Acta Informatica manuscript No.
(will be inserted by the editor)

Hierarchical Information and the Synthesis of Distributed Strategies

Dietmar Berwanger · Anup Basil Mathew · Marie van den Bogaard

Received: date / Accepted: date

Abstract Infinite games with imperfect information are known to be undecidable unless the information flow is severely restricted. One fundamental decidable case occurs when there is a total ordering among players, such that each player has access to all the information that the following ones receive. In this paper we consider variations of this hierarchy principle for synchronous games with perfect recall, and identify new decidable classes for which the distributed synthesis problem is solvable with finite-state strategies. In particular, we show that decidability is maintained when the information hierarchy may change along the play, or when transient phases without hierarchical information are allowed. Finally, we interpret our result in terms of distributed system architectures.

Keywords Infinite games · Imperfect information · Coordination · Distributed systems · Automated synthesis

Mathematics Subject Classification (2000) 91A06 · 68M14 · 93B50

1 Introduction

To realise systems that are correct by design is a persistent ambition in computing science. The stake is particularly high for systems that interact with an unpredictable environment over indeterminate time. Pioneering results in the area of synthesis, due to Büchi and Landweber (1969), and Rabin (1972), show that the task can be automatised for the case of monolithic designs with correctness conditions specified by automata over infinite objects — words or trees representing computations. A most natural framework for representing and solving the problem is in terms of infinite games with perfect information over finite graphs, as described by Pnueli and Rosner (1990) or by Thomas (1995).

For distributed systems in which multiple components interact with the objective of satisfying a global specification, the game-theoretical formulation of the

D. Berwanger, A. B. Mathew, and M. van den Bogaard
LSV, CNRS & Université Paris-Saclay, France
E-mail: dwb@lsv.fr

synthesis problem leads to games with imperfect information and to the question of whether there exists a winning strategy that can be distributed among the multiple players. Unfortunately, such games are much less amenable to automated solutions: as pointed out by Peterson and Reif (1979), it is generally undecidable whether a solution — that is, a distributed winning strategy — exists for a finitely presented game for two players against Nature (or the environment); furthermore, Janin (2007) showed that, even if a solution exists, it may not be implementable by a finite-state device. As there is no hope for solving the distributed synthesis problem uniformly, it remains to look out for classes that allow for an algorithmic treatment. For surveys on results in this direction, see, e.g., the article of Gastin et al. (2009) or the theses of Schewe (2008) and of Puchala (2013).

One fundamental case in which the distributed synthesis problem becomes decidable is that of hierarchical systems: these correspond to games where there exists a total order among the players such that, informally speaking, each player has access to the information received by the players that come later in the order. For such games, Peterson and Reif (1979) showed that it is decidable — although, with nonelementary complexity — whether distributed winning strategies exist and if so, finite-state winning strategies can be effectively synthesised. The result was extended by Pnueli and Rosner (1990) to the framework of distributed system architectures with linear-time specifications over path-shaped communication architectures where information can flow only in one direction. Later, Kupferman and Vardi (2001) developed an automata-theoretic approach that allows to extend the decidability result from linear-time to branching-time specifications, and also relaxes some of the syntactic restrictions imposed by the fixed-architecture setting of Pnueli and Rosner. Finally, Finkbeiner and Schewe (2005) gave an effective characterisation of communication architectures on which distributed synthesis is decidable. The criterion requires absence of information forks, which implies a hierarchical order in which processes, or players, have access to the observations emitted by the environment.

The setting of games is more liberal than that of architectures with fixed communication channels. A rather general, though non-effective condition for games to admit finite-state distributed winning strategies is given in Berwanger et al. (2011), based on epistemic models representing the knowledge acquired by players in a game with perfect recall. This condition suggests that, beyond the fork-free architecture classification there may be further natural classes of games for which the distributed synthesis problem is decidable.

In this paper, we study a relaxation of the hierarchical information pattern underlying the basic decidability results on games with imperfect information and distributed system architectures. Firstly, we extend the assumption of hierarchical observation, that is, *positional* information, by incorporating perfect recall. Rather than requiring that a player *observes* the signal received by a less-informed player, we require that he can *infer* it from his observation of the play history. It can easily be seen that this gives rise to a decidable class, and it is likely that previous authors had a perfect-recall interpretation in mind when describing hierarchical systems, even if the formal definitions in the relevant literature generally refer to observations.

Secondly, we investigate the case when the hierarchical information order is not fixed, but may change dynamically along the play. This allows to model situations where the schedule of the interaction allows a less-informed player to become more

informed than others, or where the players may coordinate on designating one to receive certain signals, and thus become more informed than others. We show that this condition of dynamic hierarchical observation also leads to a decidable class of the distributed synthesis problem.

As a third extension, we consider the case where the condition of hierarchical information (based on perfect recall) is intermittent. That is, along every play, it occurs infinitely often that the information sets of players are totally ordered; nevertheless, there may be histories at which incomparable information sets arise, as it is otherwise typical of information forks. We show that, at least for the case of winning conditions over attributes observable by all players, this condition of recurring hierarchical observation is already sufficient to ensure decidability of the synthesis problem, and that finite-state winning strategies exist for all solvable instances.

For all the three conditions of hierarchical information, it is decidable with relatively low complexity whether they hold for a given game. However, the complexity of solving a game is nonelementary in all cases, as they are more general than the condition of hierarchical observation, for which it was shown by Peterson and Reif (1979) that there exists no elementary solution procedure.

The last part of the paper presents an interpretation of the game-theoretic results in terms of distributed reactive sytems. Towards this, we extend the framework introduced by Pnueli and Rosner (1990) which features hard-wired communciation graphs, to a model where the global actions are transduced into signals for the individual processes by a deterministic finite-state monitor. In this framework of monitored architectures, we identify classes that correspond to games with hierarchical information and therefore admit an effective solution of the distributed synthesis problem.

2 Preliminaries

2.1 Games on graphs

We use the standard model of concurrent games with imperfect information, following the notation from Berwanger et al. (2011). There is a set $N = \{1, \ldots, n\}$ of players and a distinguished agent called Nature. We refer to a list of elements $x = (x^i)_{i \in N}$, one for each player, as a *profile*. For each player i, we fix a set A^i of *actions* and a set B^i of *observations*; these are finite sets.

A *game graph* $G = (V, E, (\beta^i)_{i \in N})$ consists of a set V of nodes called *positions*, an edge relation $E \subseteq V \times A \times V$ representing simultaneous *moves* labelled by action profiles, and a profile of *observation* functions $\beta^i : V \to B^i$ that label every position with an observation for each player. We assume that the graph has no dead ends, that is, for every position $v \in V$ and every action profile $a \in A$, there exists an outgoing move $(v, a, w) \in E$, and we denote the set of successors of a position v by $vEA := \{w \mid (v, a, w) \text{ for some } a \in A\}$.

Plays start at a designated initial position $v_0 \in V$ and proceed in rounds. In a round at position v, each player i chooses simultaneously and independently an action $a^i \in A^i$, then Nature chooses a successor position v' reachable along a move $(v, a, v') \in E$. Now, each player i receives the observation $\beta^i(v')$, and the play continues from position v'. Thus, a *play* is an infinite sequence $\pi = v_0, v_1, v_2, \ldots$

of positions, such that for all $\ell \geq 0$, there exists a move $(v_\ell, a, v_{\ell+1}) \in E$. A *history* is a nonempty prefix $\pi = v_0, v_1, \ldots, v_\ell$ of a play; we refer to ℓ as the *length* of the history, and we denote by $\mathrm{Hist}(G)$ the set of all histories in the game graph G. The observation function extends from positions to histories[1] and plays as $\beta^i(\pi) = \beta^i(v_1)\beta^i(v_2)\ldots$, and we write $\mathrm{Hist}^i(G) := \{\beta^i(\pi) \mid \pi \in \mathrm{Hist}(G)\}$ for the set of *observation histories* of player i. We say that two histories π, π' are *indistinguishable* to player i, and write $\pi \sim^i \pi'$, if they yield the same observation $\beta^i(\pi) = \beta^i(\pi')$. This is an equivalence relation, and its classes are called *information sets*. The information set of player i at history π is $P^i(\pi) := \{\pi' \in \mathrm{Hist}(G) \mid \pi' \sim^i \pi\}$. In terms of the taxonomy for distributed systems by Halpern and Vardi (1989), our model is *synchronous* and of *perfect recall*.

