The goal corresponds to a game stack, as constructed above. Therefore, all players switch to playing the game that corresponds to the current node (i.e., current state). By construction of the stack, the play will be led to the selected goal. When the goal is reached, the leader selects the next leader, and the sequence repeats.

**Acknowledgments** This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

## References

- [1] W. Thomas, “Solution of church’s problem: A tutorial,” *New Perspectives on Games and interaction*, vol. 5, 2008.
- [2] C. Hoare, “An axiomatic basis for computer programming,” *CACM*, vol. 12, no. 10, pp. 576–580, 1969.
- [3] R. W. Floyd, “Assigning meanings to programs,” in *Symposia in Applied Mathematics*, vol. 19, 1967, pp. 19–32.
- [4] F. B. Schneider, *On concurrent programming*. Springer, 1997.
- [5] A. Pnueli and R. Rosner, “On the synthesis of a reactive module,” in *POPL*, 1989, pp. 179–190.
- [6] N. Francez and A. Pnueli, “A proof method for cyclic programs,” *Acta Informatica*, vol. 9, pp. 133–157, 1978.
- [7] L. Lamport, “Specifying concurrent program modules,” *TOPLAS*, vol. 5, no. 2, pp. 190–222, 1983.
- [8] —, “The “Hoare logic” of concurrent programs,” *Acta Informatica*, vol. 14, pp. 21–37, 1980.
- [9] L. Lamport and F. B. Schneider, “The “Hoare Logic” of CSP, and all that,” *TOPLAS*, vol. 6, no. 2, pp. 281–296, 1984.
- [10] J. Misra and K. Chandy, “Proofs of networks of processes,” *TSE*, vol. 7, no. 4, pp. 417–426, 1981.
- [11] A. Pnueli, “The temporal logic of programs,” in *FOCS*, 1977, pp. 46–57.
- [12] L. Lamport, “Proving the correctness of multiprocess programs,” *TSE*, vol. 3, no. 2, pp. 125–143, 1977.
- [13] S. Owicki and L. Lamport, “Proving liveness properties of concurrent programs,” *TOPLAS*, vol. 4, no. 3, pp. 455–495, 1982.
- [14] A. Pnueli, “In transition from global to modular temporal reasoning about programs,” in *Logics and models of concurrent systems*, ser. NATO ASI Series, K. R. Apt, Ed. Springer, 1985, vol. F13, pp. 123–144.
- [15] M. Abadi and L. Lamport, “Open systems in TLA,” in *PODC*, 1994, pp. 81–90.
- [16] —, “Conjoining specifications,” *TOPLAS*, vol. 17, no. 3, pp. 507–535, 1995.
- [17] E. W. Stark, “A proof technique for rely/guarantee properties,” *Foundations of Software Technology and Theoretical Computer Science*, vol. 206, pp. 369–391, 1985.
- [18] K. L. McMillan, “Circular compositional reasoning about liveness,” in *CHARME*, 1999, pp. 342–346.
- [19] B. Meyer, “Applying “design by contract”,” *Computer*, vol. 25, no. 10, pp. 40–51, 1992.
- [20] L. de Alfaro and T. A. Henzinger, “Interface automata,” in *ESEC/FSE*, 2001, pp. 109–120.

