Discrete Event Dyn Syst (2017) 27:261–299
DOI 10.1007/s10626-017-0239-8

Dynamic hierarchical reactive controller synthesis

Anne-Kathrin Schmuck[1] · Rupak Majumdar[1] · Adrian Leva[1]

Published online: 4 May 2017

Abstract In the formal approach to reactive controller synthesis, a symbolic controller for a possibly hybrid system is obtained by algorithmically computing a winning strategy in a two-player game. Such game-solving algorithms scale poorly as the size of the game graph increases. However, in many applications, the game graph has a natural hierarchical structure. In this paper, we propose a modeling formalism and a synthesis algorithm that exploits this hierarchical structure for more scalable synthesis. We define local games on hierarchical graphs as a modeling formalism that decomposes a large-scale reactive synthesis problem in two dimensions. First, the construction of a hierarchical game graph introduces abstraction layers, where each layer is again a two-player game graph. Second, every such layer is decomposed into multiple local game graphs, each corresponding to a node in the higher level game graph. While local games have the potential to reduce the state space for controller synthesis, they lead to more complex synthesis problems where strategies computed for one local game can impose additional requirements on lower-level local games. Our second contribution is a procedure to construct a dynamic controller for local game graphs over hierarchies. The controller computes assume-admissible winning strategies that satisfy local specifications in the presence of environment assumptions, and dynamically updates specifications and strategies due to interactions between games at different abstraction layers at each step of the play. We show that our synthesis procedure is sound: the controller constructs a play that satisfies all local specifications. We illustrate our results through an example controlling an autonomous robot in a building with known floor plan and provide simulation results using an implementation of our algorithm on top of LTLMoP.

✉ Anne-Kathrin Schmuck
akschmuck@mpi-sws.org

Rupak Majumdar
rupak@mpi-sws.org

Adrian Leva
aleva@mpi-sws.org

[1] Max Planck Institute for Software Systems, Kaiserslautern, Germany

Keywords Reactive synthesis · Hierarchical control · Decomposition · Game graphs · Abstraction · Compositional synthesis

1 Introduction

Algorithmic reactive synthesis has recently emerged as a robust methodology to design correct-by-construction controllers for specifications given in temporal logics (see, e.g., Girard and Pappas 2009; Tabuada 2009; Kloetzer and Belta 2008; Wolff et al. 2013; Wong et al. 2013). In this technique, one solves a two-player discrete-time game on a graph between the *system* and the *environment* players, where the winning condition is specified in linear-time temporal logic. The game graph is usually obtained as a discrete abstraction of the underlying, possibly continuous or hybrid, dynamics. A winning strategy for the system player in such a game can be computed by algorithmic techniques from reactive synthesis (Zielonka 1998; Emerson and Jutla 1991). Such a system winning strategy gives a discrete controller, which can usually be refined to a continuous controller using primitives from continuous control. This controller synthesis methodology has been implemented in symbolic tools (Wongpiromsarn et al. 2011; Mazo et al. 2010; Finucane et al. 2010) and was successfully applied in a number of case studies, e.g., by Wong et al. (2013) and Wongpiromsarn et al. (2010).

The major concern in the application of reactive synthesis to large problems is the poor scalability of game solving algorithms with increasing size of the game graph. In this paper, we address this challenge by extending the scope of reactive synthesis for control by (i) introducing *local game graphs over hierarchies* as a new decomposed model, (ii) formalizing *hierarchical reactive games* over such models, and (iii) proposing a sound *reactive controller synthesis algorithm* for such games. This algorithm allows for *dynamic specification changes* and uses the construction of *assume-admissible winning strategies* (Brenguier et al. 2015) to explicitly model and use environment assumptions.

Local game graphs over hierarchies The modeling formalism introduced in this paper allows to exploit the intrinsic *hierarchy* and *locality* of a given large-scale system. This decomposes the controller synthesis problem into multiple small ones. Here, hierarchy means that the game graph allows for the introduction of abstract layers. Locality means that a state at a higher layer naturally corresponds to a sub-arena of the game graph at the next lower layer which is independent from all the other games at the same layer.

As an example, consider an autonomous robot traversing the floors of a building. The lowest layer of the game graph, the game under consideration in existing reactive synthesis techniques, would consist of states defined by grids giving the location and velocity of the robot in each room and each floor of the building, together with additional predicates, such as the location of obstacles, whether the robot is carrying something, or the open-closed status of each door. However, there is a natural hierarchy of abstractions: at the highest layer, we care only about the floors and may ask the robot to move from one floor to another; in the next layer, we would like to know the specific room it is in and specify which room to go next, and only within the context of a room, we may care about where exactly the robot is and where it has to go next. To model this hierarchy, we introduce a set of layers on top of a game graph, each being a game graph itself, where a state at a higher layer (e.g. a room) corresponds to a sub-arena of the game graph at the next lower layer (i.e., all states located inside this room), modeling locality within the hierarchy.

Such hierarchical and local decompositions are also heuristically applied in robotics. Examples are general modeling frameworks, such as hierarchical task-networks (HTN) Erol et al. (1995) or Object-Action Complexes (OAC) Kruger et al. (2009), or particular software architectures for incorporating long term tasks and short time motion planning for robots (Kaelbling and Lozano-Perez 2011; Srivastava et al. 2014; Stock et al. 2015). One could view our abstraction layers, their interaction, and the system dynamics as an equivalent formalism to model task networks. Our controller synthesis algorithms should also apply to design controllers in these formalisms. To the best of our knowledge, the problem of correct-by-construction synthesis for temporal logic specifications (beyond reachability) in the presence of environment assumptions has not been considered by these other formalisms.

Hierarchical approaches for control exist for other correct-by-construction controller synthesis techniques in the control community, such as supervisory control (e.g., Schmidt et al. 2008), hybrid control (e.g., Raisch and Moor 2005), or continuous control (e.g., Pappas et al. 2000), but these can usually not handle temporal logic specifications.

In many large-scale projects using reactive controller synthesis, such as autonomous vehicles (Hess et al. 2014; Wongpiromsarn et al. 2012) and autonomous flight control (Koo and Sastry 2002), similar hierarchical and local decompositions are implicitly and informally performed. However, there is no clear theoretical model connecting "low-layer" reactive control and "higher layer" task planning in their work, which is provided by our approach.

Hierarchical reactive games To effectively use the constructed hierarchies of local game graphs for reactive controller synthesis, we assume that the specification is also decomposed into a set of local requirements, each restricted to one sub-arena of a particular layer, together with one "global" game at the highest layer. Such decompositions often arise naturally for large scale systems with intrinsic hierarchy and locality. For example, for the robot, one may consider the specifications: (i) a floor-layer task "visit all floors", (ii) a room-layer task "visit all rooms" for each floor, and (iii) a low layer task "if there is an empty bottle [in the current room], reach it and pick it up" for every room.

Synthesizing winning strategies for local games over hierarchies w.r.t. such sets of local specifications becomes challenging due to the interplay between layers both in a bottom-up and a top-down manner. The top-down interplay results because applying a strategy in a higher layer introduces additional specifications for the lower layer. For example, a requested move from one room to an adjacent one requires the local game in this room to fulfill a reachability specification in addition to its local specification. The bottom-up interplay results from the fact that moves in the lowest layer game correspond to moves in all higher layers which might change the strategy. For example, consider a room with two doors to two different adjacent rooms. The higher layer strategy may initially pick one door to continue. However, if this door gets closed before it was reached in the lower layer game, the higher layer strategy might ask to reach the second door instead. Thus, in each local game, winning objectives are generated *dynamically*, based on the strategy at a higher layer, the local specification for the local game and the current system and environment state in the lowest layer.

Our interactive hierarchical games were inspired by pushdown and modular games (Walukiewicz 1996; Alur et al. 2003; De Crescenzo and La Torre 2013), where the local state and the stack determine which (single) local game is played at a particular time point and correct and complete strategies are computed in a monolithic, non-dynamic fashion. In contrast, we always play one local game in every layer simultaneously, where visited states

in different layers are projections of one another. Therefore, a move in one layer has to be correlated with the games at all other layers at all time steps, giving the dynamic interaction described above.

Our work also relates naturally to abstraction and refinement techniques in game solving, (e.g., Cousot and Cousot 1977; Henzinger et al. 2000; Abadi and Lamport 1991), which map "concrete" game structures to "abstract" ones with more abstract timing, to solve a single game for a global specification using different abstraction layers. In comparison, we propose a hierarchical structure where every system state is refined to a whole new local sub-game, having its own specification. Therefore, the game in the higher layer does only proceed for one step once the lower layer local sub-game is completed. In this sense we are "stitching" together solutions of local games in the lowest layer in a particular way which is determined by higher level games, to obtain a solution to the global game.

Dynamical controller synthesis Given the hierarchical reactive games described above, we propose a reactive controller synthesis algorithm to solve such games that allows for *dynamic specification changes* at each step of the play. Intuitively, the controller solves the dynamically constructed local games online and "stitches" their solutions together following the rules of the hierarchical game. Notice that a strategy computed at a level imposes additional conditions on games at lower levels; thus, we use a dynamic controller synthesis algorithm that updates the strategies as the game progresses.

In principle, every algorithm that calculates a winning strategy for a two-player game can be used as a building block to solve local games (e.g., Zielonka 1998; Emerson and jutla 1991). However, classical reactive synthesis algorithms calculate winning strategies against *any* environment behavior. In most applications, such as our robot example, the requirement that the system wins against any environment strategy is too strong. For instance, in the robot example, it is possible, but very unlikely, that an employee keeps an office door closed forever to prevent the robot from fulfilling its task. This situation is addressed by algorithms that use assumptions on the environment behavior (see Bloem (2014, 2015) and Brenguier et al. (2015) for a detailed overview of recent results). These assumptions model "likely" behaviors of the environment to constrain the synthesis problem. Intuitively, the constrained synthesis problem then asks if the system can win provided that the environment only behaves according to its assumptions. While we can formulate our synthesis problem using any of these approaches incorporating environment assumptions, for concreteness, we use assume-admissible winning strategies introduced by Brenguier et al. (2015) as a building block in our algorithm.

We prove that, whenever the environment meets its assumptions and all dynamically generated local games have a solution, our dynamical synthesis algorithm generates a winning hierarchical play for a given specification, i.e., the algorithm is sound. If these assumptions do not hold, we show that the play gets stuck but does not violate the specification up to this point.

The dynamic nature of our controller is also similar to the receding horizon strategies proposed by Wongpiromsarn et al. (2012) and Vasile and Belta (2014) that translate long term goals into current local reachability specifications. This approach allows for a particular two-layer hierarchy and uses time horizons to decompose the synthesis problem locally. However, the general intrinsic hierarchical and local decomposability of a synthesis problem and the interaction of multiple abstract games is not formally exploited. In our presentation, our control synthesis algorithm solves local games completely; however, we can also use a receding horizon controller for each local game.

Implementation This paper was motivated by a systems project to build an end-to-end autonomous robotic telepresence system. For the scale of this model, existing reactive synthesis techniques would not work. However, the overall problem has a natural decomposition captured by our proposed model. While this paper focuses on the theoretical foundations of such a formal model and its reactive controller synthesis, we provide an implementation of our algorithm on top of LTLMoP (Finucane et al. 2010), an open source mission planning tool for robotic applications. In our implementation, we utilize the fact that our algorithm uses the solution of local games as a black-box building block. We can therefore treat every local game as a separate instance of LTLMoP. As LTLMoP synthesizes winning strategies for two player games w.r.t. specifications in the GR1 fragment of LTL (see Bloem et al. 2012 and Finucane et al. 2010 for details), our implementation currently only supports this class of specifications.

We show that our algorithm scales significantly better than the monolithic one in terms of computation time, if a large number of predicates (such as doors or obstacles) must be tracked globally, but only a small number of those predicates are relevant locally, that is, if they only need to be considered in a small subset of local games. We show that increasing the number of such predicates causes the monolithic solution to run out of memory very quickly while the computation time of the hierarchical synthesis is hardly affected.

2 Preliminaries

In this section we first introduce notation and recall existing results from reactive synthesis. Then we discuss a detailed example to motivate our work.

2.1 Reactive synthesis revisited

Notation For a set W, we denote by W^*, W^+, and W^ω the set of finite sequences, non-empty finite sequences, and infinite sequences over W, respectively. We write $W^\infty = W^* \cup W^\omega$. For $w \in W^*$, we write $|w|$ for the length of w; the length of $w \in W^\omega$ is ∞. We define $\mathrm{dom}(w) = 0, \ldots, |w| - 1\}$ if $w \in W^*$, and $\mathrm{dom}(w) = \mathbb{N}$ if $w \in W^\omega$. We denote by $\mathrm{dom}^+(w) = \mathrm{dom}(w) \backslash \{0\}$ the positive domain of w. For $k \in \mathrm{dom}(w)$ we write $w(k)$ for the kth symbol of w, $\lceil w \rceil = w(|w| - 1)$ for the last symbol of w, and $w|_{[0,k]}$ for the restriction of w to the domain $[0, k]$. Furthermore, $w \cdot w'$ for $w \in W^*$ and $w' \in W^\infty$ denotes the concatenation of two strings. The *prefix relation* on strings is defined by $w \sqsubseteq w'$ if $\exists w'' \in W^* \,.\, w \cdot w'' = w'$. Given a set of strings $\varphi \subseteq W^\infty$, we denote by $\overline{\varphi} = \varphi \cup \{w \in W^* \mid \exists w' \in \varphi \,.\, w \sqsubseteq w'\}$ the set of strings in φ and all their *finite* prefixes. Slightly abusing notation, we denote by $\overline{w}$ the set $\overline{\{w\}}$ of all prefixes of the string $w \in W^\infty$.

Two-player games A two-player *game graph* $G = (X, Y, \delta, \rho)$ between environment and system consists of a set of environment states X, a set of system states Y, an environment transition map $\delta : X \times Y \to 2^X$, and a system transition map $\rho : X \times Y \to 2^Y$. We assume G is serial, i.e., δ and ρ map each input to non-empty sets. A sequence $\pi \in (X \times Y)^\infty$ with $\pi(k) = (x(k), y(k))$ for all $k \in \mathrm{dom}(\pi)$ is called a *play* in G if

$$\forall k \in \mathrm{dom}^+(\pi) \cdot \begin{pmatrix} x(k) \in \delta\,(x(k-1), y(k-1)) \\ \wedge y(k) \in \rho\,(x(k), y(k-1)) \end{pmatrix}. \tag{1}$$

A play π is *finite* if $|\pi| < \infty$ and *infinite* otherwise. The set of all plays is denoted by $\mathcal{G}$.

We model a *winning condition* in a two-player game as a set of plays $\varphi \subseteq \mathcal{G}$. This set can be represented in different ways, e.g., by an LTL formula or by an ω-automaton. While our results do not assume a particular representation, the latter will determine the algorithm needed to solve the two-player game.

Given a game graph G, a set of initial strings $\mathcal{I} = (X \times Y)^+ \subseteq \mathcal{G}$ and a winning condition $\varphi \subseteq \mathcal{G}$, the tuple $(G, \mathcal{I}, \varphi)$ is called a *game* on G w.r.t. $\mathcal{I}$ and φ. A play $\pi \in \mathcal{G}$ is *winning* (resp. *possibly winning*) for $(G, \mathcal{I}, \varphi)$ if there exists an $n \in \mathrm{dom}(\pi)$ s.t. $\pi|_{[0,n]} \in \mathcal{I}$ and $\pi \in \varphi$ (resp. $\pi \in \overline{\varphi}$). We denote the set of all winning and possibly winning plays for $(G, \mathcal{I}, \varphi)$ by $\mathsf{WinningPlays}(G, \mathcal{I}, \varphi)$ and $\mathsf{WinningPlays}(G, \mathcal{I}, \overline{\varphi})$, respectively.

Strategies A *system strategy* is a *partial* function $f : (X \times Y)^+ \times X \rightharpoonup Y$ such that[1] $f(w, x) \in \rho(x, \lceil w \rceil_2)$ for all $(w, x) \in \mathrm{dom}(f)$; it is *memoryless* if $f(w, x) = f(\lceil w \rceil_2, x)$ for all $(w, x) \in \mathrm{dom}(f)$. An *environment strategy* is a function $g : (X \times Y)^+ \to X$ such that $g(w) \in \delta(\lceil w \rceil)$ for all $w \in (X \times Y)^+$. We denote the sets of system and environment strategies over G by $\mathcal{S}^s(G)$ and $\mathcal{S}^e(G)$, respectively. A play $\pi \in \mathcal{G}$ with $\pi(k) = (x(k), y(k))$ for all $k \in \mathbb{N}$ is *compliant* with $f \in \mathcal{S}^s(G)$, $g \in \mathcal{S}^e(G)$ and $\mathcal{I} = (X \times Y)^+ \subseteq \mathcal{G}$ if there is an $n \in \mathrm{dom}(\pi)$ such that $\pi|_{[0,n]} \in \mathcal{I}$ and for all $k \in \mathrm{dom}(\pi)$, $k > n$, we have

$$x(k) = g(\pi|_{[0,k-1]}) \quad \text{and} \quad y(k) = f(\pi|_{[0,k-1]}, x(k)). \tag{2}$$

The set of plays compliant with f, g and $\mathcal{I}$ is denoted by $\mathsf{CompliantPlays}(f, g, \mathcal{I})$ and we define $\mathsf{CompliantPlays}(f, \mathcal{I}) := \bigcup_{g \in \mathcal{S}^e(G)} \mathsf{CompliantPlays}(f, g, \mathcal{I})$.

A system strategy $f \in \mathcal{S}^s(G)$ is *winning* for $(G, \mathcal{I}, \varphi)$ against $g \in \mathcal{S}^e(G)$ if

$$\pi \in \mathsf{CompliantPlays}(f, g, \mathcal{I}) \cap (X \times Y)^\omega \Rightarrow \pi \in \mathsf{WinningPlays}(G, \mathcal{I}, \varphi) \text{ and} \tag{3a}$$

$$\pi \in \mathsf{CompliantPlays}(f, g, \mathcal{I}) \cap (X \times Y)^* \Rightarrow \exists \xi \in \mathcal{G} \,.\, \pi \cdot \xi \in \mathsf{CompliantPlays}(f, g, \mathcal{I}) \cap \mathsf{WinningPlays}(G, \mathcal{I}, \varphi). \tag{3b}$$

The set of winning strategies for $(G, \mathcal{I}, \varphi)$ against $g \in \mathcal{S}^e(G)$ is denoted by $\mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g)$ and we define $\mathsf{WinningStrategies}(G, \mathcal{I}, \varphi) = \bigcup_{g \in \mathcal{S}^e(G)} \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g)$. It should be noted that we have defined $\mathsf{CompliantPlays}(f, g, \mathcal{I})$ to be a set containing one infinite (or terminated finite) play and all its prefixes. The resulting definition of winning strategies ensures that aborted plays according to one particular strategy are always possibly winning. This is necessary to ensure correctness of our hierarchical control algorithm.