A *strategy* for player i is a mapping $s^i : V^* \to A^i$ from histories to actions that is *information-consistent* in the sense that $s^i(\pi) = s^i(\pi')$, for any pair $\pi \sim^i \pi'$ of indistinguishable histories. We denote the set of all strategies of player i by S^i and the set of all strategy profiles by S. A history or play $\pi = v_0, v_1, \ldots$ *follows* the strategy $s^i \in S^i$ if, for every $\ell > 0$, we have $(v_\ell, a, v_{\ell+1}) \in E$ for some action profile a where $a^i = s^i(v_0, v_1, \ldots, v_\ell)$. The play π follows a strategy profile $s \in S$ if it follows all component strategies s^i. The set of possible *outcomes* of a strategy profile s is the set of plays that follow s.

A *winning condition* over a game graph G is a set $W \subseteq V^\omega$ of plays. A *distributed game* $\mathcal{G} = (G, W)$ is described by a game graph and a winning condition. We say that a play π is winning in $\mathcal{G}$ if $\pi \in W$. A strategy profile s is winning in $\mathcal{G}$ if all its possible outcomes are so. In this case, we refer to s as a *distributed winning strategy*. We generally assume that the game graph is finite or finitely presented. The general *distributed synthesis problem* asks whether for a given game graph and a fixed or given winning condition, there exists a distributed winning strategy.

2.2 Automata

Our focus is on finitely-represented games, where the game graphs are finite and the winning conditions described by finite-state automata. Specifically, winning conditions are given by a colouring function $\gamma : V \to C$ and an ω-regular set $W \subseteq C^\omega$ describing the set of plays $v_0, v_1, \ldots$ with $\gamma(v_0), \gamma(v_1), \cdots \in W$. In certain cases, we assume that the colouring is *observable* to each player i, that is, $\beta^i(v) \neq \beta^i(v')$ whenever $\gamma(v) \neq \gamma(v')$. For general background on automata for games, we refer to the handbook by Grädel et al. (2002).

Strategies shall also be represented as finite-state machines. A *Moore* machine over an input alphabet Σ and an output alphabet Γ is described by a tuple (M, m_0, μ, ν) consisting of a finite set M of *memory states* with an initial state m_0, a memory *update* function $\mu : M \times \Sigma \to M$ and an *output* function $\nu : M \to \Gamma$ defined on memory states. Intuitively, the machine starts in the initial memory state m_0, and proceeds as follows: in state m, upon reading an input symbol $x \in \Sigma$, it updates its memory state to $m' := \mu(m, x)$ and then outputs the letter $\nu(m)$. Formally, the update function μ is extended to input words in Σ^* by setting, $\mu(\varepsilon) := m_0$, for the empty word, and by setting, $\mu(x_0 \ldots x_{\ell-1} x_\ell) := \mu(\,\mu(x_0 \ldots x_{\ell-1}),\, x_\ell)$, for

[1] Note that we discard the observation at the initial position; this is technically convenient and does not restrict the model.

all nontrivial words $x_0 \dots x_{\ell-1} x_\ell$. This gives rise to the function $M : \Sigma^* \to \Gamma^*$ *implemented* by M, defined by $M(x_0, \dots, x_\ell) := \nu(\, \mu(\, x_0 \dots x_\ell)\,)$. A *strategy automaton* for player i on a game graph G, is a Moore machine M with input alphabet B^i and output alphabet A^i. The strategy implemented by M is defined as $s^i(v_0, \dots, v_\ell) := M(\beta^i(v_0 \dots v_\ell))$, for all $\ell > 0$. A *finite-state strategy* is one that can be implemented by a strategy automaton.

Sometimes it is convenient to refer to Mealy machines rather than Moore machines. These are finite-state machines of similar format, with the only difference that the output function $\nu : M \times \Sigma \to \Gamma$ is defined on transitions rather than their target state.

In the following we will refer to several classes $\mathcal{C}$ of finite games, always assuming that winning conditions are given as ω-regular languages. The *finite-state synthesis problem* for a class $\mathcal{C}$ is the following: Given a game $\mathcal{G} \in \mathcal{C}$,

(i) decide whether $\mathcal{G}$ admits a finite-state distributed winning strategy, and
(ii) if yes, construct a profile of finite-state machines that implements a distributed winning strategy for $\mathcal{G}$.

We refer to the set of distributed (finite-state) winning strategies for a given game $\mathcal{G}$ as the (finite-state) *solutions* of $\mathcal{G}$. We say that the synthesis problem is *finite-state solvable* for a class $\mathcal{C}$ if every game $\mathcal{G} \in \mathcal{C}$ that admits a solution also admits a finite-state solution, and if the above two synthesis tasks can be accomplished for all instances in $\mathcal{C}$.

3 Static Information Hierarchies

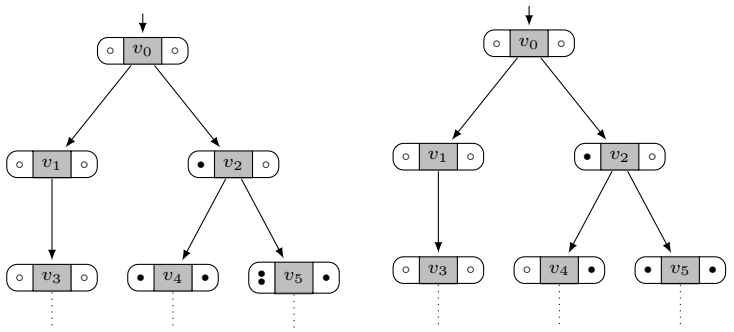
3.1 Hierarchical observation

We set out from the basic pattern of hierarchical information underlying the decidability results cited in the introduction. These results rely on a positional interpretation of information, i.e., on observations.

Definition 1 A game graph yields *hierarchical observation* if there exists a total order $\preceq$ among the players such that whenever $i \preceq j$, then for all pairs v, v' of positions, $\beta^i(v) = \beta^i(v')$ implies $\beta^j(v) = \beta^j(v')$

In other words, if $i \preceq j$, then the observation of player i determines the observation of player j. An example of such a situation is illustrated in Figure 1(a).

Peterson and Reif (1979) study a game with players organised in a hierarchy, such that each player i sees the data observed by player $i-1$. The setting is actually generic for games with reachability winning conditions and the authors show that winning strategies can be synthesised in n-fold exponential time and this complexity is unavoidable. Later, Pnueli and Rosner (1990) consider a similar model in the context of distributed systems with linear-time specifications given by finite automata on infinite words. Here, the hierarchical organisation is represented by a pipeline architecture which allows each process to send signals only to the following one. The authors show that the distributed synthesis problem for such a system, is solvable via an automata-theoretic technique. This technique is further extended by Kupferman and Vardi (2001) to more general, branching-time specifications. The key operation of the construction is that of *widening* –



(a) hierarchical observation (b) static hierarchical information

Fig. 1 Basic patterns of hierarchical information: game positions show the observation of player 1 (left) and player 2 (right); the name of the position (middle) is unobservable

a finite-state interpretation of strategies for a less-informed player j within the strategies of a more-informed player $i \preceq j$. This allows to first solve a game as if all the moves were performed by the most-informed player, which comes first in the order $\preceq$, and successively discard solutions that cannot be implemented by the less-informed players, i.e., those which involve strategies that are not in the image of the widening interpretation.