- [21] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen, “Contracts for systems design,” INRIA, Tech. Rep. 8147, 2012. [Online]. Available: <https://hal.inria.fr/hal-00757488>
- [22] P. Nuzzo, A. Iannopolo, S. Tripakis, and A. Sangiovanni-Vincentelli, “Are interface theories equivalent to contract theories?” in *MEMOCODE*, 2014, pp. 104–113.
- [23] A. Cimatti and S. Tonetta, “A property-based proof system for contract-based design,” in *EUROMICRO*, 2012, pp. 21–28.
- [24] A. Cimatti, M. Dorigatti, and S. Tonetta, “OCRA: A tool for checking the refinement of temporal contracts,” in *ASE*, 2013, pp. 702–705.
- [25] R. Alur, T. A. Henzinger, and O. Kupferman, “Alternating-time temporal logic,” in *JACM*, 2002, pp. 672–713.
- [26] L. de Alfaro, T. A. Henzinger, and F. Y. Mang, “The control of synchronous systems,” in *CONCUR*, 2000, pp. 458–473.
- [27] S. Schewe and B. Finkbeiner, “Synthesis of asynchronous systems,” in *LOPSTR*, 2007, pp. 127–142.
- [28] N. Piterman, A. Pnueli, and Y. Sa’ar, “Synthesis of reactive(1) designs,” in *VMCAI*, 2006, pp. 364–380.
- [29] A. Pnueli and U. Klein, “Synthesis of programs from temporal property specifications,” in *MEMOCODE*, 2009, pp. 1–7.
- [30] O. Kupferman and M. Y. Vardi, “Synthesis with incomplete informatio,” in *Advances in Temporal Logic*. Springer, 2000, pp. 109–127.
- [31] O. Kupferman and M. Vardi, “Safraless decision procedures,” in *FOCS*, 2005, pp. 531–540.
- [32] M. de Wulf, L. Doyen, and J.-F. Raskin, “A lattice theory for solving games of imperfect information,” in *HSCC*, 2006, pp. 153–168.
- [33] A. Pnueli and R. Rosner, “Distributed reactive systems are hard to synthesize,” in *FOCS*, vol. 2, 1990, pp. 746–757.
- [34] B. Finkbeiner and S. Schewe, “Uniform distributed synthesis,” in *LICS*, 2005, pp. 321–330.
- [35] K. Chatterjee, T. A. Henzinger, J. Otop, and A. Pavlogiannis, “Distributed synthesis for LTL fragments,” in *FMCAD*, 2013, pp. 18–25.
- [36] B. Finkbeiner and S. Schewe, “Bounded synthesis,” *STTT*, vol. 15, no. 5–6, pp. 519–539, 2013.
- [37] K. Chatterjee and T. A. Henzinger, “Assume-guarantee synthesis,” *TACAS*, pp. 261–275, 2007.
- [38] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu, “Learning assumptions for compositional verification,” in *TACAS*, 2003, pp. 331–346.
- [39] W. Nam and R. Alur, “Learning-based symbolic assume-guarantee reasoning with automatic decomposition,” in *ATVA*, 2006, pp. 170–185.
- [40] K. Chatterjee, T. A. Henzinger, and B. Jobstmann, “Environment assumptions for synthesis,” in *CONCUR*, 2008, pp. 147–161.
- [41] W. Li, L. Dworkin, and S. A. Seshia, “Mining assumptions for synthesis,” in *MEMOCODE*, 2011, pp. 43–50.
- [42] R. Alur, S. Moarref, and U. Topcu, “Counter-strategy guided refinement of GR(1) temporal logic specifications,” in *FMCAD*, 2013, pp. 26–33.

- [43] —, “Pattern-based refinement of assume-guarantee specifications in reactive synthesis,” in *TACAS*, 2015. [Online]. Available: <https://www.cis.upenn.edu/~alur/Tacas15.pdf>
- [44] R. Könighofer, G. Hofferek, and R. Bloem, “Debugging formal specifications: A practical approach using model-based diagnosis and counterstrategies,” *STTT*, vol. 15, no. 5–6, pp. 563–583, 2013.
- [45] R. Bloem, R. Ehlers, S. Jacobs, and R. Könighofer, “How to handle assumptions in synthesis,” in *SYNT*, ser. EPTCS, vol. 157, 2014, pp. 34–50.
- [46] M. Abadi, L. Lamport, and P. Wolper, “Realizable and unrealizable specifications of reactive systems,” in *ICALP*, 1989, pp. 1–17.
- [47] Z. Manna and A. Pnueli, “A hierarchy of temporal properties (invited paper, 1989),” in *PODC*, 1990, pp. 377–410.
- [48] L. Lamport, “The temporal logic of actions,” *TOPLAS*, vol. 16, no. 3, pp. 872–923, 1994.
- [49] O. Lichtenstein, A. Pnueli, and L. Zuck, “The glory of the past,” in *Conference on logics of programs*, ser. LNCS. Springer, 1985, vol. 193, pp. 196–218.
- [50] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.
- [51] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa’ar, “Synthesis of reactive(1) designs,” *JCSS*, vol. 78, no. 3, pp. 911–938, 2012.
- [52] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [53] M. Abadi and L. Lamport, “The existence of refinement mappings,” *TCS*, vol. 82, no. 2, pp. 253–284, 1991.
- [54] A. W. Naylor and G. R. Sell, *Linear operator theory in engineering and science*. Springer, 1982.
- [55] W. Thomas, “On the synthesis of strategies in infinite games,” in *STACS*, 1995, pp. 1–13.
- [56] O. Lichtenstein, “Decidability, completeness, and extensions of linear time temporal logic,” Ph.D. dissertation, The Weizmann Institute of Science, Rehovot, Israel, November 1991.
- [57] R. McNaughton, “Infinite games played on finite graphs,” *Annals of Pure and Applied Logic*, vol. 65, no. 2, pp. 149–184, 1993.
- [58] V. Raman, “Explaining unsynthesizability of high-level robot behaviors,” Ph.D. dissertation, Cornell University, Ithaca, NY, 2013. [Online]. Available: <http://hdl.handle.net/1813/34373>
- [59] R. Alur, S. L. Torre, and P. Madhusudan, “Modular strategies for recursive game graphs,” *Theoretical Computer Science*, vol. 354, no. 2, pp. 230–249, 2006.
- [60] A. Banerjee and P. Dasgupta, “The open family of temporal logics: Annotating temporal operators with input constraints,” *TODAES*, vol. 10, no. 3, pp. 492–522, 2005.