A system strategy f is *dominated* by a system strategy f' in the game $(G, \mathcal{I}, \varphi)$ (see Brenguier et al. 2014, Def.3) if for all $g \in \mathcal{S}^e(G)$

$$f \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g) \Rightarrow f' \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g)$$

holds, and there exists $g' \in \mathcal{S}^e(G)$ s.t. $f' \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g')$ and $f \notin \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g')$.

A system strategy that is not dominated is called *admissible*. The set of admissible strategies in the play $(G, \mathcal{I}, \varphi)$ is denoted by $\mathsf{AdmissibleStrategies}(G, \mathcal{I}, \varphi)$.

The synthesis problem The (unconstrained) synthesis problem takes as input a game $(G, \mathcal{I}, \varphi)$ and asks if there is a winning system strategy for the game. In most applications, the requirement that the system wins against any adversarial environment strategy

[1] Here, we write $\lceil w \rceil_2$ for the second component y of the pair $(x, y) \equiv \lceil w \rceil$.

is too stringent. The constrained synthesis problem additionally takes as input an assumption that models "likely" behaviors of the environment as a set of plays $\zeta \subseteq \mathcal{G}$. In the presence of environment assumptions, the synthesis problem looks for *assume-admissible winning strategies* for the system (see Brenguier et al. (2015) for a discussion why this is an appropriate notion).

By swapping the roles of system and environment we can equivalently define winning and admissible strategies for the environment in the game $(G, \mathcal{I}, \zeta)$ as before. Then a system strategy f is *assume-admissibly winning* for $(G, \mathcal{I}, \varphi)$ w.r.t. ζ (Brenguier et al. (2015), Rule AA) if

$$f \in \mathsf{AdmissibleStrategies}(G, \mathcal{I}, \varphi) \quad \text{and}$$
$$\forall g \in \mathsf{AdmissibleStrategies}(G, \mathcal{I}, \zeta) \cdot f \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g). \tag{4}$$

It should be noted that every winning strategy is assume-admissibly winning w.r.t. any assumption, but not vice-versa.

2.2 Example

To illustrate the theoretical results and their accompanying assumptions in this paper, we consider a robot that moves in a six story building with known floor plan, depicted in Fig. 1 (bottom) for floors 5 and 6.

To model this problem as a two-player game graph G, we partition the workspace into small cells which form a uniform grid. The resulting grid cells are enumerated by an index set Q. By assuming that the robot can only be in one grid cell at a time, the system state set is given by $Y = Q$. We furthermore define the set of environment states by $X = 2^Q$, where a state $x \in X$ is a *set* containing all grid cells that are currently occupied by an obstacle.

This modeling formalism implies that each grid cell in Fig. 1 (bottom) represents a system state. We model additional properties by adding other binary variables. For example, by adding a predicate `Bottle` to the system state, we model whether the robot is carrying

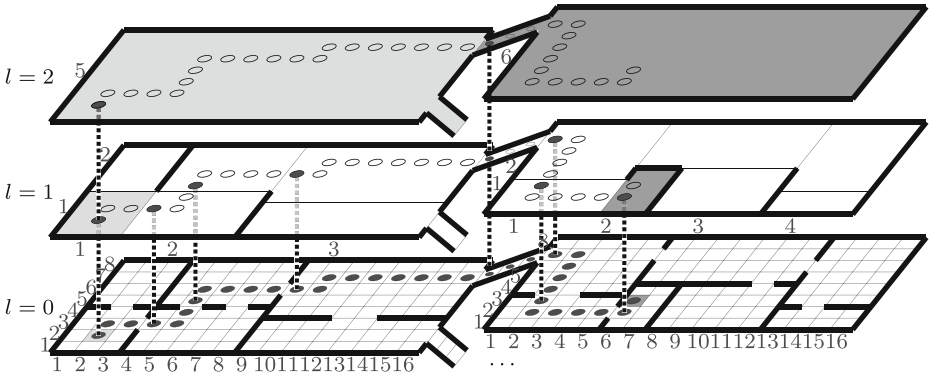

Fig. 1 Floor plan of the 5th and 6th floor of a six-story building. Using the depicted coordinates, we denote by q_{ij}^k and r_{ij}^k, respectively, the cell and the room in the ith column and jth row of floor k. Furthermore, s_{ij}, $i < j$ denotes the stair case from floor f^i to floor f^j. The workspace of this building is partitioned into grid cells (*bottom*), rooms (*middle*) and floors (*top*) which serve as abstraction layers $l = 0$ to $l = 2$ as discussed in Section 2.2. The line of dots depicts a path of the robot from the initial state (*light gray*) to the final state (*dark gray*) in every layer. Filled circles denote projected states while non-filled circles denote abstract (but not projected) states, as discussed in Expl. 2-3

a bottle or not. As this additional variable might be true in any grid cell, the resulting system state set would consist of two copies of the grid world in Fig. 1 (bottom), where one is annotated with `Bottle` and the other one is not. To keep notation simple, such additional predicates are mostly neglected in this example.

The system transition map ρ in G results from applying an appropriate abstraction method for continuous dynamics, e.g., Tabuada (2009), while adding the obvious restrictions that (i) the robot cannot move into an obstacle-occupied cell, and (ii) the robot can only move to adjacent cells that are not separated by a wall. For the environment transition map δ several levels of detail can be used to model the movement and (dis)appearance of obstacles, see e.g., Wong et al. (2013) and Vasile and Belta (2014) for examples.

Now consider a task for the robot which asks it to reach a specific room on a specific floor. This corresponds to a *reachability* winning condition. In our setting, the winning condition is captured by the language of all plays π such that there exists $k \geq 0$ with $\pi(k) = (x(k), y(k))$ and $y(k)$ is a cell in the specified room. (It can easily be described in linear temporal logic as well.) The synthesis problem for this specification over the game graph G finds a strategy (a controller for the robot) that ensures that the robot eventually reaches the room.

There are two challenges in applying reactive synthesis in this scenario. First, the requirement that the robot must reach the room against all possible environments is too stringent. In such a robot motion example the environment player naturally has a very rich set of possible moves. For the specification considered above, the environment can simply keep a couple of doors closed forever to prevent the robot to reach its goal. However, this adversarial behavior is very unlikely in a real world application as, e.g., employees in an office building will always eventually visit/exit their office. This is the reason why we introduce environment assumptions that constrain the problem. A natural environment assumption allowing to realize the above specification models that all staircases are always eventually unblocked, all doors get always eventually re-opened, and moving obstacles always eventually allow a passage to exit a room.

As discussed in Brenguier et al. (2014), one cannot simply perform reactive synthesis w.r.t. environment assumptions by considering the implication $\zeta \Rightarrow \varphi$ that requires the controller to ensure φ holds only on plays satisfying ζ. This is because the robot may win the game by simply violating the environment assumption (for example, by blocking a door and preventing the environment from opening it). Thus, we consider assume-admissible strategies in this paper.

The second challenge is that of scalability. In any realistic model of our problem, the number of states is so large that existing reactive synthesis tools do not scale. Our main contribution in this paper is to scale up reactive synthesis techniques by considering *local* structure. We now consider this in more detail.

As depicted in Fig. 1, there is a natural hierarchy on the states of the workspace imposed by rooms and floors. That is, the workspace can also be partitioned using the set of rooms R or the set of floors F as index sets.[2] This partition introduces two abstraction layers with decreasing precision with system state sets $Y^1 = R$ and $Y^2 = F$. The set of environment states in layers 1 and 2 are defined as the set of closed doors $X^1 = 2^D$ and the set of blocked

[2]For simplicity we model the stairs as a separate room and always "attach" the downward stairs to the respective floor.

staircases $X^2 = 2^S$, respectively. Even though the three layers in Fig. 1 are constructed separately, there is a natural abstraction relation between system states $f \in F$, $r \in R$, and $q \in Q$. A system state q is obviously related to the system state r if the grid cell q is "inside" room r. Furthermore, a door d is marked as `closed` if all cells intersecting with this door are occupied by an obstacle (usually being the door itself in this case), inducing a relation between environment states of layers 0 and 1. In Section 3, we present *abstract game graphs* (AGGs) which capture such hierarchies in reactive games.

The abstraction relations naturally decompose every layer in the example into small, local game graphs located "inside" a higher level system state: the game graph G is decomposed in local game graphs G_r, $r \in R$. This is possible for this example as the set of possible moves in one room is independent from the part of the environment state that does not belong to this context, e.g., all the obstacles contained in the set x that are not located inside this room. In Section 4, we introduce *local game graphs* (LGGs) which decompose AGGs to model this locality within the hierarchy.

To exploit this local structure in reactive synthesis, we additionally require that the specification is also given as a set of local specifications, one for each local game; otherwise, there is no obvious way to automatically break a global specification into local synthesis problems. For example, for the reachability task, one can consider a specification of reaching a room at the higher layer, and reaching from one point of a room to a prescribed exit point in the lower layer. Correspondingly, notice that the environment assumptions can also be decomposed into layers.

Now consider the task:

Collect all empty bottles in the building and return them to the kitchen in the 5th floor.

This task can be manually decomposed in a natural fashion as follows. The level 2 task asks the robot to visit all floors of the building and to return to floor 5 whenever its capacity to carry empty bottles is reached. While in one floor, the level 1 task asks the robot to visit all rooms until the carrying capacity is reached, and to visit the kitchen whenever the latter is true and the robot is in floor 5. Finally, the level 0 tasks ask the robot to search for empty bottles in a single room, approach each bottle and pick it up. In this paper we assume that both the system specification and the environment assumptions are already given in a decomposed manner. The automatic decomposition of a global winning condition into local ones is an orthogonal, difficult, problem.

In Section 4.2, we define *hierarchical reactive games* (HRGs) by combining the set of LGGs over hierarchies with a set of local winning conditions and a set of local environment assumptions. This generates a set of local games over an LGG w.r.t. a local specification φ and a local assumption ζ.

The main challenge for reactive synthesis for HRGs is that the games played at the various layers interact. That is, a strategy at a higher layer ("go to the kitchen") introduces additional constraints at the lower layer ("the higher level strategy requires that the robot should go to the exit that takes it to the kitchen"). In Section 5, we provide a synthesis algorithm that computes a dynamic controller for HRGs. The controller computes assume-admissible strategies for each local game, and dynamically updates the winning conditions and strategies through the hierarchy. We prove that the algorithm is sound and that it aborts the game only when a local subgame cannot be won by the system against admissible strategies of the environment.

3 Hierarchical decomposition

We now introduce a hierarchy of L two player game graphs where the higher layers are a more abstract representation of the original game graph at layer $l = 0$.

3.1 Layering, abstract plays, and timescales

Let $G = (X, Y, \delta, \rho)$ be a game graph. A sequence $\langle X^0, Y^0\rangle, \langle X^1, Y^1\rangle, \ldots, \langle X^L, Y^L\rangle$ is a *layering* of G if (i) $X^0 = X$ and $Y^0 = Y$, and (ii) for each $l \in [1, L]$, there exist *abstraction functions* $\alpha_s^l : Y^{l-1} \to Y^l$ and $\alpha_e^l : \left(X^{l-1} \times Y^{l-1}\right) \to X^l$.

Notice that while the system abstraction function maps system states at level $l-1$ to system states at level l, the environment abstraction function α_e^l maps a pair (x, y) of environment and system states at level $l-1$ into an environment state at level l. This allows us to incorporate the loss of direct control with increasing abstraction level, as illustrated in the following example.

Example 1 Consider the robot in Section 2.2 and assume that the system states of layer 0 are extended by the binary variable `Bottle`, resulting in the state $\{q, \texttt{Bottle}\}$ if the robot is in cell q and carries a bottle and the state $\{q\}$ if the latter is not true. In this example, a transition from state $\{q\}$ to $\{q, \texttt{Bottle}\}$ is enforceable in layer 0 if there is a bottle in cell q (which can be modeled by a corresponding environment variable) assuming that the robot can always pick up a bottle when it is in this cell.

Now assume that the specification in the room level (layer 1) asks the robot to go to the kitchen if it is carrying a bottle. If the information about bottle locations is not available in layer 1, the strategy in layer 1 cannot *enforce* the robot to pick up a bottle in a particular room but can only *observe* that the latter happened by an appropriate projection of the level 0 system state. This intuition can only be modeled if `Bottle` is included in the environment states rather than the system states of layer 1. To be able to trigger this environment variable in layer 1 when the robot picks up a bottle, the tuple $(x, \{q, \texttt{Bottle}\}) \in X^0 \times Y^0$ must be projected to an environment state $\{\texttt{Bottle}\} \cup x' \in X^l$ using the map α_e^1.

For notational convenience, we define the composition of abstraction functions $\alpha_e^{l\uparrow} : (X \times Y) \to X^l$ and $\alpha_s^{l\uparrow} : Y \to Y^l$ recusively for all $l \in [1, L]$ as

$$\forall y \in Y \,.\, \alpha_s^{l\uparrow}(y) = \alpha_s^l\left(\alpha_s^{l-1\uparrow}\right), \tag{5a}$$

$$\forall x \in X, y \in Y \,.\, \alpha_e^{l\uparrow}(x, y) = \alpha_e^l\left(\alpha_e^{l-1\uparrow}(x, y), \alpha_s^{l-1\uparrow}(y)\right), \tag{5b}$$

where $\alpha_e^{0\uparrow}(x, y) = x$ and $\alpha_s^{0\uparrow}(y) = y$.

A layering induces an abstraction for a play $\pi \in \mathcal{G}$ for each layer $l > 0$ as follows. Given a game G, a play $\pi \in \mathcal{G}$, and layers $\langle X^l, Y^l\rangle_{l=0}^L$ with abstraction functions α_e^l and α_s^l, we define the set of *abstract plays* $\Pi = \{\pi^l\}_{l=0}^L$ of π by $\pi^l \in (X^l \times Y^l)^\infty$ with $\pi^l(k) = (x^l(k), y^l(k))$ s.t.

$$\forall k \in \mathrm{dom}^+(\pi) \,.\, \begin{pmatrix} x^l(k) = \alpha_e^{l\uparrow}(x(k), y(k-1)) \\ \wedge y^l(k) = \alpha_s^{l\uparrow}(y(k)) \end{pmatrix} \tag{6}$$

and $\pi^l(0) = (\alpha_e^{l\uparrow}(x(0), y(0)), \alpha_s^{l\uparrow}(y(0)))$.

Intuitively, the abstract plays in Π are an abstraction of the play π which becomes coarser the higher the layer, as multiple system and environment states are clustered into one state in a higher level. Specifically, this implies that state changes occur less frequently in a higher level than in the play π as outlined in the following example.

Example 2 Consider the path of the robot depicted by filled cycles in Fig. 1 (bottom). This path represents the system state component y of a play $\pi \in \mathcal{G}$. Applying the second line of (6), this sequence y can be abstracted to layer $l = 1$ and $l = 2$ as follows.

$$\begin{aligned} y &= \ q_{22}^5\ q_{23}^5\ q_{33}^5\ q_{43}^5\ q_{53}^5\ q_{54}^5\ q_{55}^5\ q_{56}^5\ \cdots \\ y^1 &= r_{11}^5\ r_{11}^5\ r_{11}^5\ r_{21}^5\ r_{21}^5\ r_{21}^5\ r_{22}^5\ r_{22}^5\ \cdots \\ y^2 &= f^5\ \ f^5\ \ f^5\ \ f^5\ \ f^5\ \ f^5\ \ f^5\ \ f^5\ \ \cdots \end{aligned}$$

The abstract sequences y^1 and y^2 are depicted in Fig. 1 (middle) and (top), respectively. The state changes in levels 1 and 2 correspond to changes in rooms and floors, respectively. While the state at level 0 changes in each time step, observe that state transitions in layers 1 and 2 only happen irregularly and not at every time point. It should be noted that environment states in layer 1 and 2, i.e., the set of closed doors and blocked stairs, changes independently from system state changes and is not illustrated in Fig. 1.

Expl. 2 illustrates that an abstract play π^l is usually not turn-based. To obtain a turn-based game and to remove redundant information, we introduce a new time scale for every layer which is triggered by changes in the system states in an abstract game π^l as follows. Given a play $\pi \in \mathcal{G}$ and a layer $l \in [0, L]$, the *timescale transformation* κ^l of π in layer l is the identity function if $l = 0$, and defined by the strictly monotone sequence $\kappa^l \in \mathbb{N}^\infty$ s.t.

$$\kappa^l(0) = 0, \tag{7a}$$

$$\forall\, m \in \mathrm{dom}(\kappa), m > 0, k \in [\kappa(m-1), \kappa(m)) \,.\, y^l(k) = y^l(\kappa^l(m-1)) \neq y^l(\kappa^l(m)) \tag{7b}$$

$$\text{and } \forall k > \lceil \kappa^l \rceil \,.\, y^l(k) = y^l(\lceil \kappa^l \rceil), \tag{7c}$$

otherwise. The set of *projected plays* $\breve{\Pi} = \{\breve{\pi}^l\}_{l=0}^L$ of π with $\breve{\pi}^l = (\breve{x}^l, \breve{y}^l)$ is defined as the sub-sequence of the abstract play π^l at time points given by κ^l for every $l \in [1, L]$. Formally,

$$\forall k \in \mathrm{dom}(\kappa^l) \,.\, \breve{\pi}^l(k) = \pi^l(\kappa^l(k)). \tag{8}$$

Informally, the timescale transformation "projects" the game on to a more abstract layer and removes the stuttering steps introduced by many contiguous steps in the concrete game all corresponding to the same abstract state.