The automata-theoretic method for solving the synthesis problem on pipeline architectures, due to Pnueli and Rosner (1990) and Kupferman and Vardi (2001), can be adapted directly to solve the synthesis problem for games with hierarchical observation. Alternatively, the solvability result follows from the reduction of games with hierarchical observations to pipeline architectures presented in Theorem 20, as a part of our discussion on games and architectures.

Theorem 2 (Pnueli and Rosner (1990); Kupferman and Vardi (2001)) *For games with hierarchical observation, the synthesis problem is finite-state solvable.*

3.2 Incorporating perfect recall

In a first step, we extend the notion of hierarchical information to incorporate the power of perfect recall that players have. While maintaining the requirement of a fixed order, we now ask that the *information set* of a player determines the information sets of those who follow in the order, as in the example of Figure 1(b).

Definition 3 A game graph yields *(static) hierarchical information* if there exists a total order $\preceq$ among the players such that, for all histories π, if $i \preceq j$, then $P^i(\pi) \subseteq P^j(\pi)$.

The following lemma provides an operational characterisation of the condition. We detail the proof, as its elements will be used later.

Lemma 4 *A game graph G yields static hierarchical information if, and only if, for every pair $i \preceq j$ of players, there exists a Moore machine that outputs $\beta^j(\pi)$ on input $\beta^i(\pi)$, for every history π in G.*

Proof For an arbitrary game graph G, let us denote the relation between the observations of two players i and j along the histories in G by

$$T^{ij} := \{(\beta^i(\pi), \beta^j(\pi)) \in (B^i \times B^j)^* \mid \pi \in \mathrm{Hist}(G)\}.$$

Let us first assume that there exists a Moore machine that recognises T^{ij}. Then T^{ij} is actually a function and hence $\pi \sim^i \pi'$ implies

$$\beta^j(\pi) = T^{ij}(\beta^i(\pi)) = T^{ij}(\beta^i(\pi')) = \beta^j(\pi'),$$

and we can conclude that $\pi \sim^j \pi'$.

To see that the converse implication holds, notice that T^{ij} is a regular relation, recognised by the game graph G viewed as a finite-word automaton A^{ij}_G over the alphabet of observation pairs $B^i \times B^j$. Concretely, consider the nondeterministic automaton $A^{ij}_G := (V, B^i \times B^j, v_0, \Delta, V)$ on states corresponding to positions of G, with initial state v_0 and with transitions $(v, (b^i, b^j), v') \in \Delta$ if there exists a move $(v, a, v') \in E$ such that $\beta^i(v') = b^i$ and $\beta^j(v') = b^j$; all states are accepting. Further, let M^{ij} be the automaton obtained by determinising A^{ij}_G and trimming the result, that is, removing all states that do not lead to an accepting state.

Now assume that the game graph G at the outset yields static hierarchical information. Then, the relation T^{ij} recognised by M^{ij} is functional, and hence M^{ij} is deterministic in the input component i: for any state v there exists precisely one outgoing transition along each observation $b^i \in B^i$. In other words, M^{ij} is a Mealy machine, which we can transform into an equivalent Moore machine, as desired. □

Theorem 5 *For games with static hierarchical information, the synthesis problem is finite-state solvable.*

Proof Intuitively, we transform an arbitrary game graph $G = (V, E, \beta)$ with static hierarchical *information* into one with hierarchical *observation*, by taking the synchronised product of G with automata that signal to each player i the observations of all players $j \succeq i$. We shall see that this preserves the solutions to the distributed synthesis problem, for any winning condition on G.

To make the construction precise, let us fix a pair $i \preceq j$ of players, and consider the Moore machine $M^{ij} = (M, m_0, \mu, \nu)$ as in the proof of Lemma 4, which translates the observations $\beta^i(\pi)$ into $\beta^j(\pi)$, for every history π in G. We define the product $G \times M^{ij}$ as a new game graph with the same sets of actions as G, and the same observation alphabets $(B^k)_{k \neq i}$, except for player i, for which we expand the alphabet to $B^i \times B^j$ to also include observations of player j. The new game is over positions in $V \times M$ with moves $((v, m), a, (v', m'))$ if $(v, a, v') \in E$ and $\mu(m, \beta^i(v)) = m'$. The observations for player i are given by $\beta^i(v, m) = (\beta^i(v), \nu(m))$, whereas they remain unchanged for all other players $\beta^k(v, m) = \beta^k(v)$, for all $k \neq i$.

The obtained product graph is equivalent to the original game graph G, in the sense that they have the same tree unravelling, and the additional components in the observations of player i (representing observations of player j, given by the

Moore machine M^{ij}) are already determined by his own observation history, so player i cannot distinguish any pair of histories in the new game that he could not distinguish in the original game. Accordingly, the strategies on the expanded game graph $G \times M^{ij}$ correspond to strategies on G, such that the outcomes of any distributed strategy are preserved. In particular, for any winning condition over G, a distributed strategy is winning in the original game if, and only if, it is winning in the expanded game $G \times M^{ij}$. On the other hand, the (positional) observations of Player i in the expanded game determine the observations of Player j.

By applying the transformation for each pair $i \preceq j$ of players successively, we obtain a game graph that yields hierarchical observation. Moreover, every winning condition on G induces a winning condition on (the first component of positions in) $G \times M^{ij}$ such that the two resulting games have the same winning strategies. Due to Theorem 2, we can thus conclude that, under ω-regular winning conditions, the synthesis problem is finite-state solvable for games with static hierarchical information. □

To decide whether a given game graph yields static hierarchical information, the collection of Moore machines constructed according to Lemma 4, for all players i, j, may be used as a witness. However, this yields an inefficient procedure, as the determinisation of a functional transducer involves an exponential blowup; precise bounds for such translations are given by Weber and Klemm (1995). More directly, one could verify that each of the transductions A_G^{ij} relating observation histories of Players i, j, as defined in the proof of Lemma 4, is functional. This can be done in polynomial time using, e.g., the procedure described in Béal et al. (2000).

We can give a precise bound in terms of nondeterministic complexity.

Lemma 6 *The problem of deciding whether a game yields static hierarchical information is* NLogSpace*-complete.*

Proof The complement problem — of verifying that for a given game there exists a pair of players i, j that cannot be ordered in either way — is solved by the following nondeterministic procedure: Guess a pair i, j of players, then check that $i \not\preceq j$, by following nondeterministically a pair of histories $\pi \sim^i \pi'$, such that $\pi \not\sim^j \pi'$; symmetrically, check that $j \not\preceq i$. The procedure requires only logarithmic space for maintaining pointers to four positions while keeping track of the histories. Accordingly, the complement problem is in NLogSpace, and since the complexity class is closed under complementation (Immerman (1988); Szelepcsényi (1988)), our decision problem of whether a game yields static hierarchical information also belongs to NLogSpace.

For hardness, we reduce the emptiness problem for nondeterministic finite automata, known to be NLogSpace-hard (Jones (1975)), to the problem of verifying that the following game for two players playing against Nature on the graph of the automaton yields hierarchical information: Nature chooses a run in the automaton, the players can only observe the input letters, unless an accepting state is reached; if this happens, Nature sends to each player privately one bit, which violates the condition of hierarchical information. Thus, the game has hierarchical information if, and only if, no input word is accepted. □

3.3 Signals and game transformations

Functions that return information about the current history, such as those constructed in the proof of Lemma 4 will be a useful tool in our exposition, especially when the information can be made observable to certain players without changing the game.

Given a game graph G, a *signal* is a function defined on the set of histories in G, or on the set of observation histories of some player i. We say that a signal $f : \mathrm{Hist}(G) \to \Sigma$ is information-consistent for player i if any two histories that are indistinguishable to player i have the same image under f; in particular, strategies are information-consistent signals. A finite-state signal is one implemented by a Moore machine. Any finite-state signal $f : \mathrm{Hist}(G) \to \Sigma$ can also be implemented by a Moore machine M^i over the observation alphabet B^i, such that that $M(\pi) = M^i(\beta^i(\pi))$ for every history π. The *synchronisation* of G with a finite-state signal f is the expanded game graph (G, f) obtained by taking the synchronised product $G \times M$, as described in the proof of Lemma 4. In case f is information-consistent for player i, it can be made *positionally observable* to this player, without changing the game essentially. Towards this, we consider the game graph (G, f^i) that expands (G, f) with an additional observation component $f^i(v)$ for player i at every position v, such that $f(\pi) = f^i(v)$ for each history π that ends at v. The game graph (G, f^i) is *finite-state equivalent* to G, in the sense that every strategy for G maps via finite-state transformations to a strategy for (G, f^i) with the same outcome and vice versa. Indeed, any strategy for G is readily a strategy with the same outcome for (G, f^i) and, conversely, every strategy profile s in (G, f^i) can be synchronised with the Moore machines implementing the signals f^i for each player i, to yield a finite-state strategy profile s' for G with the same outcome as s. In particular, the transformation preserves solutions to the finite-state synthesis problem under any winning condition.

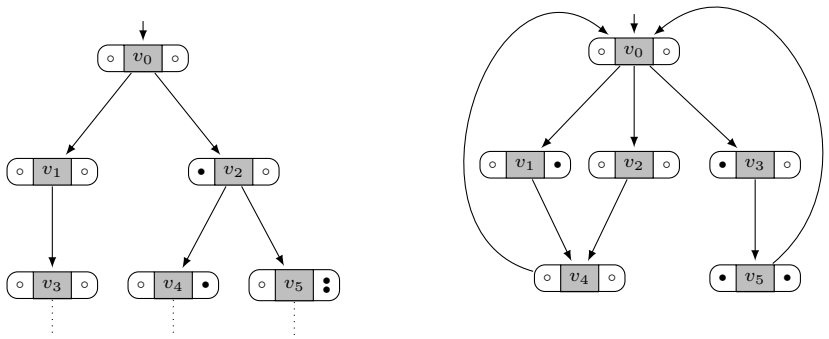
4 Dynamic Hierarchies

In this section, we maintain the requirement on the information sets of players to be totally ordered at every history. However, in contrast to the case of static hierarchical information, we allow the order to depend on the history and to change dynamically along a play. Figure 2(a) shows an example of such a situation: at the history reaching v_2, player 1 is more informed than player 2, however, the order switches when the play proceeds to position v_4, for instance.

Definition 7 A history π in a game yields *hierarchical information* if the information sets $\{P^i(\pi) \mid i \in N\}$ are totally ordered by inclusion. A game graph yields *dynamic* hierarchical information if every history yields hierarchical information.

We first observe that, for every finite game, the set of histories that yield hierarchical information is regular. We detail here the construction of an automaton for the complement language, which will also be of later use.

Lemma 8 *For every finite game graph G, we can construct a nondeterministic finite automaton that accepts the histories in G that do not yield hierarchical information. If G has n players and $|V|$ positions, the number of automaton states is at most $2n^2|V|^2$.*



(a) dynamic hierarchical information (b) recurring hierarchical information

Fig. 2 More patterns of hierarchical information

Proof Let us fix a game graph G. A history π in G fails to yield hierarchical information if there are two players with incomparable information sets at π. To verify this, we construct an automaton that chooses nondeterministically a pair i, j of players, then, while reading the input π, it guesses a pair π', π'' of histories such that $\pi' \sim^i \pi$ and $\pi'' \sim^j \pi$ and updates two flags indicating whether $\pi' \not\sim^i \pi''$ or $\pi' \not\sim^j \pi''$; the input is accepted if both flags are set. Hence, a word $\pi \in V^*$ that corresponds to a history in G is accepted if, and only if, the corresponding history does not yield hierarchical information. [2]

In its states, the constructed automaton stores the indices of the two players i, j, a pair of game positions to keep track of the witnessing histories π' and π'', and a two-bit flag to record whether the current input prefix is distinguishable from π' for player j or from π'' for player i. Clearly, it is sufficient to consider each pair of players only once, hence, the automaton needs at most $4\frac{n(n-1)}{2}|V|^2$ states, that is, less than $2n^2|V|^2$. □

To decide whether a given game graph G yields dynamic hierarchical information, we may check whether the automaton described in Lemma 8 accepts all histories in G. However, more efficient than constructing this automaton, we can use a nondeterministic procedure similar to the one of Lemma 6 to verify on-the-fly if there exists a history at which the information sets of two players are incomparable: guess two players i, j and three histories $\pi \sim^i \pi'$ and $\pi'' \sim^j \pi$, such that $\pi' \not\sim^i \pi''$ and $\pi' \not\sim^j \pi''$. Obviously, the lower bound from Lemma 8 is preserved.

Lemma 9 *The problem of deciding whether a game graph yields dynamic hierarchical information is* NLogSpace*-complete.*

In the remainder of the section, we show that, under the more liberal condition of dynamic hierarchical information, distributed games are still decidable.

[2] Notice that the automaton may also accept words that do not correspond to game histories; to avoid this, we can take the synchronised product with the game graph G and obtain an automaton that recognises precisely the set of histories that do not yield hierarchical information.

Theorem 10 *For games with dynamic hierarchical information, the synthesis problem is finite-state solvable.*

For the proof, we transform an arbitrary game $\mathcal{G}$ with dynamic hierarchical information into one with static hierarchical information, among a different set of n *shadow* players $1', \ldots, n'$, where each shadow player i' plays the role of the i-most informed player in the original game, in a sense that we will make precise soon. The information sets of the shadow players follow their nominal order, that is, if $i < j$ then $P^{i'}(\pi) \subseteq P^{j'}(\pi)$. The resulting shadow game inherits the graph structure of the orginal game, and we will ensure that, for every history π,

(i) each shadow player i' has the same information (set) as the i-most informed actual player, and
(ii) each shadow player i' has the same choice of actions as the i-most informed actual player.

This shall guarantee that the shadow game preserves the winning status of the original game.

The construction proceeds in two phases. Firstly, we expand the game graph G so that the correspondence between actual and shadow players does not depend on the history, but only on the current position. This is done by synchronising G with a finite-state machine that signals to each player his rank in the information hierarchy at the current history. Secondly, we modify the game graph, where the shadow-player correspondence is recorded as a positional attribute, such that the observation of each player is received by his shadow player, at every position; similarly, the actions of each player are transferred to his shadow player. Finally, we show how finite-state winning strategies for the shadow game can be re-distributed to the actual players to yield a winning profile of finite-state winning strategies for the original game.

4.1 Information rank signals

For the following, let us fix a game $\mathcal{G}$ with dynamic hierarchical information with the usual notation. For a history π, we write $\preceq_\pi$ for the total order among players induced by the inclusions between their information sets at π. To formalise the notion of an i-most informed player, we use the shortcut $i \approx_\pi j$ to denote that $i \preceq_\pi j$ and $j \preceq_\pi i$; likewise, we write $i \prec_\pi j$ to denote that $i \preceq_\pi j$ and not $j \preceq_\pi i$.

Then, the *information rank* of player i on the game graph G is a signal $\mathrm{rank}^i : \mathrm{Hist}(G) \to N$ defined by

$$\mathrm{rank}^i(\pi) := |\{j \in N \mid j \prec_\pi i \text{ or } (j < i \text{ and } j \approx_\pi i)\}|.$$

Likewise, we define the *order* of player i *relative* to player j as a Boolean signal $\preceq^i_j : \mathrm{Hist}(G) \to \{0,1\}$ with $\preceq^i_j(\pi) = 1$ if, and only if, $i \preceq_\pi j$.

Lemma 11 *The information rank of each player i and his order relative to any player j are finite-state signals that are information-consistent to player i.*

Proof We detail the argument for the rank, the case of relative order is similar and simpler.