A projected play $\breve{\pi}$ is called *infinite* if $|\breve{\pi}| = \infty$ and *finite* otherwise. While plays $\pi \in \mathcal{G}$ can always be made infinite (by the serial assumption on the transition relations), its projection $\breve{\pi}^l$ to layer $l > 0$ need not be infinite. For example, if the robot from Section 2.2 should just move within room r_{11}^5, this obviously induces an infinite play π. However, its projection to the room layer is given by $\breve{\pi}^1 = r_{11}^5$, i.e., $\breve{\pi}^1$ is finite with length 1.

Example 3 Consider the abstract sequences y^1 and y^2 in Expl. 2. Using (7a) and (8) their induced time scale transformations are given by

$$\kappa^1 = 0\ 3\ 6\ \ldots \quad \text{and} \quad \kappa^2 = 0\ 20$$

and the resulting projections for layer 1 and 2 are given by

$$\breve{y}^1 = r_{11}^5\, r_{12}^5\, r_{22}^5 \ldots \quad \text{and} \quad \breve{y}^2 = f^5\, f^6$$

corresponding to changes in rooms and floors respectively at those times. In Fig. 1, system states of projected plays are depicted by filled circles, whereas states only belonging to abstract plays are depicted by non-filled cycles.

It can be easily shown (see Lem. 1 in Appendix A) that the range of the timescale transformation κ^{l+1} is a subset of the range of κ^l; if there is an event at the $(l+1)$st layer, there is a corresponding event at the lth (and so, in each lower) layer. Using this observation we can simplify notation by defining

$$\kappa_l^{l+1}(k) := (\kappa^l)^{-1}(\kappa^{l+1}(k)) \tag{9}$$

to denote the time position in the lth layer of the kth event in the $(l+1)$st layer.

3.2 Abstract game graphs

Using the notion of abstract states and plays from the previous section, we now construct game graphs for every layer l. We remark that the actual game is only played in the lowest layer, i.e., in the game graph G, and the higher layers only model projected plays of this game.

Definition 1 Let $G = (X, Y, \delta, \rho)$ be a game graph, and $\langle X^l, Y^l \rangle_{l=0}^L$ a layering of G using the abstraction functions α_e^l and α_s^l. Then we define the set of *abstract game graphs* (AGG) $\{G^l\}_{l=0}^L$ for each layer $l \in [1, L]$ by $G^l := (X^l, Y^l, \delta^l, \rho^l)$ s.t.

$$x' \in \delta^l(x, y) \Leftrightarrow \left(\exists \pi \in \mathcal{G}, y' \in Y^l \cdot \begin{pmatrix} \pi^l(\kappa^l(0)) = (x, y) \\ \wedge \exists k \in (0, \kappa^l(1)] \,.\, \pi^l(k) = (x', y') \end{pmatrix} \right) \tag{10a}$$

$$y' \in \rho^l(x, y) \Leftrightarrow \left(\exists \pi \in \mathcal{G}, x' \in X^l \cdot \begin{pmatrix} \pi^l(\kappa^l(1) - 1) = (x', y) \\ \wedge \pi^l(\kappa^l(1)) = (x, y') \end{pmatrix} \right). \tag{10b}$$

and for $l = 0$ by $G^0 := G$.

Intuitively, the maps δ^l and ρ^l collect all transitions that can occur in projected plays $\breve{\pi}^l$ of possible lowest level plays $\pi \in \mathcal{G}$, as illustrated in the following example. It should be noted that all lowest level plays π are existentially quantified in (10a), i.e., all possible plays in the lowest layer are considered.

Example 4 Consider the play $\pi \in \mathcal{G}$ and its abstract play π^1 depicted in Fig. 2. The existence of the play π introduces the depicted system and environment transitions using (10a) and (10b), respectively. Observe that the construction considers every environment change (induced by the play π) as an environment transition from the environment state at the last triggering instance indicated by κ. Furthermore, system transitions are only generated at those triggering instances. It can be seen in Fig. 2 that the environment state in layer $l > 0$ possibly changes multiple times before a system state change follows.

The construction in Def. 1 allows us to prove that projected plays $\breve{\pi}^l$ as defined in (8) are also plays in the game graph G^l, i.e., $\breve{\pi}^l \in \mathcal{G}^l$. Intuitively, the proof shows that there always exist transitions, as the ones emphasized in Fig. 2, connecting system and environment states at triggering instances.

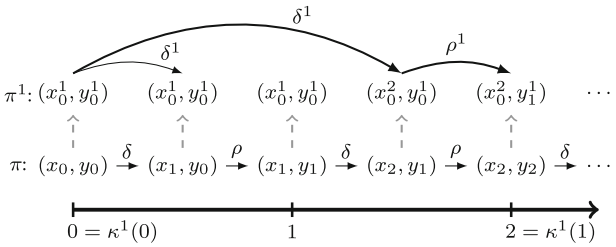

Fig. 2 Generation of system and environment transitions for layer $l = 1$ from a play π as formalized in Def. 1 and discussed in Expl. 4

Proposition 1 *For any game G, any play $\pi \in \mathcal{G}$, and any $l \in [0, L]$, we have that $\breve{\pi}^l$ is a play in G^l, i.e., $\breve{\pi}^l \in \mathcal{G}^l$.*

Proof The claim follows directly from Lem. 2 in Appendix A as (1) holds for $\breve{\pi}^l$ and G^l when we pick $n = \kappa^l(m + 1)$ in (35). □

4 Context-based decomposition

A set of AGGs imposes an abstraction hierarchy on top of a given game graph G. However, AGGs by themselves are not enough to decompose a synthesis problem. For example, if the winning condition is given by a set of plays on the lowest layer, the induced abstraction layers cannot be exploited by a synthesis algorithm. In order to derive an efficient synthesis technique, in this section, we introduce the second ingredient: *local* winning conditions, which induce *local game graphs*.

Roughly, a *local* winning condition for the game G^l at layer l is a set of abstract plays π^l whose states belong to a single state at layer $l + 1$. For example, reaching a different floor is a local specification at layer 2. A synthesis procedure to enforce φ^L would require solving games at lower levels; in our example, the robot will have to successively reach a set of rooms, followed by the stairs to achieve its goal. Each of these "lower level" games occur in, roughly, the "local" game structure defined by states in the lower level that map to the current state of the higher level. We formalize this notion as *local game graphs*.

4.1 Local game graphs over hierarchies

Fix a layer l and consider the games G^l and G^{l+1}. Consider a system state $\nu \in Y^{l+1}$. A first attempt to define a local game is to restrict the game G^l to the set of system states $\{y \in Y^l \mid \alpha_s^{l+1}(y) = \nu\}$. However, this is not sufficient, because plays in the local game should be allowed to leave the region specified by ν for one step at the end. This is necessary to ensure that plays in consecutive local games can be concatenated to form a play over the game graph G^l without formalizing a special reset action, as e.g., used in modular games by Alur et al. (2003). To account for these states, we introduce the Post operation:

$$\mathrm{Post}^l(\nu) := \left\{ \nu' \in Y^l \mid \begin{pmatrix} \nu' \neq \nu \\ \wedge \exists x \in X^l \,.\, \nu' \in \rho^l(x, \nu) \end{pmatrix} \right\}. \tag{11}$$

Including the one-step post states allows us to view the actual game as a layer 0 game and use the hierarchical and local decompositions as modeling formalism for hierarchical controller synthesis only.

Considering environment states instead of system states, a straightforward restriction to a context ν is not naturally given by $\alpha_e^{l+1\uparrow}$, as the following example shows.

Example 5 Consider the example from Section 2.2 and its floor plan depicted in Fig. 3. Recall from Section 2.2 that an environment state $x \in X^0$ contains all grid cells that are occupied by an obstacle. However, by playing a game in room r_{11}^5 one is only interested in obstacles that are located inside $Y^0{}_{r_{11}^5}$.

Therefore, instead of using $\alpha_e^{l+1\uparrow}$ to restrict X^l to context ν, we use a restricting function $\mathfrak{r}_\nu^l : X^l \to X^l{}_\nu$, where $X^l{}_\nu$ is the set of environment states at layer l restricted context ν. For Expl. 5, the map $\mathfrak{r}^1_{r_{11}^5}$ simply maps the set x of obstacle locations to the subset $x' \subseteq x$ of such locations that are inside the striped area in layer 0 of Fig. 3. For notation convenience, we define $\mathfrak{r}^L$ as the identity map. Using the above intuition, we define *local game graphs* as follows.

Definition 2 Given an AGG G^l, the *local* game graph (LGG) $G^l_\nu := \left(X^l_\nu, Y^l_\nu, \delta^l_\nu, \rho^l_\nu\right)$ at layer l restricted to $\nu \in Y^{l+1}$ consists of

$$X^l_\nu := \left\{ \mathfrak{r}^l_\nu(x) \,\middle|\, x \in X^l \right\} \quad \text{and} \tag{12a}$$

$$Y^l_\nu = Y^l_{\nu\rceil} \cup Y^l_{\nu\lfloor} \tag{12b}$$

$$s.t. \quad Y^l_{\nu\rceil} := \{ y \in Y^l \mid \nu = \alpha_s^{l+1}(y) \} \quad \text{and} \tag{12c}$$

$$Y^l_{\nu\lfloor} := \left\{ y' \in Y^l_{\nu'\rceil} \,\middle|\, \begin{pmatrix} \nu' \in \ \mathrm{Post}^l(\nu) \\ \wedge \exists y \in Y^l_{\nu\rceil}, x \in X^l{}_\nu \ . \ y' \in \rho^l(x, y) \end{pmatrix} \right\}, \tag{12d}$$

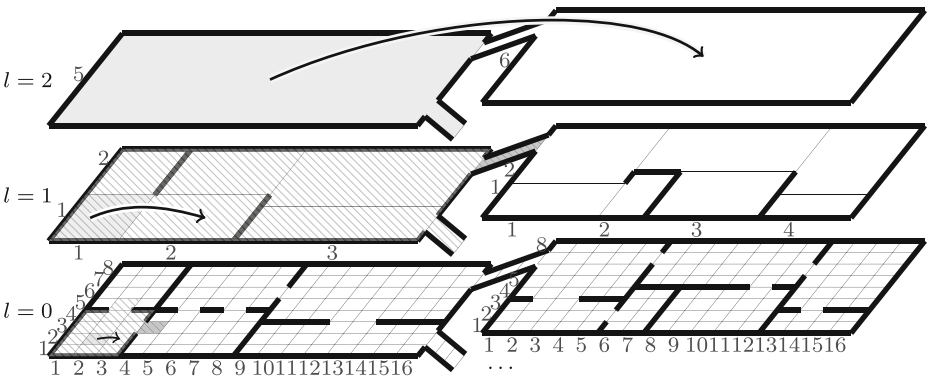

Fig. 3 Floor plan from Fig. 1. The striped areas in layers 0 and 1 correspond to $Y^0{}_{r_{11}^5}$ and $Y^1{}_{f^5}$, respectively. The three arrows denote context changes requested by layer l which induce a reachability specification for layer $l-1$ whose initial and goal states are depicted in light and dark gray, respectively

and transition maps $\delta^l_\nu : {X^l}_\nu \times Y^l_{\nu\rceil} \to 2^{{X^l}_\nu}$ and $\rho^l_\nu : {X^l}_\nu \times Y^l_{\nu\rceil} \to 2^{Y^l_\nu}$ s.t.

$$\left(x' \in \delta^l(x, y) \wedge y \in Y^l_{\nu\rceil}\right) \Rightarrow \mathfrak{r}^l_\nu(x') \in \delta^l_\nu\left(\mathfrak{r}^l_\nu(x), y\right) \tag{13a}$$

and

$$\left(y' \in \rho^l(x, y) \wedge y \in Y^l_{\nu\rceil} \wedge y' \in Y^l_\nu\right) \Rightarrow y' \in \rho^l_\nu\left(\mathfrak{r}^l_\nu(x), y\right). \tag{13b}$$

We write $[\mathbb{G}] := \left\{\{G^l_\nu\}_{\nu \in Y^{l+1}}\right\}_{l=0}^{L-1} \cup \{G^L\}$ for the set of LGGs over G.

Example 6 Consider the example from Section 2.2 and its floor plan depicted in Fig. 3. The striped areas in layers 0 and 1 correspond to the context restricted system state sets ${Y^0}_{r^5_{11}}$ and ${Y^1}_{f^5}$, respectively. It is easy to see that $Y^0_{r^5_{11}\lfloor} = \{q^5_{25}, q^5_{43}\}$ and $Y^1_{f^5\lfloor} = \{s_{56}\}$, while layer $l = 2$ is not decomposed.

In the robot example of Section 2.2 the generated set of LGGs is "truly local" in the sense that the local system dynamics do not depend on environment variables from other contexts. E.g., an obstacle in another room r' does not influence the dynamics of the robot in room $r \neq r'$. This inherent decomposability of the system dynamics, similar to the natural relations among states of different layers, is a feature of the system we want to control which is necessary for the subsequently proposed synthesis algorithm and formalized in the following assumption.

Assumption 1 For every layer $l \in [0, L-1]$ and context $\nu \in Y^{l+1}$ it holds for all $x \in X^l$ and $y \in Y^l_{\nu\rceil}$ that

$$y' \in \rho^l(x, y) \Rightarrow y' \in \rho^l\left(\mathfrak{r}^l_\nu(x), y\right). \tag{14}$$

It should be noted that the right hand side of (14) uses ρ^l instead of ρ^l_ν. Therefore, $\rho^l_\nu \subseteq \rho^l$ if Ass. 1 holds, which implies that in this case (13a) holds in both directions.

Similarly to Prop. 1 we can prove that the part of a play π^l that takes place in context ν is actually a play in G^l_ν. However, to formalize this we need to define *local plays* that are projected to the current context. Given a set of LGGs $[\mathbb{G}]$, a play $\pi \in \mathcal{G}^0$ and its sets of abstract and projected plays Π and $\breve{\Pi}$, the *local restriction* of π^l and $\breve{\pi}^l$ is defined for all $m \in \mathrm{dom}^+(\breve{\pi}^l)$ by

$$\pi^l_\downarrow(m) := (x^l_\downarrow(m), y^l(m)) \quad \text{with} \quad \pi^l_\downarrow(m) := \mathfrak{r}^l_{y^{l+1}(\kappa^l(m)-1)}(x^l(m)) \text{ and} \tag{15a}$$

$$\breve{\pi}^l_\downarrow(m) := (\breve{x}^l_\downarrow(m), \breve{y}^l(m)) \quad \text{with} \quad \breve{\pi}^l_\downarrow(m) := \mathfrak{r}^l_{y^{l+1}(\kappa^l(m)-1)}(\breve{x}^l(m)). \tag{15b}$$

The restriction of $x^l(m)$ (resp. $\breve{x}^l(m)$) at time $k = \kappa^l(m)$ is defined w.r.t. the last system state $y^{l+1}(k-1)$ as $y^{l+1}(k)$ is only available after the next system move that is depended on $x(k)$. The local restriction $\breve{\pi}^l_\downarrow$ of the projected play introduces a sequence $\breve{p}^l_\downarrow$ of local projected plays defined by

$$\forall m \in \mathrm{dom}^+(\breve{\pi}^{l+1}) \,.\, \breve{p}^l_\downarrow(m-1) := \breve{\pi}^l_\downarrow\Big|_{\left[\kappa^{l+1}_l(m-1), \kappa^{l+1}_l(m)\right]} \text{ and} \tag{16a}$$

$$\breve{p}^l_\downarrow(\mathsf{end}(\breve{\pi}^{l+1})) = \lceil \breve{p}^l_\downarrow \rceil := \breve{\pi}^l_\downarrow\Big|_{\left[\lceil \kappa^{l+1}_l \rceil, \mathsf{end}(\breve{\pi}^l)\right]}, \tag{16b}$$

where $\mathsf{end}(w) = |w| - 1$ denotes the time of the last element of w. We write $[\check{p}]_\pi := \left\{\check{p}^l_\downarrow\right\}_{l=0}^{L-1} \cup \left\{\check{p}^L_\downarrow\right\}$ for the set of all such sequences induced by π, where $\check{p}^L_\downarrow(0) = \check{\pi}^L$ and $\mathsf{end}\left(\check{p}^L_\downarrow\right) = 0$.

Example 7 Consider the play π whose y-component is depicted by filled cicles in Fig. 1 (bottom). For illustration purposes, assume a static environment with a closed door between room r^5_{11} and r^5_{12}, denoted by the binary variable d, and an obstacle in q^5_{63}. The closed door, which is an environment variable for layer 1, corresponds to obstacles in q^5_{24} and q^5_{25} for layer 0. For this play, the local plays contained in the set $[\check{p}]_\pi$ are given by the following strings.

$$
\begin{aligned}
\check{p}^0_\downarrow(0) &= \left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{22}\right)\left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{23}\right)\left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{33}\right)\left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{43}\right)\\
\check{p}^0_\downarrow(1) &= \left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{43}\right)\left(\left\{q^5_{63}\right\}, q^5_{53}\right)\left(\left\{q^5_{63}\right\}, q^5_{54}\right)\left(\left\{q^5_{63}\right\}, q^5_{55}\right)\\
&\vdots\\
\check{p}^0_\downarrow(7) &= \left(\{\perp\}, q^6_{62}\right)\left(\{\perp\}, q^6_{63}\right)\\
\check{p}^1_\downarrow(0) &= \left(\{d\}, r^5_{11}\right)\left(\{d\}, r^5_{21}\right)\left(\{d\}, r^5_{22}\right)\left(\{d\}, r^5_{32}\right)(\{d\}, s_{56})\\
\check{p}^1_\downarrow(1) &= (\{d\}, s_{56})\left(\{\perp\}, r^6_{12}\right)\left(\{\perp\}, r^6_{11}\right)\left(\{\perp\}, r^6_{21}\right)\\
\check{p}^2_\downarrow(0) &= (\{\perp\}, f^5)(\{\perp\}, f^6).
\end{aligned}
$$

where $\{\perp\}$ denotes that no obstacles are present. Due to the definition of Y^l_ν in Def. 2, contexts of neighboring cells overlap. This is also visible by the above local plays, which overlap for one time instant. E.g, the state $\left(\left\{q^5_{24}, q^5_{25}\right\}, q^5_{43}\right)$ belongs both to $\check{p}^0_\downarrow(0)$ and $\check{p}^0_\downarrow(1)$, which are the local plays in context $Y^0_{r^5_{11}}$ and $Y^0_{r^5_{21}}$, respectively. As we use the convention that the environment moves first, the environment variables of such overlapping states are always restricted to the context that is currently left.