Given a game $\mathcal{G}$ as in the statement, let us verify that the signal rank^i is information-consistent, for each player i. Towards this, consider two histories $\pi \sim^i \pi'$ in G, and suppose that some player j does not count for the rank of i at π, in the sense that either $i \prec_\pi j$ or ($i \approx_\pi j$ and $i < j$) — in both cases, it follows that $\pi \sim^j \pi'$, hence $P^j(\pi) = P^j(\pi')$, which implies that j does not count for the rank of i at π' either. Hence, the set of players that count for the rank of player i is the same at π and at π', which means that $\mathrm{rank}^i(\pi) = \mathrm{rank}^i(\pi')$.

To see that the signal rank^i can be implemented by a finite-state machine, we first build, for every pair i, j of players, a nondeterministic automaton A_i^j that accepts the histories π where $j \prec_\pi i$, by guessing a history $\pi' \sim^i \pi$ and verifying that $\pi' \not\sim^j \pi$. To accept the histories that satisfy $i \approx_\pi j$, we take the product of the automata A_i^j and A_j^i for $i \preceq_\pi j$ and $j \preceq_\pi i$ and accept if both accept. Combining the two constructions allows us to describe, for every player j, an automaton A_j to recognise the set of histories at which j counts for $\mathrm{rank}^i(\pi)$.

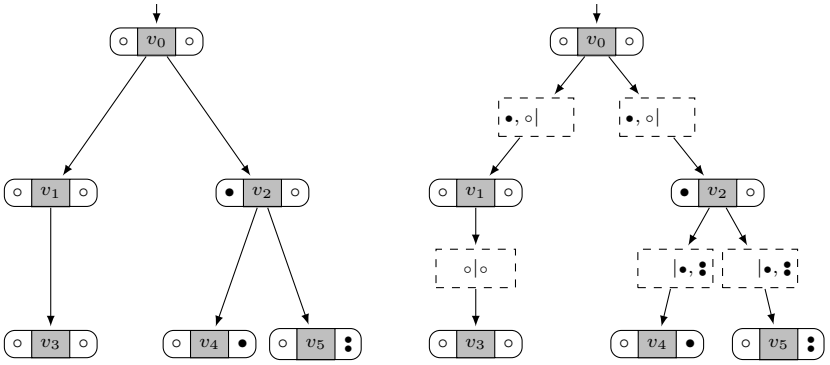
Next, we determinise each of the automata A_j and take appropriate Boolean combinations to obtain a Moore machine M^i with input alphabet V and output alphabet $\mathcal{P}(N)$, which upon reading a history π in G, outputs the set of players that count for $\mathrm{rank}^i(\pi)$. Finally we replace each set in the output of M^i by its size to obtain a Moore machine that returns on input $\pi \in V^*$, the rank of player i at the actual history π in G.

As we showed that rank^i is an information-consistent signal, we can conclude that there exists a Moore machine that inputs observation histories $\beta^i(\pi)$ of player i and outputs $\mathrm{rank}^i(\pi)$. □

One consequence of this construction is that we can view the signals rank^i and $\preceq_j^i$ as attributes of positions rather than properties of histories. Accordingly, we can assume without loss of generality that the observations of each player i have an extra rank component taking values in N and that the symbol j is observed at history π in this component if, and only if, $\mathrm{rank}^i(\pi) = j$. When referring to the positional attribute $\preceq_j^i$ at v, it is more convenient to write $i \preceq_v j$ rather than $\preceq_j^i$.

4.2 No crossing

As we suggested in the proof outline, each player i and his shadow player, identified by the observable signal rank^i, should be equally informed. To achieve this, we will let the observation of player i be received by his shadow, in every round of a play. However, since the rank of players, and hence the identity of the shadow, changes with the history, an information loss can occur when the information order between two players, say $1 \prec 2$ along a move is swapped to become $2 \prec 1$ in the next round. Intuitively, the observation received by player 2 after this move contains one piece of information that allows him to catch up with player 1, and another piece of information to overtake player 1. Due to their rank change along the move, the players would now also change shadows. Consequently, the shadow of 1 at the target position, who was previously as (little) informed as player 2, just receives the new observation of player 1, but he may miss the piece of information that allowed player 2 to catch up (and which player 1 had). Figure 3(a) pictures such a situation.



(a) crossing: $1 \prec 2$ at v_2 to $2 \prec 1$ at v_4 (b) half-step lookahead

Fig. 3 Eliminating crossings

We describe a transformation to eliminate the crossings due to switches in the information order, such that this artefact does no longer occur. Formally, for a play π in a game, we say that Player i and j cross at stage ℓ if $P^i(\pi_\ell) \subsetneq P^j(\pi_\ell)$ and $P^j(\pi_{\ell+1}) \subsetneq P^i(\pi_{\ell+1})$. We say that a game with dynamic hierarchical information is *cross-free* if there are no crossing players in any play.

Lemma 12 *Every game with dynamic hierarchical information is finite-state equivalent to a game that is cross-free.*

Proof Let G be a game graph with dynamic hierarchical information. We define a signal for each pair of players i, j that represents the knowledge that player j has about the current observation of player i. If this signal is made observable to Player i only at histories π at which $i \preceq_\pi j$, the game remains essentially unchanged, as players only receive information from less-informed players, which they could hence deduce from their observation. Concretely, we define the signal $\lambda^i_j : V^* \to \mathcal{P}(B^i)$ by

$$\lambda^i_j(\pi) := \{\beta^i(v') \ : \ v' \text{ is the last state of some history } \pi' \in P^j(\pi)\}.$$

Clearly, this is a finite-state signal.

Now we look at the synchronised product of G with the signals $(\lambda^i_j)_{i,j \in N}$ and the relative-order signal $\preceq^i_j$ constructed in the proof of Lemma 11. In the resulting game graph, the signal value $\lambda^i_j(\pi)$ at a history π that ends at a position w is represented by the position attribute $\lambda^i_j(w)$. We add to every move (v, a, w) an intermediary position u, at which we assign, for every player i the observation $\{\lambda^i_j(w) \ : \ i \preceq_w j\}$. Intuitively, this can be viewed as a half-step lookahed signal that player i receives from player j who may have been more informed at the source position v – thus the signal is not necessarily information-consistent for player i. Nevertheless, the game remains essentially unchanged after adding the signal, as the players cannot react to the received observation before reaching the target w, at which point the information is readily revealed. On the other hand, along moves at which the information order between players switches, the intermediary position

ensures that the players attain equal information. The construction is illustrated in Figure 3(b).

For any game $\mathcal{G}$ on G, we adjust the winning condition to obtain one for the new game graph, by ignoring the added intermediary positions. As the added positions have only one successor, the players have no relevant choice, so any distributed winning strategy for the new game corresponds to one for $\mathcal{G}$ and vice versa. In particular, for ω-regular winning conditions, the construction yields a game with no crossings that is finite-state equivalent to the original game. □

4.3 Shadow players

We are now ready to describe the construction of the shadow game associated to a game $\mathcal{G} = (V, E, \beta, W)$ with dynamic hierarchical information. Without loss of generality, we can assume that every position in G is marked with the attributes $\mathrm{rank}^i(v)$ and $\preceq^i_j$, for all players i, j according to Lemma 11 and that the game graph is cross-free, according to Lemma 12.

The shadow game $\mathcal{G}' = (V \cup \{\ominus\}, E', \beta', W)$ is also played by n players and has the same winning condition as $\mathcal{G}$. The action and the observation alphabet of each shadow player consists of the union of the action and observation alphabets of all actual players. The game graph G' has the same positions as G, plus one sink $\ominus$ that absorbs all moves along unused action profiles. The moves of G' are obtained from G by assigning the actions of each player i to his shadow player $j = \mathrm{rank}^i(v)$ as follows: for every move $(v, a, v') \in E$, there is a move $(v, x, v') \in E'$ labelled with the action profile x obtained by a permutation of a corresponding to the rank order, that is, $a^i = x^j$ for $j = \mathrm{rank}^i(v)$, for all players i. Finally, at every position $v \in V$, the observation of any player i in the original game $\mathcal{G}$ is assigned to his shadow player, that is $\beta'^j(v) := \beta^i(v)$, for $j = \mathrm{rank}^i(v)$.