Proposition 2 *Let $[\mathbb{G}]$ be a set of LGGs and $\mathcal{G}^l_y$ the set of plays in G^l_y. Furthermore, let $\pi \in \mathcal{G}$ and $[\check{p}]_\pi$ its induced set of local projected play sequences. Then it holds for all $l \in [0, L-1]$ and $m \in \mathrm{dom}(\check{\pi}^{l+1})$ that*

$$\check{p}^l_\downarrow(m) \in \mathcal{G}^l_{\check{y}^{l+1}}(m). \tag{17}$$

Proof Equation (17) follows by combining the last lines of (36a) and (36b) in Lem. 3 proven in Appendix A. □

4.2 Hierarchical reactive games over sets of LGGs

We have seen in the example of Section 2.2 that the motivation for constructing LGGs comes from the natural decomposability of system dynamics, environment assumptions and tasks into local and global components which are naturally restricted to a context $\nu \in Y^{l+1}$. Recall that local specifications should intuitively only contain finite strings to eventually allow progress in the higher layer upon completion of the local task. This observation is

formalized as follows. Given a set $[\mathbb{G}]$ of LGGs, layer $l \in [0, L-1]$, and context $\nu \in Y^{l+1}$, the sets

$$\varphi_\nu^l \subseteq \left(X_\nu^l \times Y_{\nu\rceil}^l\right)^* \cap \mathcal{G}_\nu^l \text{ and } \zeta_\nu^l \subseteq \left(X_\nu^l \times Y_{\nu\rceil}^l\right)^\infty \cap \mathcal{G}_\nu^l \tag{18}$$

are the *local system specification* and the *local environment assumption* for G_ν^l, respectively. The sets $\varphi^L \subseteq \mathcal{G}^L$ and $\zeta^L \subseteq \mathcal{G}^L$ are a system specification and an environment assumption for G^L, respectively. We define sets of local system specifications and local environment assumptions over $[\mathbb{G}]$ as

$$[\varphi] := \left\{\left\{\varphi_\nu^l\right\}_{\nu \in Y^{l+1}}\right\}_{l=0}^{L-1} \cup \{\varphi^L\} \quad \text{and} \quad [\zeta] := \left\{\left\{\zeta_\nu^l\right\}_{\nu \in Y^{l+1}}\right\}_{l=0}^{L-1} \cup \{\zeta^L\}. \tag{19}$$

A winning strategy for a local specification in layer $l+1$ induces transitions from a state (x, y) to a (possibly different) state (x, y'). As $y, y' \in Y^{l+1}$ are different contexts for layer l, this order of contexts must be obeyed by the strategy in layer l. Therefore, we need a proper translation of transitions in level $l+1$ into reachability specifications for local games in layer l and combine these specifications with the given low level tasks. Formally, the reachability specification for a layer $l \in [0, L-1]$ in context $\nu \in Y^{l+1}$ w.r.t. the next context $\nu' \in \text{Post}^{l+1}(\nu)$ is defined by

$$\psi_\nu^l(\nu') := \begin{cases} \left\{w \in \left(X_\nu^l \times Y_\nu^l\right)^* \cap \mathcal{G}_\nu^l \mid \lceil w \rceil \in Y_{\nu\lfloor}^l \cap Y_{\nu'\rceil}^l\right\}, & \nu \neq \nu' \\ \left\{\left(X_\nu^l \times Y_{\nu\rceil}^l\right)^\omega \cap \mathcal{G}_\nu^l\right\}, & \nu = \nu' \end{cases} \tag{20}$$

and the combination of $\psi_\nu^l(\nu')$ with a local task $\varphi_\nu^l \in [\varphi]$ is defined by

$$\phi_\nu^l(\nu') := \left\{\xi \cdot \xi' \,\middle|\, \xi \in \varphi_\nu^l \wedge \xi \cdot \xi' \in \psi_\nu^l(\nu')\right\}. \tag{21}$$

Example 8 Consider the floor plan in Fig. 3 and assume that the robot is in state q_{22}^5 corresponding to the states r_{11}^5 and f^5 in layers $l = 1$ and $l = 2$, respectively, as indicated by the light gray coloring. Now assume that the controller in layer $l = 2$ requests a context change from f^5 to f^6. This induces the reachability specification $\psi_{f^5}^1(f^6)$ containing all sequences of rooms in $\mathcal{G}^1{}_{f^5}$ with final room s_{56}. Now a memoryless strategy for this specification first needs to request a context change from r_{11}^5 to r_{21}^5. This request, in turn, induces the reachability specification $\psi_{r_{11}^5}^1(r_{21}^5)$ containing all sequences of cells in $\mathcal{G}_{r_{11}^5}$ with final cell q_{43}^5. A possible first move of the robot to fulfill this specification is from q_{22}^5 to q_{32}^5. The respective goal states of the two specifications are indicated in dark gray in Fig. 3.

The construction in (21) implies that only a (possibly strict) prefix ξ of a play $\pi \in \phi_\nu^l(\nu')$ needs to be contained in φ_ν^l. While this might seem restrictive for non-suffix closed specifications such as safety, one can circumvent this problem by using the idea of "weak until". Intuitively, one would specify to stay safe, i.e., only visit states from a set Q_{safe}, "until" the context is left. Then (21) checks if the current requested context change can be enforced by staying in safe states. For reachability type specifications, such as the request of the completion of a certain task, this issue does not arise.

Given the above definitions of local specifications, hierarchical reactive games can be constructed from a set of LGGs as follows.

Definition 3 Given a set of local specifications $[\varphi]$ over a set of LGGs $[\mathbb{G}]$ and a set of level 0 initial states $\mathcal{I} \subseteq (X \times Y)$, the tuple $([\mathbb{G}], \mathcal{I}, [\varphi])$ is called a *hierarchical reactive game* (HRG) over $[\mathbb{G}]$. Furthermore, given the set of local initial conditions

$$\mathcal{I}^l(m) := \begin{cases} \left\{ \left(\alpha_e^{l\uparrow}(x,y), \alpha_s^{l\uparrow}(y) \right) \mid (x,y) = \mathcal{I} \right\}, & m = 0 \\ \left\{ \left\lceil \breve{p}_{\downarrow}^{l}(m-1) \right\rceil \right\} & m > 0, l < L \\ \text{undefined} & else, \end{cases} \tag{22}$$

a set $[\breve{p}]_\pi$ is defined to be *winning* (resp. *possibly winning*) for $([\mathbb{G}], \mathcal{I}, [\varphi])$ if for all $l \in [0, L-1]$ it holds that

(i) for all $m \in \mathrm{dom}(\breve{\pi}^{l+1})$ (with $m < \mathsf{end}(\breve{\pi}^{l+1})$ if $\mathsf{end}(\breve{\pi}^{l+1}) < \infty$) there exists a prefix $\xi \sqsubseteq \breve{\pi}_{\downarrow}^{l}(m)$ s.t. ξ is winning for $\left(\mathcal{G}^l_{\breve{y}^{l+1}(m)}, \mathcal{I}^l(m), \varphi^l_{\breve{y}^{l+1}(m)} \right)$, and

(ii) for $m = \mathsf{end}(\breve{\pi}^{l+1}) < \infty$ there exists a string $\xi = \breve{p}_{\downarrow}^{l}(m)$ (resp. $\xi \sqsubseteq \breve{p}_{\downarrow}^{l}(m)$ or $\breve{p}_{\downarrow}^{l}(m) \sqsubseteq \xi$) s.t. ξ is winning for $\left(\mathcal{G}^l_{\breve{y}^{l+1}(m)}, \mathcal{I}^l(m), \varphi^l_{\breve{y}^{l+1}(m)} \right)$, and

(iii) $\breve{\pi}^L$ is winning (resp. possibly winning) for $(\mathcal{G}^L, \mathcal{I}^L(0), \varphi^L)$.

5 Assume-admissible hierarchical strategy construction

Let $([\mathbb{G}], \mathcal{I}, [\varphi])$ be a HRG with initial condition $\mathcal{I} \in (X \times Y)$ and let $[\zeta]$ be a set of local environment assumptions over $[\mathbb{G}]$. Then we want to synthesize a strategy (i.e., a controller) for layer 0 that generates a play whose projection is winning for the set of local system specifications $[\varphi]$ if $[\zeta]$ holds. We assume that $[\varphi]$ and $[\zeta]$ are both ω-regular languages. While in principle one can flatten the game and solve one global game to obtain a solution to this problem, this will be prohibitively expensive. We therefore propose an algorithm that constructs a winning strategy in each local game that is encountered and "stitches together" these winning strategies dynamically. Additionally, one could statically solve and memorize all possibly constructed local games. Our algorithm avoids this expensive construction by only solving games that actually arise online. Hence, our procedure is *dynamic* in that it solves a series of local games in each step starting from the current state — this is conceptually similar to receding horizon control approaches. To incorporate environment assumptions, we use a slightly modified version of the algorithm from Brenguier et al. (2015) to compute an assume-admissible winning strategy for a local game and a local environment assumption. Our procedure treats this algorithm as a black box; in principle, a different strategy synthesis algorithm can be used.

5.1 Synthesis of assume-admissibly winning strategies

Assume-admissibly winning strategies for the play $(G, \mathcal{I}, \varphi)$ w.r.t. the assumption ζ can be computed by the algorithm given by Brenguier (2015, Thm. 4) in case φ and ζ are ω-regular objectives. We denote the outcome of this strategy synthesis by $\mathrm{Sol}^{\mathsf{AA}}(G, \mathcal{I}, \varphi, \zeta)$. Whenever the environment does not play admissible, the definition of assume-admissibly winning strategies does only restrict the behavior of the system to an admissible one. This does not give any guarantees w.r.t. φ in case the environment does not play admissible. To circumvent this issue we slightly modify the outcome of the available strategy synthesis.

Definition 4 Let $f^{\mathsf{AA}} = \mathrm{Sol}^{\mathsf{AA}}(G, \mathcal{I}, \varphi, \zeta)$ be an assume-admissibly winning strategy, then its associated *possibly winning strategy* f, is defined for all $\pi \in \mathcal{G}$ s.t.

$$f(\pi|_{[0,k]}, x(k+1)) = \begin{cases} f^{\mathsf{AA}}(\pi|_{[0,k]}, x(k+1)), & \pi|_{[0,k]} \cdot (x(k+1), f^{\mathsf{AA}}(\pi|_{[0,k]}, x(k+1))) \in \overline{\varphi} \\ \emptyset, & \text{else.} \end{cases} \tag{23}$$

We define the set of all possibly winning strategies for the game $(G, \mathcal{I}, \varphi)$ w.r.t. ζ by $\mathrm{Sol}\,(G, \mathcal{I}, \varphi, \zeta)$.

A strategy $f = \mathrm{Sol}\,(G, \mathcal{I}, \varphi, \zeta)$ blocks whenever the environment forces the play into a state from which the play cannot be won anymore. This implies that all finite plays π compliant with f are possibly winning, i.e. $\pi \in \overline{\varphi}$, even if the environment does not play admissible. However, if it does, the compliant play is winning. This is formalized by the following proposition.

Proposition 3 *Given* $f = \mathrm{Sol}\,(G, \mathcal{I}, \varphi, \zeta)$, $g \in \mathcal{S}^e(G)$, *it holds for all* $\pi \in \mathcal{G}$ *that*

$$g \in \mathsf{AdmissibleStrategies}(G, \mathcal{I}, \zeta) \Rightarrow f \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g), \tag{24a}$$

$$\text{and } \begin{pmatrix} \pi \in \mathsf{CompliantPlays}(f, \mathcal{I}) \\ \wedge |\pi| < \infty \end{pmatrix} \Rightarrow \pi \in \mathsf{WinningPlays}(G, \mathcal{I}, \overline{\varphi}). \tag{24b}$$

Proof Let $f^{\mathsf{AA}} = \mathrm{Sol}^{\mathsf{AA}}(G, \mathcal{I}, \varphi, \zeta)$ and f its associated possibly winning strategy. Using (4), $g \in \mathsf{AdmissibleStrategies}(G, \mathcal{I}, \zeta)$ implies $f^{\mathsf{AA}} \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g)$. Using (3a), this implies $\pi \in \mathsf{WinningPlays}(G, \mathcal{I}, \overline{\varphi})$. Therefore, the second case in (4) cannot occur and we obtain $f = f^{\mathsf{AA}}$, i.e., $f \in \mathsf{WinningStrategies}(G, \mathcal{I}, \varphi, g)$. Observe that the left side of (24b) implies that the right side of (2) holds for π and f, hence $f(\pi|_{[0,k-1]}, x(k)) \neq \emptyset$ for all $k \in \mathrm{dom}(\pi)$. Using (4), this implies $\pi|_{[0,k]} \in \mathsf{CompliantPlays}(f^{\mathsf{AA}}, \mathcal{I})$ and $\pi|_{[0,k]} \in \overline{\varphi}$, hence, $\pi \in \mathsf{WinningPlays}(G, \mathcal{I}, \overline{\varphi})$. □

We remark that the algorithm to compute assume-admissible strategies in Brenguier (2015, Thm. 4) can be trivially adapted to ensure Prop. 3, by blocking the game whenever a losing state (one in which there is no winning strategy for the system) is entered.

5.2 The strategy synthesis algorithm

Recall that we aim to synthesize a strategy (i.e., a controller) for layer 0 that generates a play whose projection is assume-admissible winning for the HRG $([\mathbb{G}], \mathcal{I}, [\varphi])$ w.r.t. $[\zeta]$. Hence, the goal of each computation round of our algorithm is to determine the next system state $y(k+1)$ in layer 0, i.e., to calculate the current control action that needs to be applied to the system. This depends on the environment state $x(k+1)$ in layer 0 which is sensed in the beginning of each such computation round and projected to all layers $l \in [1, L]$ in an "bottom up" fashion. The current state in every layer local game is given by the restriction of $x^l(k+1)$ to the current context and the projection $y^l(k)$ of the last system state. Based on this information, the next step in every layer local game needs to be calculated.

This calculation is challenging due to the interaction between plays in different layers. In particular, a move from system state v to v' requested by a strategy in layer $l \in [1, L]$ results in an additional reachability specification for the current local game in layer $l-1$.

Furthermore, such an "induced" reachability specification for the local game in layer $l-1$ and context ν might change multiple times, before this context is left. This is due to the fact that an environment state in layer $l > 0$ possibly changes multiple times before a system state change follows, as discussed in the construction of abstract game graphs (see Section 3.2). Hence, whenever such a specification change occurs, the strategy in layer $l-1$ needs to be re-calculated. The only strategy that is not influenced by this interplay is the highest level strategy, which is computed only once when initializing the algorithm. Once the strategies are updated in a "top down" manner, the controller picks the next move at layer 0 based on the updated strategy for layer 0 and plays it. This changes the states for all higher layers and the algorithm continues with the next computation cycle.

We now describe the algorithm formally.

Algorithm 1 (strategy synthesis procedure) Let $([\mathbb{G}], \mathcal{I}, [\varphi])$ be a HRG with $\mathcal{I} \in (X \times Y)$ and $[\zeta]$ a set of local environment assumptions over $[\mathbb{G}]$. Then the dynamic hierarchical strategy $F = \{f^l\}_{l=0}^{L}$ for the game $([\mathbb{G}], \mathcal{I}, [\varphi])$ w.r.t. $[\zeta]$ and its compliant play π are iteratively defined as follows:

- Initialization:
  - Using $\mathcal{I}^L$ as in (22), calculate the assume admissible winning strategy for the highest layer L using
$$h^L = \mathrm{Sol}\,(G^L, \mathcal{I}^L(0), \varphi^L, \zeta^L). \tag{25a}$$
  - Initialize the play and the local history for $l \in [0, L]$, respectively, with
$$\pi = (x(0), y(0)) = \mathcal{I} \quad \text{and} \quad \breve{\gamma}^l(0) = \mathcal{I}^l(0). \tag{25b}$$
- Iteration for all $k \in \mathbb{N}$:
  - Sense the environment move
$$x(k+1) \in \delta^0(\pi). \tag{25c}$$
  - Compute the local environment state $x_{\downarrow}^l(k+1)$ using (6) and (15a), i.e.,
$$x_{\downarrow}^l(k+1) = \mathfrak{r}_{y^{l+1}(k)}^l \left(\alpha_e^{l\uparrow}(x(k+1), y(k))\right) \tag{25d}$$
for each layer l.
  - Iteratively calculate the current strategy by
$$f^L(k) = h^L \tag{25e}$$
and
$$\forall l \in [0, L-1]\,.\, f^l(k) = \begin{cases} \emptyset & \mathrm{GotStuck}^{l+1}(k) \\ h^l(k) & \mathrm{Done}^{l+1}(k) \\ f_{\nu\nu'^{l+1}}^l(k) & else \end{cases} \tag{25f}$$

with

$$\begin{aligned} \nu &:= y^{l+1}(k), \\ \nu'^{l+1}(k) &:= f^{l+1}(k)\left(\breve{\gamma}^{l+1}(k), x_{\downarrow}^{l+1}(k+1)\right), \end{aligned} \tag{25g}$$

$$h^l(k) := \begin{cases} \mathrm{Sol}\left(G_\nu^l, \{\breve{\gamma}^l(k)\}, \varphi_\nu^l, \zeta_\nu^l\right), & \nu \neq y^{l+1}(k-1) \\ h^l(k-1), & else \end{cases} \tag{25h}$$

$$f_{\nu\nu'^{l+1}}^l(k) = \begin{cases} \mathrm{Sol}\left(G_\nu^l, \{\breve{\gamma}^l(k)\}, \phi_\nu^l(\nu'^{l+1}(k)), \zeta_\nu^l\right), & \begin{pmatrix} \nu \neq y^{l+1}(k-1) \\ \vee \nu'^{l+1}(k) \neq \nu'^{l+1}(k-1) \end{pmatrix} \\ f_{\nu\nu'^{l+1}}^l(k-1) & else \end{cases} \tag{25i}$$

and the predicates are defined by

$$\mathrm{Win}^l(k) \Leftrightarrow \breve{\gamma}^l(k) \in \begin{cases} \varphi_\nu^l, & l \in [0, L-1] \\ \varphi^L, & l = L \end{cases}, \tag{25j}$$

$$\mathrm{Done}^l(k) \Leftrightarrow \begin{pmatrix} (l = L \vee \mathrm{Done}^{l+1}(k)) \\ \wedge \mathrm{Win}^l(k) \\ \wedge (\breve{\gamma}^l(k), x_{\downarrow}^l(k+1)) \notin \mathrm{dom}(h^l(k)) \end{pmatrix}, \tag{25k}$$

and

$$\mathrm{GotStuck}^l(k) \Leftrightarrow \begin{pmatrix} \neg \mathrm{Done}^l(k) \\ \wedge (\breve{\gamma}^l(k), x_{\downarrow}^l(k+1)) \notin \mathrm{dom}(f^l(k)) \end{pmatrix}. \tag{25l}$$

▷ Play the next move following the current system strategy for layer $l = 0$

$$y(k+1) = f^0(k)\left(\breve{\gamma}^0(k), x_{\downarrow}^0(k+1)\right). \tag{25m}$$

▷ Append $(x(k+1), y(k+1))$ to the play giving

$$\pi = (x|_{[0,k+1]}, y|_{[0,k+1]}). \tag{25n}$$

▷ Using (16b), compute the new context restricted history

$$\breve{\gamma}^l(k+1) = \left\lceil \breve{p}_{\downarrow}^l \right\rceil \quad \text{with} \quad \breve{p}_{\downarrow}^l \in [\breve{p}]_\pi. \tag{25o}$$

As discussed before, every computation round k of the construction in (25a) starts with the sensing of the next environment move in (25c), giving the full 0-level environment state $x(k+1) = x^0(k+1)$. This state is used to compute the local restricted environment states $x_{\downarrow}^l(k+1)$ for every layer and current context $y^{l+1}(k)$ in (25d). Note that this construction is done "bottom up".

Thereafter, the selection of the current strategy f^l for every layer and its respective current goal state ν'^l are calculated. Observe that this is done "top down", as ν'^l is used to calculated the current reachability specification for the reachability game in layer $l-1$. The construction of f^l in (25f) distinguishes three cases: the play at the highest layer has been won, or the play at the higher layer got stuck, or none of these conditions occurred. We consider these cases separately.

For the first case observe that the specification of level L might be a set of finite strings and local specifications are sets of finite strings by definition (see Section 4.2). Therefore, the play constructed in (25) does not need to be infinite to be winning for $[\varphi]$. If the play in layer L is winning for φ^L and the strategy does not request any other move (denoted by the predicate Done^L in (25k)), then this is communicated downwards using the second line

of (25f). In this case all lower level strategies must be winning for local specifications only, using the assume-admissible strategy calculated in (25h).

For the second case, observe that the strategy calculation in (25h) and (25i) does not need to have a solution. Further, even if it has a solution, system strategies are not assumed to be left-total. Hence, there might exist (non-admissible) environment moves that cause a blocking of f without the game being winning. These two situations are modeled by the predicate GotStuck^l in (25k). If such a situation occurs, it is communicated downwards by the first line of (25f) resulting in $\text{GotStuck}^{l'}$ for all $l' < l$ and therefore an abortion of the game. Intuitively, the first time GotStuck^l occurs, it is because of an "unrealizeable" local specification. We introduce a fourth predicate

$$\text{UnRealizable}^l(k) \Leftrightarrow \begin{cases} \text{GotStuck}^l(k), & l = L \\ \neg\,\text{GotStuck}^{l+1}(k) \wedge \text{GotStuck}^l(k), & l < L \end{cases} \tag{26}$$

to remember the first layer where the controller got stuck. We will show in Section 5.3 that an unrealizable specification is the only reason for a non-winning play constructed in (25) to be aborted.

In the third case, i.e., if neither GotStuck^l nor Done^{l+1} are true, the strategy for level l is calculated by (25i) using again two subcases. In the first subcase, either a new context was entered (resulting in a new local game) or the "top down induced" reachability specification has changed (due to a change of ν'^l caused by a new environment state in layer $l+1$). In this case the strategy for level l needs to be re-calculated. However, if neither of these two situations occurs, the strategy from the previous time step can be used, avoiding unnecessary re-computations.

After the strategy construction in (25f)–(25l), the system state is updated to $y(k+1)$, using the currently selected lowest level strategy $f^0(k)$ in (25m). Hence, (25f)–(25l) only utilize the hierarchical structure of the game graph to compute $f^0(k)$, which is the only control action that is actually applied to the system, e.g., the robot in our example. Then $(x(k+1), y(k+1))$ is appended to the constructed play π. As intuitively assumed, such plays π generated by Alg. 1 up to length k are plays in G, i.e., $\pi \in \mathcal{G}$, as shown in the following proposition. Observe that this implies that also $\check{\pi}^l \in \mathcal{G}^l$ for all $l \in [0, L]$ (from Prop. 1) and $\check{p}^l_{\downarrow}(m) \in \mathcal{G}^l_{\check{y}^{l+1}(m)}$ for all $l \in [0, L-1]$ and $m \in \text{dom}(\check{\pi}^{l+1})$ (from Prop. 2).

Proposition 4 *Let π be a play computed in Alg. 1. Then $\pi \in \mathcal{G}$.*

Proof It follows from (25c) and (25m) that

$$\forall k \in \text{dom}^+(\pi). \left(\begin{array}{l} x(k) \in \delta(x(k-1), y(k-1)) \\ \wedge y(k) = f^0(k-1)(\check{\gamma}^0(k-1), x^0_{\downarrow}(k)) \end{array} \right), \tag{27}$$

implying $f^0(k-1) \neq \emptyset$ for all $k \in \text{dom}^+(\pi)$. Therefore, (25f)–(25l) imply that $f^0(k-1)$ is a system strategy over $\mathcal{G}^0_{y^1(k-1)}$ and the definition of the latter in Section 2 gives $f^0(k-1)(\check{\gamma}^0(k-1), x^0_{\downarrow}(k)) \in \rho^0_{y^1(k-1)}(x^0_{\downarrow}(k-1), \lceil \check{\gamma}^0(k-1) \rceil_2)$. Now observe from (25o), (16b) and (8) that $\lceil \check{\gamma}^0(k-1) \rceil_2 = y^0(k-1)$. Now using $\rho^0_{y^1(k-1)} \subseteq \rho^0$ from Ass. 1 along with this observation, we see that (27) actually implies (1), hence $\pi \in \mathcal{G}$. □

We call a play π calculated in (25) up to length $k = |\pi|$ *maximal* if

$$k < \infty \Rightarrow \left(\check{\gamma}^0(k), x^0_{\downarrow}(k+1) \right) \notin \text{dom}(f^0(k)). \tag{28}$$

One round of the construction in (25) is ended by calculating the current local histories $\breve{\gamma}^l(k+1)$ for every layer. Intuitively, $\breve{\gamma}^l(k+1)$ models the part of $\breve{\pi}^l$ generated after the last context change in layer l and is therefore equivalent to $\lceil \breve{p}^l_{\downarrow} \rceil$. These histories are used in the calculation of assume-admissible strategies to ensure that a re-computation of a strategy within one context does result in a continuation of the already generated string w.r.t. the given specification.

While the local system strategies $f^l(k)$ are explicitly calculated for every time step k in (25f)–(25l), the local environment strategies $g^l(k)$ are only given implicitly by the observed environment move (25c) and its abstraction to every layer l. Formally, a play π calculated in (25) was played against an admissible environment strategy if for all $l \in [0, L-1]$, $m \in \mathrm{dom}(\breve{\pi}^l)$ there exists an environment strategy $g^l_{\breve{y}^{l+1}(m)} \in \mathsf{AdmissibleStrategies}(G^l_{\breve{y}^{l+1}(m)}, \mathcal{I}^l(m), \zeta^l_{\breve{y}^{l+1}(m)})$ s.t. $\breve{p}^l_{\downarrow}(m) \in \mathsf{CompliantPlays}(G^l_{\breve{y}^{l+1}(m)}, g^l_{\breve{y}^{l+1}(m)})$ and for layer L exist $g^L \in \mathsf{AdmissibleStrategies}(G^L, \mathcal{I}^L(0), \zeta^L)$ s.t. $\breve{\pi}^L \in \mathsf{CompliantPlays}(G^L, g^L)$. If this holds, we call π an *environment admissible play*.

Example 9 Consider the play π whose y-component is depicted by filled cycles in Fig. 1 (bottom) and (for simplicity) the static environment used in Expl. 7, where we use $o = \{q^5_{24}, q^5_{25}, q^5_{63}\}$ and $o_\downarrow = \{q^5_{24}, q^5_{25}\}$ for notational convenience. In this game the only objective is to reach q^6_{63} in r^6_{21} and f^6. This implies that $[\varphi]$ contains only empty sets except for

$$\varphi^2 = \{\bot\} \times \{f^5 f^6\},\ \varphi^1_{f^6} = \{\bot\} \times R^* \cdot \left\{r^6_{21}\right\},\ \text{and}\ \varphi^0_{r^6_{21}} = \{\bot\} \times Q^* \cdot \left\{q^6_{21}\right\}.$$

To illustrate Alg. 1 we pick $k = 2$, i.e., π was generated for 3 time steps and we are now calculating $\pi(3) = (x(3), y(3))$ using (25).
First recall from Expl. 7 that

$$\pi(2) = \left(o, q^5_{33}\right),\ \pi^1(2) = \pi^1(0) = (\{d\}, r^5_{11},\ \pi^2(2) = \pi^2(0) = (\{\bot\}, f^5),$$

and

$$\breve{\gamma}^0(2) = \left(o_\downarrow, q^5_{22}\right)\left(o_\downarrow, q^5_{23}\right)\left(o_\downarrow, q^5_{33}\right),\ \breve{\gamma}^1(2) = (\{d\}, r^5_{11}),\ \breve{\gamma}^2(2) = (\{\bot\}, f^5).$$

We furthermore assume that the strategy calculation for $k = 0$ resulted in the requested moves depicted by the arrows in Fig. 3 (middle and top). Whith this initialization we obtain the following steps of the algorithm.

- ▹ Due to the static environment assumption, (25c) gives $x(k+1) = x(3) = o$.
- ▹ Applying (25d) yields $x^0_\downarrow(3) = o_\downarrow$, $x^1_\downarrow(3) = \{d\}$ and $x^2_\downarrow(3) = \{\bot\}$.
- ▹ First, (25e) and (25f) imply $f^2(2) \neq \emptyset$, $\nu'^2(2) = \nu'^2(1) = f^6$ and $\neg\mathrm{Done}^2(2)$. Therefore, (25i) and (25f) imply $f^1(2) = f^1_{f^5 f^6}(0) \neq \emptyset$, $\nu'^1(2) = \nu'^1(1) = r^5_{11}$ and $\neg\mathrm{Done}^1(2)$. With this, the lowest level strategy is given by $f^0(2) = f^0_{r^5_{11}, r^5_{21}}(0)$.
- ▹ As we assume a static environment and no obstacles block the way between the robot and the exit to room r^5_{21}, we assume that $f^0_{r^5_{11}, r^5_{21}}$ is a shortest path strategy and (25m) gives $y(k+1) = y(3) = q^5_{43}$.
- ▹ Observe that a context change has occurred during this step, i.e., (25o) gives

$$\breve{\gamma}^0(3) = \left(x^0_\downarrow(3), y(3)\right) = \left(o_\downarrow, q^5_{43}\right),\ \breve{\gamma}^1(3) = \left(\{d\}, r^5_{11}\{d\}, r^5_{21}\right),\ \breve{\gamma}^2(3) = (\{\bot\}, f^5).$$

With this local history the next iteration of the algorithm is started. For the assumed very simple static environment, Alg. 1 will never get stuck. Observe that once we reach floor f^6, the level 2 game is won and Done^2 is true. In this case h^1 will be calculated w.r.t. the specification $\varphi^1_{f^6}$. If in addition r^6_{21} is reached, Done^1 is also set to true and h^0 is calculated. After one more time step also Done^0 is true and the algorithm terminates. The generated play is obviously winning for $[\varphi]$.

5.3 Soundness

In this section we prove three different soundness results for the play constructed in Alg. 1. Intuitively, Alg. 1 is sound if a play π calculated in (25) is winning for the HGG $([\mathbb{G}], \mathcal{I}, [\varphi])$ if all generated local specifications are realizable and the environment plays admissible w.r.t. $[\zeta]$, which will be proven last in Thm. 3. As a first intermediate result we show that the only two reasons for a maximal play to terminate are actually that (i) a current local specification is not realizable or (ii) the play is already winning given a finite winning condition in layer L.

Theorem 1 *Let π be a maximal play computed by (25). Then it holds that*

$$|\pi| < \infty \Leftrightarrow \begin{pmatrix} \forall l \in [0, L] \,.\, \text{Done}^l(\mathsf{end}(\pi)) \\ \vee \exists l \in [0, L] \,.\, \text{UnRealizable}^l(\mathsf{end}(\pi)) \end{pmatrix}. \tag{29}$$

Proof To prove this theorem we need that

$$\Big(\exists l \in [0, L]. \text{UnRealizable}^l(\mathsf{end}(\pi))\Big) \Leftrightarrow \text{GotStuck}^0(\mathsf{end}(\pi)) \tag{30a}$$

which is proven for all $k \in \text{dom}(\pi)$ in Lem. 5 (see Appendix A). Furthermore, as we assume environment strategies to be left-total, (25c) can always be computed. Hence, π becomes finite while being maximal iff (25m) cannot be evaluated, i.e.,

$$\mathsf{end}(\pi) < \infty \Leftrightarrow (\check{\gamma}^0(\mathsf{end}(\pi)), x^0_{\downarrow}(\mathsf{end}(\pi) + 1)) \notin \text{dom}(f^0(\mathsf{end}(\pi))). \tag{30b}$$

Now we pick $k = \mathsf{end}(\pi)$ and prove both directions separately.
"$\Rightarrow$" Using (30b) and (25l) implies that either (i) $\neg\text{Done}^0(k)$ and $\text{GotStuck}^0(k)$, or (ii) $\text{Done}^0(k)$. Using (30a), (i) implies[3] $\langle$(29).right.2$\rangle$. As $\text{Done}^0(k)$ implies $\forall l \in [0, L].\text{Done}^l(k)$ (from (25k)), (ii) implies $\langle$(29).right.1$\rangle$.
"$\Leftarrow$" If $\langle$(29).right.2$\rangle$ is true, it follows from (30a) that $\text{GotStuck}^0(k)$ and $\neg\text{Done}^0(k)$ (see the proof of Lem. 5). Hence, (29) and (30b) implies $\langle$(29).left$\rangle$. If $\langle$(29).right.1$\rangle$ is true, we know from (25f) that $f^0(k) = h^0(k)$. Therefore, $\langle$(25k).right.3$\rangle$ and (30b) implies $\langle$(29).left$\rangle$. □

While the second case in Thm. 1 is not desired w.r.t. the goal of constructing a winning play, it can usually not be avoided in a realistic scenario as we can (i) not enforce the environment to play admissible and (Ii) checking feasibility of all possibly occurring local games before startup might not be appropriate, as this set might be very large. However, Alg. 1 ensures that if this situation occurs, the local specifications are not falsified up to this

[3]To simplify notation we denote by $\langle$(#).right.n$\rangle$ (resp. $\langle$(#).left.n$\rangle$) the nth statement on the right (resp. left) side of the implication/equivalence relation in equation (#).

point. This is formalized by the notion of possibly winning, which ensures that generated finite plays always stay in the prefix closure of the considered local specifications.

Theorem 2 *Given the preliminaries of Alg. 1, let π be the play computed by (25) up to length k, and $[\check{p}]_\pi$ its set of local projected play sequences. Then $[\check{p}]_\pi$ is possibly winning for $([\mathbb{G}], \mathcal{I}, [\varphi])$.*

Proof We have two important observations that we use in this proof. First, it holds for all $l \in [0, L]$ and $m \in \mathrm{dom}^+(\check{\pi}^l)$ that

$$\begin{pmatrix} \check{x}^l_\downarrow(m) \in \delta^l_{\check{y}^l(m-1)}(\check{x}^l_\downarrow(m-1), \check{y}^l(m-1)) \\ \wedge \check{y}^l(m) = f^l(\kappa^l(m)-1)(\check{\gamma}^l(\kappa^l(m)-1), \check{x}^l_\downarrow(m)) \end{pmatrix} \tag{31a}$$

as proven in Lem. 8 (see Appendix A). Second, it holds for all $l \in [0, L-1]$ and $m \in \mathrm{dom}^+(\check{\pi}^{l+1})$ that

$$\check{p}^l_\downarrow(m-1) \in \phi^l_{\check{y}^{l+1}(m-1)}(\check{y}^{l+1}(m)) \tag{31b}$$

and for $m = \mathsf{end}(\check{\pi}^{l+1})$ there exists $\nu' \in \mathrm{Post}^{l+1}(\check{y}^{l+1}(m))$ s.t.

$$\check{p}^l_\downarrow(m) \in \overline{\phi^l_{\check{y}^{l+1}(m)}(\nu')}, \tag{31c}$$

as proven in Lem. 9 (see Appendix A).