By construction, the shadow game yields static hierarchical information, according to the nominal order of the players. We can verify, by induction on the length of histories, that for every history π, the information set of player i at π in G is the same as the one of his shadow player $\mathrm{rank}^i(\pi)$ in G'.

Finally, we show that the distributed synthesis problem for G reduces to the one on G', and vice versa. To see that $\mathcal{G}'$ admits a winning strategy if $\mathcal{G}$ does, let us fix a distributed strategy s for the actual players in $\mathcal{G}$. We define a signal $\sigma^j : \mathrm{Hist}(G') \to A$ for each player in $\mathcal{G}'$, by setting $\sigma^j(\pi) := s^i(\pi)$ if $j = \mathrm{rank}^i(\pi)$, for each history π. This signal is information-consistent for player j, since, at any history π, his information set is the same as for the actual player i with $\mathrm{rank}^i(\pi) = j$, and because the strategy of the actual player i is information-consistent for himself. Hence, σ^j is a strategy for player j in G'. Furthermore, at every history, the action taken by the shadow player $j = \mathrm{rank}^i(\pi)$ has the same outcome as if it was taken by the actual player i in G. Hence, the set of play outcomes of the profiles s and σ are the same and we can conclude that, if there exists a distributed winning strategy for G, then there also exists one for G'. Notice that this implication holds under any winning condition, without assuming ω-regularity.

For the converse implication, let us suppose that the shadow game $\mathcal{G}'$ admits a winning profile σ of finite-state strategies. We consider, for each actual player i of G,

the signal $s^i : \mathrm{Hist}(G) \to A^i$ that maps every history π to the action $s^i(\pi) := \sigma^j(\pi)$ of the shadow player $j = \mathrm{rank}^i(\pi)$. This is a finite-state signal, as we can implement it by synchronising G with rank^i, the observations of the shadow players, and the winning strategies σ^j, for all shadow players j. Moreover, s^i is information-consistent to the actual player i, because all histories $\pi \in P^i(\pi)$, have the same value $\mathrm{rank}^i(\pi) =: j$, and, since σ^j is information-consistent for player j, the actions prescribed by $\sigma^j(\pi)$ must be the same, for all $\pi \in P^j(\pi) = P^i(\pi)$. In conclusion, the signal s^i represents a finite-state strategy for player i. The profile s has the same set of play outcomes as σ, so s is indeed a distributed finite-state strategy, as desired.

In summary, we have shown that any game $\mathcal{G}$ with dynamic hierarchical information admits a winning strategy if, and only if, the associated shadow game with static hierarchical observation admits a finite-state winning strategy. The latter question is decidable according to Theorem 5. We showed that for every positive instance G', we can construct a finite-state distributed strategy for G. This concludes the proof of Theorem 10.

5 Transient Perturbations

As a third pattern of hierarchical information, we consider the case where incomparable information sets may occur at some histories along a play, but it is guaranteed that a total order will be re-established in a finite number of rounds.

Definition 13 A play yields *recurring hierarchical information* if it has infinitely many prefix histories that yield hierarchical information. A game yields *recurring hierarchical information* if all its plays do so.

Since the set of histories that yield hierarchical information is regular in any finite game, according to Lemma 8, it follows that the set of plays that yield recurring hierarchical information is ω-regular as well.

Lemma 14 *For every finite game, we can construct a deterministic Büchi automaton that recognises the set of plays that yield recurring hierarchical information. If G has n players and $|V|$ positions, the number of automaton states is bounded by $2^{O(n^2|V|^2)}$.*

Proof Given a game graph G, we can construct a nondeterministic finite automaton A that recognises the set of histories in G that do *not* yield hierarchical information, as in Lemma 8. By determinising the automaton A via the standard powerset construction, and then complementing the set of accepting states, we obtain a deterministic automaton $A^{\complement}$ with at most $2^{2n^2|V|^2}$ states that accepts a word $\pi \in V^*$ if either $\pi \notin \mathrm{Hist}(G)$, or π represents a history in G that yields hierarchical information. Finally, we take the synchronised product B of $A^{\complement}$ with the graph G. If we now view the resulting automaton as a Büchi automaton, which accepts an infinite words if infinitely many prefixes are accepted by B, we obtain a deterministic automaton that recognises the set of plays in G that yield hierarchical information. The number of states in B is at most $|V|\, 2^{2n^2|V|^2}$, hence bounded by $2^{O(n^2|V|^2)}$. □

The automaton construction provides an important insight about the number of consecutive rounds in which players may have incomparable information. Given a play π on a game graph G, we call a *gap* any interval $[t, t+\ell]$ of rounds such that the histories of π in in any round of $[t, t+\ell]$ do not yield hierarchical information; the length of the gap is $\ell + 1$. The game graph has *gap size* k if the length of all gaps in its plays is uniformly bounded by k.

Clearly, every game graph with finite gap size yields recurring hierarchical information. Conversely, the automaton construction of Lemma 14 implies, via a standard pumping argument, that the gap size of any game graph with recurring hierarchical information is at most the number of states in the constructed Büchi automaton. In conclusion, a game G yields recurring hierarchical information if and only if, the size of a gap in any play of G is bounded by $2^{O(n^2|V|^2)}$.

Corollary 15 *If a game yields recurring hierarchical information, then its gap size is bounded by* $2^{O(n^2|V|^2)}$*, where* n *is the number of players and* $|V|$ *is the number of positions.*

A family of game graphs where the gap size grows exponentially with the number of positions is illustrated in Figure 4. The example is adapted from Berwanger and Doyen (2008): There are two players with no relevant action choices and they can observe one bit, or a special symbol that identifies a unique sink position $v_\bullet$. The family is formed of graphs $(G_m)_{m\geq 1}$, each constructed of m disjoint cycles $(C^r)_{1\leq r\leq m}$ of lengths $p_1, p_2, \ldots, p_m$ corresponding to the first m prime numbers, respectively. We number the positions on the cycle corresponding to the r-th prime number as $C^r := \{c^r_0, \ldots, c^r_{p_r-1}\}$. On each cycle, both players receive the same observation 0. Additionally, there are two special positions v_{01} and v_{10}, that yield different observations to the players: $\beta^1(v_{01}) = \beta^2(v_{10}) = 0$ and $\beta^1(v_{01}) = \beta^2(v_{10}) = 1$. From the initial position v_0 of a game graph G_m, Nature can choose a cycle C^r with $r \leq m$. From each position $c^r_\ell \in C^r$, except for the last one with $\ell = p_r - 1$, there are is a moves to the subsequent cycle position $c^r_{\ell+1}$ and, additionally, to v_{01} and v_{10}. In contrast, the last cycle position $c^r_{p_r-1}$ has only the first position c^r_0 of the same cycle as a successor. From the off-cycle positions v_{01} and v_{10} the play proceeds to the unique sink state $v_\bullet$ that emits the special observation $\bullet$ to both players.

Now, we can verify, for each game graph G_m, that in every play π that proceeds only through cycle positions, the information sets of the two players are comparable at a prefix history of length $t > 2$ in π, if and only if, all the first m primes divide $t-2$; any play that leaves a cycle reaches the sink $v_\bullet$, where the information sets of both players coincide. Accordingly, G_m yields recurring hierarchical information. On the other hand, since the product of the first m primes is exponential in their sum, (for a more precise analysis, see Berwanger and Doyen (2008)), we can conclude that the gap size of the game graphs G_m grows exponentially with the number of positions.