Recall from Prop. 4 that $\pi \in \mathcal{G}$, hence Prop. 2 implies $\check{p}^l_\downarrow(m) \in \mathcal{G}^l{}_{\check{y}^{l+1}(m)}$ and (16a) obviously gives $\check{p}^l_\downarrow(m)|_{0,0} = \lceil \check{p}^l_\downarrow(m-1) \rceil = \mathcal{I}^l(m)$ for all $m \in \mathrm{dom}^+(\check{\pi}^{l+1})$. As (31b) holds, (21) implies

$$\exists \xi \in \overline{\{\check{p}^l_\downarrow(m-1)\}} \,.\, \xi \in \varphi^l_{\check{y}^{l+1}(m-1)}. \tag{32a}$$

Now consider $m = \mathsf{end}(\check{\pi}^{l+1})$. As (31c) holds, (21) implies that either

$$\check{p}^l_\downarrow(m) \in \overline{\varphi^l_{\check{y}^{l+1}(m)}} \quad \text{or} \quad \exists \xi \in \overline{\left\{\check{p}^l_\downarrow(m)\right\}} \,.\, \xi \in \varphi^l_{\check{y}^{l+1}(m)}. \tag{32b}$$

Using the definitions of winning from Section 2, (32a)–(32b) imply that conditions (i)-(ii) for possibly winning HRGs from Section 4.2 hold. To prove condition (iii), observe from (25e) that $\forall k \in \mathbb{N} \,.\, f^L(k) = h^L$. Furthermore, recall from the definition of $[\check{p}]_\pi$ that $\check{p}^L_\downarrow(0) = \check{\pi}^L$ and $\mathsf{end}(\check{p}^L_\downarrow) = 0$ and therefore $\check{\gamma}^L(\kappa^l(m)-1) = \check{\pi}^L\big|_{[0,\kappa^l(m)-1]}$. Using these observations in (31a), it follows that (2) holds for $\check{\pi}^L$ w.r.t. h^L and $\mathcal{I}^L(0)$, implying $\pi^L \in \overline{\mathsf{CompliantPlays}(h^L, \mathcal{I}^L(0))}$. As $h^L = \mathrm{Sol}\left(G^L, \mathcal{I}^L(0), \varphi^L, \zeta^L\right)$ and $\check{\pi}^L \in \mathcal{G}^L$ (from Prop. 4 and Prop. 1), it follows from (24b) in Prop. 3 that $\check{\pi}^L$ is possibly winning for $(\mathcal{G}^L, \mathcal{I}^L(0), \varphi^L)$. □

We now prove the main result of this paper, namely that maximal plays π calculated by Alg. 1 (finite and infinite) are actually winning for $([\mathbb{G}], \mathcal{I}, [\varphi])$ if the environment plays admissible and all constructed local plays have a solution, i.e.,

$$\forall k \in \mathrm{dom}(\pi), l \in [0, L] \,.\, \neg \mathrm{UnRealizable}^l(k). \tag{33}$$

Theorem 3 *Let π be a maximal and environment admissible play computed by (25) s.t. (33) holds and let $[\check{p}]_\pi$ be its set of local play sequences. Then $[\check{p}]_\pi$ is winning for $([\mathbb{G}], \mathcal{I}, [\varphi])$.*

Proof In this proof we use the following two observations

$$\Big(\forall k \in \mathrm{dom}(\pi), l \in [0, L].\neg \mathrm{Done}^l(k)\Big) \Leftrightarrow (|\pi| = \infty) \Leftrightarrow \Big(\forall l \in [0, L].|\breve{\pi}^l| = \infty\Big). \tag{34a}$$

$$\forall l \in [0, L]\ .\ \mathrm{Done}^l(\mathsf{end}(\pi)) \Leftrightarrow (|\pi| < \infty) \Leftrightarrow \Big(\forall l \in [0, L]\ .\ |\breve{\pi}^l| < \infty\Big). \tag{34b}$$

where (34a) was proven in Lem. 11 (see Appendix A), the left side of (34b) follows from Thm. 1 and (33), and the right side of (34b) is a simple consequence from the definition of projections in (8). Hence, we generally have two cases to consider when proving the three conditions for winning HRGs from Section 4.2.

First observe that condition (i) is equivalent for winning and possibly winning, no matter whether π is finite or not. It therefore follows directly from Thm. 2. Furthermore, condition (ii) only needs to be proven if $|\breve{\pi}^{l+1}| < \infty$ and recall that for this case Thm. 2 shows that $\breve{p}^l_{\downarrow}(\mathsf{end}(\breve{\pi}^{l+1}))$ is *possibly* winning for $\Big(\mathcal{G}^l_{\breve{y}^{l+1}(m)}, \lceil \breve{P}^l_{\downarrow}(m-1)\rceil, \varphi^l_{\breve{y}^{l+1}(m)}\Big)$ for all $l \in [0, L]$. Now observe from (34b) that $\mathrm{Done}^l(\mathsf{end}(\pi))$ which implies from (25k) and (25j) that $\breve{p}^l_{\downarrow}(\mathsf{end}(\breve{\pi}^{l+1})) = \breve{\gamma}^l(\mathsf{end}(\pi)) \in \varphi^l_{\breve{y}^{l+1}(m)}$, where the first equality follows from (25o) and (16a). This obviously implies that $\breve{p}^l_{\downarrow}(\mathsf{end}(\breve{\pi}^{l+1}))$ is winning in the above game. For finite plays, this reasoning also proves condition (iii). We therefore assume $|\breve{\pi}^L| = \infty$ and recall from the proof of Thm. 2 that (2) holds for $\breve{\pi}^L$ w.r.t. h^L and $\mathcal{I}^L(0)$. As $|\breve{\pi}^L| = \infty$ we have $\breve{\pi}^L \in \mathsf{CompliantPlays}\left(h^L, \mathcal{I}^L(0)\right)$. As $h^L = \mathrm{Sol}\left(G^L, \mathcal{I}^L(0), \varphi^L, \zeta^L\right)$ and $\breve{\pi}^L \in \mathcal{G}^L$ (from Prop. 4 and Prop. 1) and $g^L \in \mathsf{AdmissibleStrategies}\left(G^L, \mathcal{I}^L(0), \varphi^L, \zeta^L\right)$, it follows from (24b) in Prop. 3 that $\breve{\pi}^L$ is winning for $(\mathcal{G}^L, \mathcal{I}^L(0), \varphi^L)$. □

The important difference between Thm. 2 and Thm. 3 is that environment admissible infinite plays can only be generated if layer L does not win in finite time, i.e., $\neg\,\mathrm{Done}^L(k)$ for all $k \in \mathrm{dom}(\breve{\pi}^L)$. If the environment does not play admissible, infinite plays can also be generated if $\mathrm{Done}^L(k)$ is true, as the environment might never "help" to reach the specification (i.e., does not play admissible) but also never moves to a losing state (i.e., causing the game to be aborted).

Remark 1 It should be noted that Alg. 1 works identically if we use a different synthesis technique to calculate local strategies in $\mathrm{Sol}(\cdot)$. For example, one could either calculate winning (instead of assume-admissibly winning) strategies in $\mathrm{Sol}(\cdot)$ (e.g., from the methods by Zielonka (1998) and Emerson and Jutla (1991) for general ω-regular conditions, or more specialized procedures, e.g., by Bloem et al. (2012)) or use a different synthesis method that incorporates environment assumptions, as, e.g., in Bloem et al. (2015).

The main purpose of Thm. 1-Thm. 3 is to show that local plays according to local strategies are "stitched together" correctly by Alg. 1, such that the resulting play fulfills the overall specification in the proposed hierarchical manner. The proofs of these theorems use only very general properties of the local strategy synthesis algorithms. In particular, the proof of Thm. 1 does not use any special information about local system or environment strategies. Furthermore, the proof of Thm. 2 only uses property (24b) in Prop. 3 from assume-admissible strategies, which also holds for "usual" winning strategies. Finally, the proof of Thm. 3 also uses (24a) but only to show that generated plays are winning if the environment plays admissible. By using other notions of winning, the proof of Thm. 3 should hold with only slight modifications.

5.4 Comments on completeness

Intuitively, the synthesis procedure given in Alg. 1 is complete if, whenever there exists a strategy $\hat{f}$ over the game graph G s.t. all plays $\hat{\pi} \in \mathcal{G}$ compliant with $\hat{f}$ induce a set of local play sequences that are winning for $([\mathbb{G}], \mathcal{I}, [\varphi])$ (if the environment plays an admissible strategy), then there exists a hierarchical strategy F s.t. its compliant play π generated by (25) induces projected plays that are also winning for $([\mathbb{G}], \mathcal{I}, [\varphi])$ (if the environment plays an admissible strategy).

Unfortunately, this statement is not true. The major problem arises from the fact that (assume-admissibly) winning strategies are usually not unique for a particular game. Therefore, using one particular strategy calculated by Sol $(\cdot)$ disregards other winning plays. This has two important consequences. First, a move of the current layer l strategy cannot be revised if the current layer $l-1$ game is not realizable for the corresponding reachability specification, even if there exists a different possibly winning extension in layer l. In our robot example, this corresponds to the case where the robot is in a particular room r with two adjacent rooms r' and r'', where visiting either of them is winning. Now the current strategy for the room layer deterministically picks room r'. If the way towards room r' is blocked by a static obstacle, the game in layer 0 and context r does not have a solution and the play gets stuck.

This problem also arises in reverse layer interaction, as assume-admissibly winning strategies are only ensured to be winning against a "local" admissible environment strategy. They do not consider admissible environment moves in higher layers that might cause specification changes in the current layer. Hence, the local strategy synthesis might pick a strategy that leads the play to a region of the state space that is losing for a different specification that might occur later in this game due to such an admissible environment move in a higher layer. In the above example this would correspond to the case that the door to room r' gets closed, which is visible to layer 1, and therefore causes the strategy to request the robot to move to room r'', instead. Now assume that the way towards both r' and r'' was unblocked initially. Given the specification to reach r' the robot might pick one of two passages that allow to reach r' but the selected one is to narrow for the robot to turn. When the specification changes, the robot cannot turn and approach r'', hence the game in layer 0 and context r does not have a solution and the play gets stuck. Taking these interactions into account when synthesizing local assume-admissible winning strategies is a promising idea for future work to obtain a complete algorithm. This would also reduce blocking situations that are caused by this interplay.

Completeness holds in the special case of a trivial environment (which has no choice of moves) and the strategy only picks one among the choice of system moves (as e.g. in Kloetzer and Belta 2008; Vasile and Belta 2014). However, in this case, one can compute a strategy statically using a dynamic programming procedure similar to context free reachability (see Reps et al. 1995; Alur et al. 2003).

6 Simulation examples

We have implemented our hierarchical strategy synthesis procedure (Algorithm 1) on top of LTLMoP (Finucane et al. 2010), an open source mission planning tool for robotic applications. In our implementation, we use the fact that Algorithm 1 uses the solution of local games, denoted by Sol $(\cdot)$, as a black-box building block. We can therefore treat every local game as a separate instance of LTLMoP with its own defined sub-regions, its own

specification, and configuration. The hierarchical algorithm is then implemented as an abstract handler in LTLMoP with customized executors for local games. A detailed description of the current implementation can be found in Leva (2016). As LTLMoP synthesizes winning strategies for two player games w.r.t. specifications in the GR1 fragment of LTL (see Bloem et al. (2012) and Finucane et al. (2010) for details), our implementation currently only supports procedures Sol $(\cdot)$ for this specification class.

6.1 Robot simulation

To illustrate how hierarchical games are simulated using our implementation, we consider a very simple type of GR1 specifications of the form:

$$\bigwedge_i \Box\Diamond A_i \rightarrow \bigwedge_i \Box\Diamond B_i,$$

where $\Box$ and $\Diamond$ encode the temporal properties "always" and "eventually", respectively, and A_i and B_i are Boolean formulas over environment predicates and robot goals, respectively. The formula $\Box\Diamond A$ states that the formula A is true infinitely often. We evaluate this specification on a simple hierarchical game with two layers. It consists of six connected rooms each composed of nine sub-regions together with an exit-region for every available exit. Figure 4 shows the regions for layer $L = 1$ (left) and details the setup for room r_1 (middle). Note that the exit labeled by e_{12} goes from room r_1 to room r_2. All other exits are labeled accordingly. In this setup, there are doors located between rooms r_1 and r_2 and between rooms r_5 and r_6 which are called d_{12} and d_{56}, respectively, and are treated as the only environment variables in both layer 0 and layer 1. We furthermore assume that these doors are infinitely often open and ask the robot to go to region 5 in room r_3. Including the assumptions on the environment, this task is given by the GR1 formula for the highest layer $L = 1$ by $\varphi^1 = (\Box\Diamond\neg d_{12} \wedge \Box\Diamond\neg d_{56}) \rightarrow \Box\Diamond r_3$. For layer 0 we have only assumptions on the door predicates in the local specifications, i.e., $\varphi^0_{r_1} = \varphi^0_{r_2} = \Box\Diamond\neg d_{12} \rightarrow \mathsf{true}$ and $\varphi^0_{r_5} = \varphi^0_{r_6} = \Box\Diamond\neg d_{56} \rightarrow \mathsf{true}$ and require region 5 to be reached inside r_3, i.e., $\varphi^0_{r_3} = \mathsf{true} \rightarrow \Box\Diamond c_5$. We furthermore have $\varphi^0_{r_4} = \mathsf{true}$.

Using these specifications, the algorithm first starts by solving the game in layer 1. As door d_{12} is open initially, the resulting play in layer 1 requests a move to room r_2 first. Therefore a game in room r_1 is instantiated by adding the guarantee to reach e_{12} to the local specification, giving $\varphi^0_{r_1} = \Box\Diamond\neg d_{12} \rightarrow \Box\Diamond e_{12}$. While the robot is approaching this exit, door d_{12} gets closed. While this usually causes the robot to wait in room 1 until the door gets reopened, in our algorithm this change of the status of the environment variable d_{12} is communicated upwards to layer 1 and causes the strategy in this layer to request a move

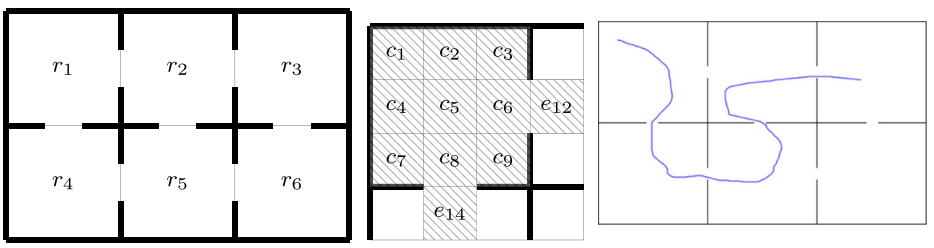

Fig. 4 Simulation setup for layer 1 (*left*) and room r_1 (*middle*), and the resulting trajectory of the robot (*right*) for the scenario described in Section 6.1

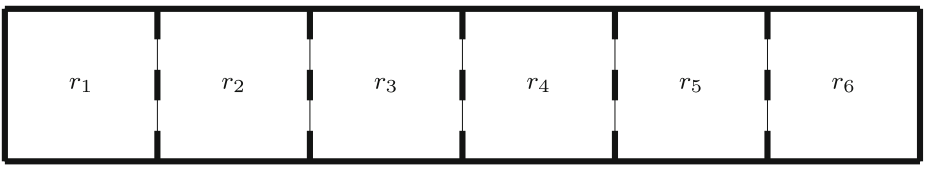

Fig. 5 Simulation setup for layer 1 for the scenario described in Section 6.2

from r_1 to r_4 instead. This implies that a new local game is started in r_1 with the changed specification $\varphi^0_{r_1} = \square\lozenge\neg d_{12} \rightarrow \square\lozenge e_{14}$. When e_{14} is reached the abstract handler realizes that the robot is now located in room r_4, hence the transition from r_1 to r_4 in layer 1 was completed. Therefore, a new local game is instantiated in room r_4 with the specification $\varphi^0_{r_4} = \square\lozenge e_{45}$. This process goes on until region 6 in room r_3 is reached. The complete path of the robot is depicted in Fig. 4 (right). In this particular simulation the door d_{56} also got shut when the robot was approaching it, causing it to move to r_2 instead.

6.2 Comparison to a monolithic solution

To illustrate how our hierarchical algorithm reduces the state explosion problem in synthesis, we consider a second example depicted in Fig. 5 where the robot starts in room r_1 and should go to region 5 in room r_6. All rooms are divided into 9 sub-regions and additional exit regions. We now add doors to every connection between two adjacent rooms which we handle as environment variables in level 0. Note that there are two doors $d^0_{ij,a}$ and $d^0_{ij,b}$ for every two adjacent rooms r_i and r_j. Furthermore, we introduce door-like predicates d^1_{ij} in the first layer which get activated if both $d^0_{ij,a}$ and $d^0_{ij,b}$ are true. We compare this hierarchical game to a monolithic one which contains the 54 level 0 grid cells (all level 0 grid cells which are not an exit cell) and all level 0 doors as environment predicates.

For both the hierarchical and the monolithic setup we have measured the influence of adding doors on the time needed to synthesize a solution, i.e., a winning play for the robot going from r_1 to r_6, for the special case that doors are never shut. These times were created as the average runtime of 10 synthesis trails in a virtual machine running Ubuntu 14.04LTS with 2GB of RAM and 2 CPUs on a Macbook Air 2013 with an Intel Core i7 (1,7Ghz) and 8GB of RAM. The results are depicted in Table 1. Going from left to right in Table 1, we have gradually added a set of doors D_{ij} to all games, denoting that two doors $d^0_{ij,a}$ and $d^0_{ij,b}$ have been added in the monolithic game and the local games of room r_i and r_j and that the door $d^1_{i,j}$ has been added in the level 1 game. As all subgames in level 0 have almost an identical number of grid cells we have not experienced noticeable differences in computation time between those, when they were generated with the same number of doors. We therefore use the representative values of 0.7sec, 0.8sec, and 1.4sec for a level 0 subgame with 0, 2, and 4 doors, respectively. With these values the approximate sum of level 0 computation times (second line in Table 1) can be easily calculated. For example, the value in the fourth column ("$+D_{34}$") is given by $5.8 = 2 \times 0.7 + 2 \times 0.8 + 2 \times 1.4$ as r_5 and r_6 have no doors, r_1 and r_4 have two doors each, and r_2 and r_3 have four doors each.

We see that the monolithic solution outperforms the hierarchical one when less then 5 doors are added to the game. This is not surprising: the hierarchical algorithm has additional overhead. However, we see that if the number of predicates grows beyond this point the computation times for the monolithic game grow very fast, while the computation times for

Table 1 Comparison of computation times (in seconds) for the hierarchical (top) and the monolithic (bottom) solution to the game depicted in Fig. 5

	$\emptyset$ (#0)	$+D_{12}$ (#2)	$+D_{23}$ (#4)	$+D_{34}$ (#6)	$+D_{45}$ (#8)	$+D_{56}$ (#10)
$l = 1$	0.5	0.6	0.7	0.9	1.1	1.5
$l = 0$ (sum)	4.2	4.4	5.1	5.8	6.5	7.2
hierarchical (sum)	4.7	5.0	5.8	6.7	7.6	8.7
monolithic	1.9	2.2	3.9	13.4	failed	failed

The column "$+D_{ij}$" shows the effect of adding two doors $d^0_{ij,a}$ and $d^0_{ij,b}$ to the monolithic game and the local games of room r_i and r_j and adding door $d^1_{i,j}$ in the level 1 game. "#k" denotes that this results in k doors in the monolithic game

the hierarchical case grow at a small constant rate. In particular, the current implementation of LTLMoP fails to synthesize a solution when more then 7 doors are present, due to an out-of-memory exception.[4] Intuitively, this is due to the exponential blow up of the state space caused by tracking the predicates in the state space. This blow up is more severe in the monolithic case as all the predicates are always added to the overall state space. On the contrary, in the hierarchical algorithm, predicates are added "locally" to a small number of sub-games at each time, which are only partially intersecting. Thus, the blow up is less severe.

We emphasize that doors are only a special kind of predicate. It is very likely for a game to have even more predicates, modeling for example that the robot is carrying something or that a cell is occupied by a moving obstacle. As long as these predicates are only used in a small subset of local games, our hierarchical algorithm scales significantly better then a monolithic solution.

7 Conclusion

We have shown in this paper how a large-scale reactive controller synthesis problem with intrinsic *hierarchy* and *locality* can be modeled as a hierarchical two player game over a set of local game graphs w.r.t. to a set of local strategies on multiple, interacting abstraction layers. We have proposed a reactive controller synthesis algorithm for such hierarchical games that allows for *dynamic specification changes* at each step of the play which is recalculated online in every step. This re-calculation becomes computationally tractable by the proposed decomposition. We have shown that our algorithm is sound: whenever the environment meets its assumptions and all dynamically generated local games have a solution, the controller synthesis algorithm generates a winning hierarchical play for a given specification. If these assumptions do not hold, the algorithm terminates but the generated finite play does not violate the specification up to this point.

Acknowledgments Open access funding provided by Max Planck Society.

[4] Adding the door $d^0_{45,a}$ was completed after 73.3 seconds, while adding both doors $d^0_{45,a}$ and $d^0_{45,b}$ resulted in an out-of-memory exception after 144.1 seconds.

Appendix A: Additional lemmas

Lemma 1 *Let π be a play and κ^l its timescale transformation for level $l \in [0, L]$. For all $l \in [0, L-1]$, we have*

$$\forall k \in \mathrm{dom}(\kappa^{l+1}) \,.\, \exists m \in \mathrm{dom}(\kappa^l) \,.\, \kappa^{l+1}(k) = \kappa^l(m) \quad \textit{and} \quad \lceil \kappa^l \rceil \geq \lceil \kappa^{l+1} \rceil.$$

Proof We prove both statements by contradiction.

Take $k \in \mathrm{dom}(\kappa^{l+1})$ and define $n = \kappa^{l+1}(k)$. Assume that there exists no $m \in \mathrm{dom}(\kappa^l)$ s.t. $n = \kappa^l(m)$. This implies, by the definition of κ^l in (7a), that $y^l(n-1) = y^l(n)$. However, this implies (by definition of layers) that $y^{l+1}(n-1) = y^{l+1}(n)$, which is a contradiction as the assumption $n = \kappa^{l+1}(k)$ implies (from (7a)) that $y^{l+1}(n-1) \neq y^{l+1}(n)$.

Assume that there exists a $k \in \mathrm{dom}(\kappa^{l+1})$ s.t. $k > \lceil \kappa^l \rceil$ and $n = \kappa^{l+1}(k)$. As before, this implies $y^l(n-1) = y^l(n)$ and hence $y^{l+1}(n-1) = y^{l+1}(n)$ which is a contradiction to the assumption that $k \in \mathrm{dom}(\kappa^{l+1})$. □

Lemma 2 *For each game G, each play π of G and each $l \in [0, L]$, we have*

$$\forall m \in \mathrm{dom}(\breve{\pi}^l), n \in (\kappa^l(m), \kappa^l(m+1)] \,.\, \begin{pmatrix} x^l(n) \in \delta^l(\breve{x}^l(m), \breve{y}^l(m)) \\ \wedge \breve{y}^l(m+1) \in \rho^l(\breve{x}^l(m+1), \breve{y}^l(m)) \end{pmatrix}. \quad (35)$$

Proof Pick $l \in [1, L]$ and $m \in \mathrm{dom}(\breve{\pi}^l)$ s.t. $m < \mathsf{end}(\kappa^l)$ and $\pi' = \pi\,|_{\kappa^l(m), \mathsf{end}(\pi)}$ and $\pi'' = \pi\,|_{\kappa^l(m+1)-1, \mathsf{end}(\pi)}$. Observe that $\pi', \pi'' \in \mathcal{G}$ by definition and we denote by κ'^l and κ''^l their respective timescale transformations defined via (7a). Observe that $m < \mathsf{end}(\kappa^l)$ implies $\mathsf{end}(\kappa'^l), \mathsf{end}(\kappa''^l) > 0$. We therefore obviously have $n \in (0, \kappa'^l(1)]$ and observe from the construction of π' that

$$\begin{aligned} \pi'^l(\kappa'^l(0)) &= \pi^l(\kappa^l(m)) = \left(\breve{x}^l(m), \breve{y}^l(m)\right) \\ &\text{and} \\ \pi'^l(\kappa'^l(1)) &= \pi^l(\kappa^l(m+1)) = \left(\breve{x}^l(m+1), \breve{y}^l(m+1)\right). \end{aligned}$$

With this it immediately follows from (10a) that $x^l(n) \in \delta^l\left(\breve{x}^l(m), \breve{y}^l(m)\right)$. Observe that $m < \mathsf{end}(\kappa^l)$ implies that $\breve{y}^l(m) \neq \breve{y}^l(m+1)$. It furthermore follows from (7a) that $y^l(\kappa^l(m+1) - 1) = \breve{y}^l(m)$. Using these observations we have

$$\begin{aligned} \pi''^l(\kappa''^l(1) - 1) &= \pi^l(\kappa^l(m+1) - 1) = (x^l(\kappa''^l(1) - 1), \breve{y}^l(m)) \\ &\text{and} \\ \pi''^l(\kappa''^l(1)) &= \pi^l(\kappa^l(m+1)) = (\breve{x}^l(m+1), \breve{y}^l(m+1)). \end{aligned}$$

With this it immediately follows from (10b) that $\breve{y}^l(m+1) \in \rho^l\left(\breve{x}^l(m+1), \breve{y}^l(m)\right)$. □

Lemma 3 *Let* $[\mathbb{G}]$ *be a set of LGGs and* $\mathcal{G}^l_y$ *the set of plays in* G^l_y*. Furthermore, let* $\pi \in \mathcal{G}$ *and* $[\check{p}]_\pi$ *its induced set of local projected play sequences. Then it holds for all* $l \in [0, L-1]$ *and* $m \in \mathrm{dom}^+(\check{\pi}^{l+1})$ *that*

$$\begin{pmatrix} \forall k \in [\kappa_l^{l+1}(m-1), \kappa_l^{l+1}(m)) \,.\, \check{y}^l(k) \in Y^l_{\check{y}^{l+1}(m-1)\rceil} \\ \wedge \check{y}^l(\kappa_l^{l+1}(m)) \in Y^l_{\check{y}^{l+1}(m-1)\rfloor} \cap Y^l_{\check{y}^{l+1}(m)\rceil} \\ \wedge \check{p}^l_\downarrow(m-1) \in \mathcal{G}^l_{\check{y}^{l+1}}(m-1) \end{pmatrix} \tag{36a}$$

and for all $l \in [0, L-1]$ *that*

$$\begin{pmatrix} \forall k \in [\kappa_l^{l+1}(\mathsf{end}(\check{\pi}^{l+1})), \mathsf{end}(\check{\pi}^l)] \,.\, \check{y}^l(k) \in Y^l_{\lceil \check{y}^{l+1}\rceil\rceil} \\ \lceil \check{p}^l_\downarrow \rceil \in \mathcal{G}^l_{\lceil \check{y}^{l+1}\rceil} \end{pmatrix}. \tag{36b}$$

Proof As the proof of (36b) is a simplified version of the proof for (36a), we only give the latter. We fix $l \in [0, \mathcal{L}-1]$, $m \in \mathrm{dom}\kappa^{l+1}$ and $k \in [\kappa_l^{l+1}(m-1), \kappa_l^{l+1}(m))$ and prove all lines of the statement separately. To simplify notation we use $\nu := \check{y}^{l+1}(m-1)$ and $\nu' := \check{y}^{l+1}(m)$.

- Pick $r := \kappa^l(k)$ and $r' := \kappa^{l+1}(m)$ and observe that $r \in [\kappa^{l+1}(m-1), \kappa^{l+1}(m))$. With this choice, (7a), (8) and (5a) imply

$$y^{l+1}(r) = \nu \neq \nu' = y^{l+1}(r'), \tag{37a}$$
$$y^{l+1}(r) = \alpha_s^{l+1}(y^l(r)) \text{ and } y^{l+1}(r') = \alpha_s^{l+1}(y^l(r')). \tag{37b}$$

  Substituting $y^l(r) = \check{y}^l(k)$ and $y^l(r') = \check{y}^l(\kappa_l^{l+1}(m))$ in (37b) and using (12c) gives

$$\check{y}^l(k) \in Y^l_{\nu\rceil}, \quad \text{and} \quad \check{y}^l\left(\kappa_l^{l+1}(m)\right) \in Y^l_{\nu'\rceil}, \tag{38}$$

  where the left side of (38) proves the first line of (36a).
- Recall from Prop. 4 that $\check{\pi}^l \in \mathcal{G}^l$. Using Def. 1 this implies that

$$\check{x}^l(k+1) \in \delta^l\left(\check{x}^l(k), \check{y}^l(k)\right) \quad \text{and} \quad \check{y}^l(k+1) \in \rho^l\left(\check{x}^l(k+1), \check{y}^l(k)\right). \tag{39a}$$

  Using the left side of (38) and Ass. 1, (39a) implies

$$\check{y}^l(k+1) \in \rho^l\left(\mathfrak{r}^l_\nu(\check{x}^l(k+1)), \check{y}^l(k)\right) = \rho^l\left(\check{x}^l_\downarrow(k+1), \check{y}^l(k)\right). \tag{39b}$$

  As $\check{x}^l_\downarrow(k+1) = \mathfrak{r}^l_\nu(\check{x}^l(k+1)) \in X^l{}_\nu$ (from (12a)) it follows from (39b) and (12d) that

$$\check{y}^l(\kappa_l^{l+1}(m)) \in Y^l_{\nu\rfloor}. \tag{39c}$$

  Combining (39c) with the right side of (38) proves the second line of (36a).
- Using (38), (39c), (12a) and (39a) in (13a) implies that

$$\check{x}^l_\downarrow(k+1) \in \delta^l_\nu\left(\check{x}^l_\downarrow(k), \check{y}^l(k)\right) \text{ and } \check{y}^l(k+1) \in \rho^l_\nu\left(\check{x}^l_\downarrow(k+1), \check{y}^l(k)\right), \tag{40}$$

  hence, the third line of (36a) holds.

□

Lemma 4 *Let π be a maximal play computed by (25). Then it holds for all $l \in [0, L-1]$ and $k \in \mathrm{dom}\pi$ that*

$$\begin{pmatrix} \neg \mathrm{UnRealizable}^l(k) \\ \wedge \neg\, \mathrm{GotStuck}^{l+1}(k) \end{pmatrix} \Leftrightarrow (\check{\gamma}^l(k), x^l_{\downarrow}(k+1)) \in \mathrm{dom}(f^l(k)) \tag{41}$$

if $\neg\, \mathrm{Done}^{l+1}(k)$.

Proof "$\Rightarrow$" The left side of (41) and (26) implies $\neg\, \mathrm{GotStuck}^l(k)$ and $\neg\, \mathrm{Done}^{l+1}(k)$ implies $\neg\, \mathrm{Done}^l(k)$ from (25k). Using both observations in (25l) implies $(\check{\gamma}^l(k), x^l_{\downarrow}(k+1)) \in \mathrm{dom}\, f^l(k)$. "$\Leftarrow$" The right side of (41) implies $f^l(k) \neq \emptyset$. Therefore, it follows from (25f) that $\neg\, \mathrm{GotStuck}^{l+1}(k)$ and (as $\neg\, \mathrm{Done}^l(k)$) from (25l) $\neg\, \mathrm{GotStuck}^l(k)$. Using both observations in (26) also gives $\neg\, \mathrm{UnRealizable}^l(k)$. □

Lemma 5 *Let π be a maximal play computed by (25a). Then it holds for all $k \in \mathrm{dom}(\pi)$ that*

$$\left(\exists l \in [0, L]\ .\ \mathrm{UnRealizable}^l(k)\right) \Leftrightarrow \mathrm{GotStuck}^0(k). \tag{42}$$

Proof "$\Rightarrow$": Pick l s.t. $\mathrm{UnRealizable}^l(k)$ and observe that this implies $\mathrm{GotStuck}^l(k)$ (from (26)) and hence $\neg\, \mathrm{Done}^l(k)$ (from (25l)). Using the first line of (25f) this implies $f^{l-1}(k) = \emptyset$. As $\neg\, \mathrm{Done}^l(k)$ also implies $\neg\, \mathrm{Done}^{l-1}(k)$ from (25k) it follows from Eq. 25l that $\mathrm{GotStuck}^{l-1}(k)$ is true (i.e., $\mathrm{GotStuck}^l(k) \Rightarrow \mathrm{GotStuck}^{l-1}(k)$). Applying this reasoning repetitively we eventually obtain $\mathrm{GotStuck}^0(k)$.
"$\Leftarrow$": Using (26), $\mathrm{GotStuck}^0(k)$ implies that the right side of (41) in Lem. 4 is false. Hence, either $\mathrm{UnRealizable}^0(k)$ or $\mathrm{GotStuck}^1(k)$ is true. If $\mathrm{UnRealizable}^0$ is true the statement is proven. We therefore assume that $\mathrm{GotStuck}^1(k)$ is true. We can reuse the same reasoning to either eventually get $\mathrm{UnRealizable}^l$ for some $l \in [0, L]$ (what proves the statement) or reach $\mathrm{GotStuck}^L(k)$. However, it follows from (26) that the latter is equivalent to $\mathrm{UnRealizable}^L$, what proves the statement.

□

Lemma 6 *Let $\pi = (x, y) \in \mathcal{G}^l{}_\nu$ for some $\nu \in Y^{l+1}$ s.t. $y(0) \in Y^l_{\nu\rceil}$ and $\psi^l_\nu(\nu')$ with $\nu' \in Y^{l+1}$, $\nu \neq \nu'$ as in (20). Then it holds that*

$$\pi \in \psi^l_\nu(\nu') \Leftrightarrow \begin{pmatrix} \mathsf{end}(\pi) < \infty \\ \wedge \forall k < \mathsf{end}(\pi)\ .\ y(k) \in Y^l_{\nu\rceil} \\ \wedge \lceil y \rceil \in Y^l_{\nu'\rceil} \cap Y^l_{\nu'\rceil} \end{pmatrix}. \tag{43}$$

Proof "$\Leftarrow$" $\langle$(43).right.1$\rangle$ and $\langle$(43).right.3$\rangle$ immediately imply that $\pi \in \psi^l_\nu(\nu')$ (from the first line of (20)). "$\Rightarrow$" $\langle$(43).right.2$\rangle$ is the only non-obvious conclusion from (20). Recall that $\pi \in \mathcal{G}^l{}_\nu$ and $y(0) \in Y^l_{\nu\rceil}$. Therefore it holds for all $r \leq \mathsf{end}(\pi)$ that $y^l_\nu(r) \in Y^l_{\nu\rceil} \cup Y^l_{\nu\lfloor}$. Now assume that there exists $r' < \mathsf{end}(\pi)$ s.t. $y(r') \in Y^l_{\nu\lfloor}$. Using (12b) this would imply $y(r') \notin Y^l_{\nu\rceil}$ and therefore from (13b) there exist no $\tilde{x}, \tilde{y}$ s.t. $\tilde{y} \in \rho^l_\nu\left(\tilde{x}, y(r')\right)$, implying $r' = \mathsf{end}(\pi)$ which is a contradiction to the assumption. □

Lemma 7 *Let π be a play computed by (25) up to length $\mathsf{end}(\pi)$. Then it holds for all $l \in [1, L]$ and $k < \kappa^l(\mathsf{end}(\breve{\pi}^l))$ that $\neg \mathrm{Done}^l$.*

Proof We prove the statement by contradiction. Pick any $l \in [1, L]$ and $k < \kappa^l(\mathsf{end}(\breve{\pi}^l))$ and assume that Done^l is true. First observe that this implies $\mathrm{Done}^{l'}$ for all $l' \in [l, L]$. With this it follows from (25f) that $f^l(k) = h^l$. Now using (25k) this implies that $(\breve{\gamma}^l(k), x^l_\downarrow(k+1)) \notin \mathrm{dom}(f^l(k))$ and therefore the play would not be able to leave the current context. This is a contradiction to the assumption that $k < \kappa^l(\mathsf{end}(\breve{\pi}^l))$, what proves the statement. □

Lemma 8 *Let π be a play computed by (25) up to length $\mathsf{end}(\pi)$. Then it holds for all $l \in [0, L]$ and $m \in \mathrm{dom}^+(\breve{\pi}^l)$ that*

$$\breve{x}^l_\downarrow(m \in \delta^l_{\breve{y}^l(m-1)}\left(\breve{x}^l_\downarrow(m-1), \breve{y}^l(m-1)\right) \text{ and} \tag{44a}$$

$$\breve{y}^l(m) = f^l(\kappa^l(m)-1)\left(\breve{\gamma}^l(\kappa^l(m)-1), \breve{x}^l_\downarrow(m)\right). \tag{44b}$$

Proof Recall that $\pi \in \mathcal{G}$ from Prop. 4. Therefore, (44a) follows directly from (40) in Lem. 3. We show (44b) by induction.