As a further consequence of the automaton construction in Lemma 14, it follows that we can decide whether a game graph G yields recurring hierarchical information, by constructing the corresponding Büchi automaton and checking whether it accepts all plays in G. However, due to the exponential blow-up in the determinisation of the automaton, this straightforward approach would require exponential time (and space) in the size of the game graph and the number of

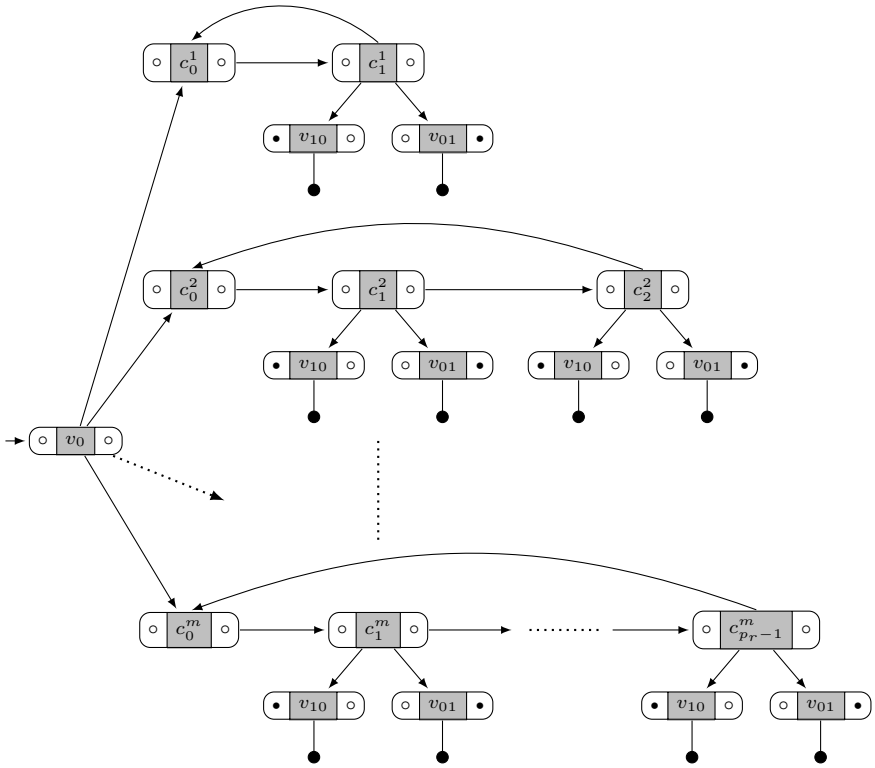


Fig. 4 Game with an exponential gap of non-hierarchical information (for better readability, the positions v_{01}, v_{10}, and $v_\bullet$ are multiply represented)

(pairs of) players. Here, we describe an on-the fly procedure that yields better complexity bounds.

Theorem 16 *The problem of deciding whether a game graph yields recurring hierarchical information is* PSPACE*-complete.*

Proof We describe a nondeterministic procedure for verifying that an input game G does *not* yield recurring hierarchical information, that is, there exists a play in G such that from some round t onwards, no prefix of length $\ell \geq t$ yields hierarchical information. By looking at the deterministic Büchi automaton B constructed in Lemma 14, we can tell that this is the case if, and only if, there exists a finally periodic play $\pi = \tau\rho^\omega \in V^\omega$ such that the run of B on π visits only non-accepting states after reading the prefix τ and, moreover, it returns to a previously visited state when, but not earlier than, reaching the prefix $\tau\rho$. In other words, the run on $\tau\rho$ induces a *lasso* in B, and hence, the length of $\tau\rho$ is bounded by the number of states $|V|\,2^{2n^2|V|^2}$ in the automaton.

The idea of the procedure, pictured in Algorithm 1, is to guess such a history $\tau\rho$, and to keep track of the states visited in the corresponding run of the automaton B, or more precisely, in the powerset construction of the automaton A from Lemma 8.

Let us first fix a pair of players i, j. Then, every state reachable by A upon reading the prefix π of an input word from V^ω is described by a tuple (u, c, w, d) consisting of a pair of game positions u, w and two binary flags c, d such that there exist histories π', π'' in G that end at u and w, satisfying $\pi \sim^i \pi'$, $\pi \sim^j \pi''$, and the flags c or d are set if, and only if, $\pi'' \not\sim^i \pi$ or $\pi \not\sim^j \pi'$, respectively. Essentially, this means that π', π'' are candidates for witnessing that players i, j have incomparable information at some continuation of π. Every state in the automaton B on the powerset of A records a set $Z_{ij} \in (V \times \{0,1\})^2$ of such tuples, each collecting the terminal positions of all witness candidates for i, j flagged correspondingly – we call such a set a *cell*. At the beginning, the procedure guesses one cell $\hat{Z}_{ij}$ for every pair of players i, j (Line 1); we call such a collection of cells a *configuration*. The guessed configuration stands for a state of B that shall be reached after reading τ and to which the run returns at $\tau\rho$. Then, starting from the initial configuration, where all players just see the initial position v_0 (Line 3), the procedure generates successively game positions of τ while updating the current configuration to simulate the run of B on τ. The current configuration Z summarises the possible runs of A on the prefix of τ generated so far, and it is updated for every new game position v according to Procedure Update. Once the configuration $Z = \hat{Z}$ is reached, the procedure enters a new loop. Here, it verifies in every iteration that the current configuration indeed contains a witness of incomparability for the input history, that is, there exists, in the cell of some pair of players, a state of A in which both distinguishability flags are set (Line 8). Provided the test succeeds, the procedure successively guesses the positions of ρ while updating the current configuration, until $\hat{Z}$ is reached again, in which case the procedure accepts. Apart of the case when the test in Line 8 fails, the procedure also rejects by looping.

Correctness and soundness The procedure mainly requires space to store the configurations $Z, \hat{Z}$ that collect a set of tuples from $(V \times \{0,1\})^2$ for each pair of players, hence it runs in polynomial space. To show correctness, we consider the sequence π of positions v generated in Lines 5 and 9, and argue that it forms a history in G and that, at every iteration of the loops in Lines 4 and 7, the configuration Z contains, in each cell Z_{ij}, precisely the set of pairs u, w of terminal positions reachable by candidate witnesses $\pi' \sim^i \pi$ such that $\pi'' \sim^j \pi$, with associated flags c, d indicating correctly whether $\pi' \sim^j \pi''$ and $\pi'' \sim^i \pi$, due to the way they are maintained in Lines 4 and 5 of Procedure Update. Accordingly, if the procedure accepts, then there exists a finally periodic play $\pi := \tau\rho^\omega$ in G such that the information sets of at least two players are incomparable, in every round from τ onwards. Soundness follows from the construction of the Büchi automaton in Lemma 14 and the observation that every run of the procdedure corresponds to a run of the automaton with the same acceptance status.

Hardness Finally, we argue that the problem of deciding whether a game graph yields recurring hierarchical information is PSPACE-hard by reduction from the universality problem for nondeterministic finite automata, shown by Meyer and Stockmeyer (1972) to be PSPACE-hard. Given a nondeterministic automaton $A = (Q, \Sigma, \Delta, q_0, F)$ over an alphabet Σ, we construct a game graph G for two players with no action choices and with observations in $B^1 = B^2 = \Sigma \times \{0,1\}$ corresponding to input letters for A tagged with one bit. The set of positions in G contains q_0 and all letter-state pairs $(a, q) \in \Sigma \times Q$. From every state (a, q) and from $q = q_0$, there

Algorithm 1: Deciding recurring hierarchical information

Data: game graph $G = (V, E, \beta)$ for n players
Result: accept if G does not yield recurring hierarchical information

type *cell*: subset of $(V \times \{0,1\})^2$
type *configuration*: matrix of *cells* $(Z_{i,j})_{1 \le i < j \le n}$
var $Z, \hat{Z}$: configurations
var $v, \hat{v}$: positions in V
var i, j: players in $\{1, \ldots, n\}$