- $l = 0$:
  Recall that (27) holds for $l = 0$. As κ^0 is the identity map, the second line in (27) and (44a) is equivalent for $l = 0$.
- $l \to l+1$:
  - Pick $m \in \mathrm{dom}^+(\breve{\pi}^{l+1})$, $k = \kappa^{l+1}(m)$, $\nu := \breve{y}^{l+1}(m-1)$ and $\nu' := \breve{y}^{l+1}(m)$ and recall from Lem. 1 that there exists $r \in \mathbb{N}$ s.t. $r = \kappa^{l+1}_l(m)$ and $\kappa^l(r) = k$, implying (from (7a)) that

$$\begin{aligned} &y^{l+1}(k-1) = \nu \neq \nu' = y^{l+1}(k),\ y^l(k) = \breve{y}^l(r), \\ &\breve{x}^{l+1}_\downarrow(m) = x^{l+1}_\downarrow(k), \text{ and } \breve{x}^l_\downarrow(r) = x^l_\downarrow(k). \end{aligned} \tag{45a}$$

    Now it follows from Lem. 3 that $\breve{p}^l_\downarrow(m-1) \in \mathcal{G}^l_\nu$, hence Lem. 6 holds for $\breve{p}^l_\downarrow(m-1)$. Now using the first and second line of (36a) in Lem. 6 immediately implies

$$\breve{p}^l_\downarrow(m-1) \in \psi^l_\nu(\nu'). \tag{45b}$$

  - We now show that $\breve{p}^l_\downarrow(m-1)$ is compliant with $f^l(k-1)$:
    As (44a) holds for l we know that for all $k' < \mathrm{end}\pi$ we have $\neg\,\mathrm{GotStuck}^{l+1}(k')$ (from Lem. 4). As additionally $\neg\mathrm{Done}^{l+1}(k')$ from Lem. 7, (25f) gives that

$$f^l(k') = f^l_{y^{l+1}(k')\nu^{l+1}}(k') \tag{45c}$$

$$\text{s.t. } \nu^{l+1}(k') = f^{l+1}(k')(\breve{\gamma}^{l+1}(k'), x^{l+1}_\downarrow(k'+1)). \tag{45d}$$

    Now pick s s.t.

$$\forall k', k'' \in [\kappa^l(r-s), \kappa^l(r)-1]\ .\ y^{l+1}(k') = y^{l+1}(k'') \wedge \nu^{l+1}(k') = \nu^{l+1}(k''), \tag{45e}$$

with ν^{l+1} as in (45d) and observe that this implies $\kappa^l(r-s) \in [\kappa^{l+1}(m-1), \kappa^{l+1}(m))$. Using (45e) in (25i) therefore gives for all $k' \in [\kappa^l(r-s), \kappa^l(r)-1]$ that

$$f^l(k') = f^l(k-1) = \mathrm{Sol}\left(G^l_\nu, \{\check{\gamma}^l(\kappa^l(r-s))\}, \phi^l_\nu(\nu'^{l+1}(k-1))\right). \tag{45f}$$

As (44a) holds for l we can therefore substitute $f^l(\kappa^l(r)-1)$ in (44b) by $f^l(k-1)$ and obtain for all $r' \in [r-s, r-1]$ that

$$\check{y}^l(r') = f^l(k-1)(\check{\gamma}^l(\kappa^l(r')-1), \check{x}^l_\downarrow(r')). \tag{45g}$$

It furthermore follows from the construction of $\check{\gamma}^l$ in (25o) and $\check{p}^l_\downarrow$ in (16a) that

$$\check{p}^l_\downarrow(m-1) = \check{\gamma}^l(\kappa^l(r-s)) \cdot \check{\pi}^l_\downarrow\,|_{[r-s+1,r]} \tag{45h}$$

Now pick $n = \mathsf{end}(\check{\gamma}^l(\kappa^l(r-s)))$ and observe that $\check{p}^l_\downarrow(m-1)\,|_{0,n} \in \{\check{\gamma}^l(\kappa^l(r-s))\}$. Additionally using (45g) therefore implies that $\check{p}^l_\downarrow(m-1) \in \mathsf{CompliantPlays}(f^l(k-1), \{\check{\gamma}^l(\kappa^l(r-s))\})$ (from (2)). Using (45f) and (24b) from Prop. 3 it follows that

$$\check{p}^l_\downarrow(m-1) \in \overline{\phi^l_\nu(\nu^{l+1}(k-1))}. \tag{45i}$$

- It remains to shown that $\nu^{l+1}(k-1) = \nu' (= \check{y}^{l+1}(m) = y^{l+1}(k))$:
  Using the fact that $y^l(k) \in Y^l\nu$ it follows from Lem. 6 and (21) that (45b) and (45i) can only be satisfied simultaneously if

$$\check{p}^l_\downarrow(m-1) \in \phi^l_\nu(\nu^{l+1}(k-1)) \quad \text{and} \quad \nu^{l+1}(k-1) = \nu'. \tag{45j}$$

  With this observation (44b) immediately follows for $l+1$ from (45d) as $\nu^{l+1}(k-1) = \check{y}^{l+1}(m)$.

□

Lemma 9 *Let π be a play computed by (25) up to length $\mathsf{end}(\pi)$ and $[\check{p}]_\pi$ its induced set of local projected play sequences. Then it holds for all $l \in [0, L-1]$ and $m \in \mathrm{dom}^+(\check{\pi}^{l+1})$ that*

$$\check{p}^l_\downarrow(m-1) \in \phi^l_{\check{y}^{l+1}(m-1)}(\check{y}^{l+1}(m)) \tag{46a}$$

and for $m = \mathsf{end}(\check{\pi}^{l+1})$ there exists $\nu' \in \mathrm{Post}^{l+1}(\check{y}^{l+1}(m))$ s.t.

$$\check{p}^l_\downarrow(m) \in \overline{\phi^l_{\check{y}^{l+1}(m)}(\nu')}. \tag{46b}$$

Proof Equation (46a) follows from (45j) in the proof of Lem. 8. We prove (46b):

Pick $l \in [0, L-1]$ and $m = \mathrm{end}\check{y}^{l+1}$ and recall from Lem. 8 that (44a) holds for all $k \in \mathrm{dom}^+(\check{\pi}^l)$. Therefore $\neg\,\mathrm{GotStuck}^{l+1}(k')$ (from Lem. 4) for all $k' < \mathrm{end}(\pi)$. Now we have two cases.

(i) If $\neg\text{Done}^{l+1}(\kappa^{l+1}(m))$, (45d) in the proof of Lem. 8 holds for $k' \in [\kappa^{l+1}(m), \kappa^l(\text{end}(\breve{\pi}^l))]$. Following exactly the same reasoning as in (45d)–(45i) we obtain

$$\breve{p}^l_{\downarrow}(m) \in \overline{\phi^l_{\breve{y}^{l+1}(m)}(\nu^{l+1}(k-1))}$$

with $\nu^{l+1}(k-1)$ as in (45d), implying (46b).

(ii) If $\text{Done}^{l+1}(\kappa^{l+1}(m))$, it follows from (25f) and (25h) that for $k' \in [\kappa^{l+1}(m), \kappa^l(\text{end}(\breve{\pi}^l))]$

$$f^l(k') = h^l(\kappa^{l+1}(m)) = \text{Sol}\left(G^l_{\breve{y}^{l+1}(m)}, \{\breve{\gamma}^l(\kappa^{l+1}(m))\}, \varphi^l_{\breve{y}^{l+1}(m)}, \zeta^l_v\right) \tag{47}$$

and from the construction of $\breve{\gamma}^l$ and $\breve{p}^l_{\downarrow}$ in (25o) and (16a) that $\breve{\gamma}^l(\kappa^{l+1}(m)) = \lceil \breve{p}^l_{\downarrow}(m-1) \rceil$. By substituting (47) in (44b) we therefore obtain $\breve{p}^l_{\downarrow}(m) \in \mathsf{CompliantPlays}(h^l(\kappa^{l+1}(m)), \lceil \breve{p}^l_{\downarrow}(m-1) \rceil)$ (from (2)). Using (45f) and (24b) from Prop. 3 it follows that $\breve{p}^l_{\downarrow}(m) \in \overline{\varphi^l_{\breve{y}^{l+1}(m)}}$. Now recall from (21) that $\overline{\varphi^l_{\breve{y}^{l+1}(m)}} \subseteq \overline{\phi^l_{\breve{y}^{l+1}(m)}(\nu^{l+1}(k-1))}$, what proves the statement.

□

Lemma 10 *Let π be a maximal and environment admissible play computed by (25a) s.t. (33) holds. Then it holds that*

$$(\exists k \in \text{dom}(\pi), l \in [0, L]\text{Done}^l(k)) \Rightarrow (\exists k' \in \text{dom}(\pi), k' \geq k \,.\, \text{Done}^0(k')).$$

Proof Pick $k \in \text{dom}(\pi), l \in [0, L]$ s.t. $\text{Done}^l(k)$ and assume $l > 0$ as for $l = 0$ the statement follows trivially. Giving $\text{Done}^l(k)$, (25l) implies $\neg\,\text{GotStuck}^l(k)$ and (25f) implies $f^{l-1}(k) = h^{l-1}(k)$. Giving $\neg\,\text{UnRealizable}^{l-1}(k)$ and $\neg\text{GotStuck}^l(k)$, (26) implies $\neg\text{GotStuck}^{l-1}(k)$ and therefore (from (25l)) either $\text{Done}^{l-1}(k)$ or there exists a next step according to $h^{l-1}(k)$. Assume the latter is true. Recall from (25o) that $h^{l-1}(k)$ is an assume admissible winning strategy for the game $\left(G^l_{y^l(k)}, \{\breve{\gamma}^{l-1}(\lceil \kappa^{l-1} \rceil)\}, \varphi^l_{y^l(k)}\right)$ and from (18) that $\varphi^l_{y^l(k)}$ only contains finite strings. If the environment plays admissible, we therefore eventually obtain $\text{Done}^{l-1}(k')$ with $k < k' < \infty$. Applying this reasoning iteratively, eventually leads to $\text{Done}^0(k'')$ where the time between k and k'' is ensured to be finite. □

Lemma 11 *Let π be a maximal and environment admissible play computed by (25a) s.t. (33) holds. Then it holds that*

$$(\forall k \in \text{dom}(\pi), l \in [0, L]\neg\text{Done}^l(k)) \Leftrightarrow (\forall l \in [0, L]|\breve{\pi}^l| = \infty) \Leftrightarrow (|\pi| = \infty)\,.$$

Proof We show this proof in two steps.

► Show $\left(\forall k \in \text{dom}(\pi), l \in [0, L]\neg\text{Done}^l(k)\right) \Leftrightarrow (|\pi| = \infty)$:
Using (33) in (29) of Thm. 1 gives

$$\exists l \in [0, L] \,.\, \forall k \in \text{dom}(\pi) \,.\, \neg\text{Done}^l(k) \Leftrightarrow |\pi| = \infty, \tag{48}$$

immediately implying the "⇒" part of the statement. Now we prove the "⇐" part by contradiction. Assume that there exists $l \in [0, L]$, $k \in \text{dom}\pi$ s.t. $\text{Done}^l(k)$. Then Lem.

10 implies $\text{Done}^0(k')$. Using (from (25k)) this implies $\text{Done}^l(k')$ for all $l \in [0, L]$, which gives a contradiction as the left side of (48) holds from $(|\pi| = \infty)$.

▶ Show $\forall l \in [0, L]\ .\ |\breve{\pi}^l| = \infty \Leftrightarrow |\pi| = \infty$:

First observe that "⇒" trivially holds as $\breve{\pi}^0 = \pi$. We prove "⇐" by contradiction. Assume there exists $l \in [0, L]$ s.t. $|\breve{\pi}^l| < \infty$, i.e., with $k = \text{end}(\breve{\pi}^l)$ we have $(\breve{\gamma}^l(k), x^l_{\downarrow}(k+1)) \notin \text{dom}(f^l(k))$. Now recall from the first part of this proof that $|\pi| = \infty$ implies $\neg\,\text{Done}^{l'}(k)$ for all $l' \in [0, L]$ and (33) implies $\neg\text{UnRealizable}^{l+1}(k)$. Then it follows from Lem. 4 that $\text{GotStuck}^{l+1}(k)$. With this $\text{GotStuck}^l(k)$ (from (25f)) and therefore eventually $\text{GotStuck}^0(k)$, which implies $|\pi| < \infty$ with $\mathsf{end}(\pi) = k$, which is a contradiction to the assumption.

□

References

Abadi M, Lamport L (1991) The existence of refinement mappings. Theor Comput Sci 82(2):253–284

Alur R, La Torre S, Madhusudan P (2003) Modular strategies for recursive game graphs. In: TACAS, volume 2619 of LNCS, pp 363–378

Bloem R, Jobstmann B, Piterman N, Pnueli A, Saar Y (2012) Synthesis of reactive(1) designs. J Comput Syst Sci 78(3):911–938

Bloem R, Ehlers R, Jacobs S, Könighofer R (2014) How to handle assumptions in synthesis. In: SYNT 2014, Vienna, Austria, pp 34–50

Bloem R, Ehlers R, Könighofer R (2015) Cooperative reactive synthesis. In: ATVA Shanghai, China, pp 394–410

Brenguier R, Raskin J-F, Sassolas M (2014) The complexity of admissibility in omega-regular games. In: CSL-LICS, vol 23, pp 1–10

Brenguier R, Raskin J-F, Sankur O (2015) Assume-admissible synthesis. In: CONCUR 2015, volume 42 of LIPIcs. Dagstuhl, Germany, pp 100–113

Cousot P, Cousot R (1977) Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL 77. ACM, pp 238–252

De Crescenzo I, La Torre S (2013) Modular synthesis with open components. In: Reachability problems, volume 8169 of LNCS, pp 96–108

Emerson E, Jutla C (1991) Tree automata, mu-calculus and determinacy. In: FOCS, pp 368–377

Erol K, Hendler JA, Nau DS (1995) Semantics for hierarchical task-network planning. Technical report, University of Maryland

Finucane C, Jing G, Kress-Gazit H (2010) LTLMop experimenting with language, temporal logic and robot control. In: IROS, pp 1988–1993

Girard A, Pappas G (2009) Hierarchical control system design using approximate simulation. Automatica 45(2):566–571

Henzinger TA, Majumdar R, Mang F, Raskin J-f (2000) Abstract interpretation of game properties. In: Static analysis, LNCS, vol 1824, pp 220–239

Hess D, Althoff M, Sattel T (2014) Formal verification of maneuver automata for parameterized motion primitives. In: IROS, pp 1474–1481

Kaelbling L, Lozano-Perez T (2011) Hierarchical task and motion planning in the now. In: ICRA, pp 1470–1477

Kloetzer M, Belta C (2008) Dealing with nondeterminism in symbolic control. In: HSCC, volume 4981 of LNCS, pp 287–300

Koo T, Sastry S (2002) Bisimulation based hierarchical system architecture for single-agent multi-modal systems. In: HSCC, volume 2289 of LNCS, pp 281–293

Kruger N, Piater J, Worgotter F, Geib C, Petrick R, Steedman M, Asfour T, Kraft D, Hommel B, Agostini A et al (2009) A formal definition of object-action complexes and examples at different levels of the processing hierarchy. Computer and Information Science, pp 1–39

Leva A (2016) Reactive controller synthesis for mobile robotics. Master's thesis, University of Kaiserslautern, Germany. Available at https://www.mpi-sws.org/tr/2017-001.pdf

Mazo J, Manuel A, Tabuada P (2010) PESSOA Davitian A tool for embedded controller synthesis. In: CAV, volume 6174 of LNCS, pp 566–569
Pappas G, Lafferriere G, Sastry S (2000) Hierarchically consistent control systems. IEEE Trans Autom Control 45(6):1144–1160
Raisch J, Moor T (2005) Hierarchical hybrid control synthesis and its application to a multiproduct batch plant. In: Control and observer design for nonlinear finite and infinite dimensional systems, volume 322 of LNCS, pp 199–216
Reps T, Horwitz S, Sagiv S (1995) Precise interprocedural dataflow analysis via graph reachability. In: POPL 95. ACM, pp 49–61
Schmidt K, Moor T, Perk S (2008) Nonblocking hierarchical control of decentralized discrete event systems. IEEE Trans Autom Control 53(10):2252–2265
Srivastava S, Fang E, Riano L, Chitnis R, Russell S, Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: ICRA, pp 639–646
Stock S, Mansouri M, Pecora F, Hertzberg J (2015) Hierarchical hybrid planning in a mobile service robot. In: KI 2015: Advances in artificial intelligence, pp 309–315
Tabuada P (2009) Verification and control of hybrid systems - a symbolic approach, vol 1. Springer
Vasile CI, Belta C (2014) Reactive sampling-based temporal logic path planning. In: ICRA, pp 4310–4315
Walukiewicz I (1996) Pushdown processes: Games and model checking. In: CAV 96: computer-aided verification, LNCS 1102, pp 62–74
Wolff E, Topcu U, Murray R (2013) Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. In: CDC, pp 3197–3204
Wong KW, Finucane C, Kress-Gazit H (2013) Provably-correct robot control with ltlmop, ompl and ros. In: IROS, pp 2073–2073
Wongpiromsarn T, Topcu U, Murray R (2010) Automatic synthesis of robust embedded control software. In: AAAI Spring Symposium: Embedded Reasoning
Wongpiromsarn T, Topcu U, Ozay N, Xu H, Murray R (2011) TuLiP: a software toolbox for receding horizon temporal logic planning. In: HSCC, pp 313–314
Wongpiromsarn T, Topcu U, Murray R (2012) Receding horizon temporal logic planning. IEEE Trans Autom Control 57(11):2817–2830
Zielonka W (1998) Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor Comput Sci 200(1-2):135–183

Anne-Kathrin Schmuck received the Dipl.-Ing. (M.Sc) degree in engineering cy- bernetics from OvGU Magdeburg, Germany, in 2009 and the Dr.-Ing. (Ph.D.) degree in electrical engineering from TU Berlin, Germany, in 2015. She is currently a post- doctoral researcher at the MPI-SWS in Kaiserslautern, Germany. From 2010 to 2015 she was a research assistant in the Control Systems Group at TU Berlin, Germany. During her studies she was a visiting student at UBC in Vancouver, Canada, LTH in Lund, Sweden, and UCLA in Los Angeles, USA. Her current research interests include abstraction based controller synthesis, reactive synthesis, supervisory control theory and hierarchical control.

Rupak Majumdar received the B.Tech. degree in computer science from the Indi- an Institute of Technology, Kanpur, India, in 1998 and the Ph.D. degree in computer science from the University of California at Berkeley CA, USA, in 2003. He is current- ly a Scientific Director at the Max Planck Institute for Software Systems. Previously, he was a Professor in the Department of Computer Science, University of California at Los Angeles, Los Angeles, USA. His research interests are in the verification and control of reactive, real-time, hybrid, and probabilistic systems, software verification and programming languages, logic, and automata theory.

Adrian Leva did his Bachelor degree in Computer Science at the University of Kaiserslautern. He has done the consecutive Master in Computer Science and finished his master's thesis in cooperation with the Max Planck Institute for Software Systems in the field of reactive controller synthesis.