1 **guess** $(\hat{v}, \hat{Z})$ // fix target on cycle
2 $v \leftarrow v_0$
3 **foreach** i, j *with* $i < j$ **do** $Z_{i,j} \leftarrow \{(v_0, 0, v_0, 0)\}$ // initial configuration
4 **while** $(v, Z) \neq (\hat{v}, \hat{Z})$ **do**
5 **guess** $v \in vEA$ // guess next history state
6 $Z \leftarrow \text{Update}(v, Z)$ // follow powerset construction
7 **repeat**
8 **if** $\bigwedge_{i,j} Z_{i,j} \cap (V \times \{1\})^2 = \emptyset$ **then reject** // hierarchical information
9 **guess** $v \in vEA$
10 $Z \leftarrow \text{Update}(v, Z)$
until $(v, Z) = (\hat{v}, \hat{Z})$ // cycle found
11 **accept**

is a move to position (a', q') if $(q, a', q') \in \Delta$. Each position (a, q) yields the same observation $(a, 0)$ to both players (the observation at q_0 is irrelevant). Thus, every run of A on a word $\alpha \in \Sigma^*$ corresponds to a unique history in G where both players observe the letters of α tagged with 0. In addition, G has two fresh positions v_a and v'_a, for every letter $a \in A$, with observations $\beta^1(v_a) = \beta^2(v'_a) = (a, 1)$ whereas $\beta^1(v'_a) = \beta^2(v_a) = (a, 0)$. Whenever there is a move in G from a position v to some position (a, q) with $q \in F$, we also allow a move from v to v_a. These fresh positions have one common successor, identified by a distinct observations to both players (essentially, indicating that the game is over). Clearly, G can be constructed from A in polynomial time.

Now, consider a nontrivial word $\alpha \in \Sigma^*$ and suppose that it admits an accepting run in A. At the corresponding history π in G, which yields the letters of α tagged with 1 as an observation to both players, the information sets are incomparable, because each player considers it possible that the other recieved the last letter with a 0-tag. In contrast, if α is rejected, the information sets at the corresponding history in G coincide, hence we have hierarchical information. In conclusion, the language of the automaton A is universal if, and only if, the constructed game graph yields herarchical information. □

We can show that the synthesis problem for the class of games with recurring hierarchical information is finite state-solvable, at least in the case when the winning conditions are observable. We conjecture that the result extends to the general case.

Theorem 17 *For games with recurring hierarchical information and observable ω-regular winning conditions, the synthesis problem is finite-state solvable.*

Proof The argument relies on the tracking construction decribed in Berwanger et al. (2011), which reduces the problem of solving distributed games with imper-

Procedure Update(v, Z)

Data: new position v, current configuration Z
Result: successor configuration after observing $\beta(v)$

```
1 foreach i, j do
2     foreach (u, c, w, d) ∈ Z_{i,j} do
3         foreach u' ∈ uEA, w' ∈ wEA with β^i(u') = β^i(v) and β^j(w') = β^j(v) do
4             c' ← c ∨ β^j(u') ≠ β^j(w')        // flag witness for j ⪯ i
5             d' ← d ∨ β^i(u') ≠ β^i(w')        // flag witness for i ⪯ j
6             Z'_{i,j} ← Z'_{i,j} ∪ {(u', c', w', d')}
return Z'
```

fect information for n players against Nature to that of solving a zero-sum game for two players with perfect information. The construction proceeds via an unravelling process that generates epistemic models of the player's information along the rounds of a play, and thus encapsulates their uncertainty.

This process described as "epistemic unfolding" in the paper (Berwanger et al. 2011, Section 3) is outlined as follows. An *epistemic model* for a game graph G with the usual notation, is a Kripke structure $\mathcal{K} = (K, (Q_v)_{v\in V}, (\sim^i)_{1\leq i\leq n})$ over a set K of histories of the same length in in G, equipped with predicates Q_v designating the histories that end in position $v \in V$ and with the players' indistinguishability relations $\sim^i$. The construction keeps track of how the knowledge of players about the actual history is updated during a round, by generating for each epistemic model $\mathcal{K}$ a set of new models, one for each assignment of an action profile a_k to each history $k \in K$ such that the action assigned to any player i is compatible with his information, i.e. for all $k, k' \in K$ with $k \sim^i k'$, we have $a^i_k = a^i_{k'}$. The update of a model $\mathcal{K}$ with such an action assignment $(a_k)_{k\in K}$ leads to a new, possibly disconnected epistemic model $\mathcal{K}'$ over the universe

$$K' = \{ka_kw \mid k \in K \cap Q_v \text{ and } (v, a_k, w) \in E\},$$

with predicates Q_w designating the histories $ka_kw \in K'$, and with $ka_kw \sim^i k'a_kw'$ whenever $k \sim^i k'$ in $\mathcal{K}$ and $w \sim^i w'$ in G. By taking the connected components of this updated model under the coarsening $\sim := \bigcup_{i=1}^n \sim^i$, we obtain the set of epistemic successor models of $\mathcal{K}$ in the unfolding. The tracking construction starts from the trivial model that consists only of the initial position of the game G. By successively applying the update, it unfolds a tree labelled with epistemic models, which corresponds to a game graph G' for two players with perfect information where the strategies of one player translate into distributed strategies in G and vice versa. According to (Berwanger et al. 2011, Theorem 5), for any winning condition, a strategy in $\mathcal{G}'$ is winning if and only if the corresponding joint strategy in $\mathcal{G}$ is so.

The construction can be exploited algorithmically if the perfect-information tracking of a game can be folded back into a finite game. A homomorphism from an epistemic model $\mathcal{K}$ to $\mathcal{K}'$ is a function $f : K \to K'$ that preserves the state predicates and the indistinguishability relations, that is, $Q_v(k) \Rightarrow Q_v(f(k))$ and $k \sim^i k' \Rightarrow f(k) \sim^i f(k')$. The main result of Berwanger et al. (2011) shows that, whenever two nodes of the unfolded tree carry homomorphically equivalent labels,

they can be identified without changing the (winning or losing) status of the game (Berwanger et al. 2011, Theorem 9). This holds for all imperfect-information games with ω-regular winning conditions that are observable. Consequently, the strategy synthesis problem is decidable for a class of such games, whenever the unravelling process of any game in the class is guaranteed to generate only finitely many epistemic models, up to homomorphic equivalence.

Game graphs with recurring hierarchical information satisfy this condition. Firstly, for a fixed game, there exist only finitely many epistemic models, up to homomorphic equivalence, where the $\sim^i$-relations are totally ordered by inclusion (Berwanger et al. 2011, Section 5). In other words, epistemic models of bounded size are sufficient to describe all histories with hierarchical information. Secondly, by Corollary 15, from any history with hierarchical information, the (finitely branching) tree of continuation histories with incomparable information is of bounded depth, hence only finitely many epistemic models can occur in the unravelling. Overall, this implies that every game with recurring hierarchical information and observable winning condition has a finite quotient under homomorphic equivalence. According to (Berwanger et al. 2011, Theorems 9 and 11), we can conclude that the distributed strategy problem for the class is finite-state solvable. □

We point out that the number of hierarchic epistemic models in games, and thus the complexity of our synthesis procedure, grows nonelementarily with the number of players. This should not come as a surprise, as the solution applies in particular to games with hierarchical observation under safety or reachability conditions (here, the distinction between observable and non-observable conditions is insubstantial), and it is known that already in this case no elementary solution exists (see Peterson and Reif (1979); Azhar et al. (2001)).

6 Games and Architectures

The game perspective focuses on the interaction between players letting their individual entity slide into the background. For instance, two players may interact locally without observing actions of a third, more remote player. Or, there may be independent teams specialised in solving particular tasks independently in response to inputs received from a central authority. Such organisational aspects can be crucial for coordinating distributed processes, yet they are hardly apparent from the representation of the system as a game graph.

6.1 Monitored architectures

To interpret our results with regard to processes and the infrastructure in which they interact, we wish to illustrate how the game model relates to the standard framework of distributed architectures of reactive systems introduced by Pnueli and Rosner (1990). Here, we formulate the setting in more general terms to allow for a meaningful translation of games into architectures, and for the presentation of new decidable architectures. In our framework, processes are equipped with a local transition structure according to which actions are enabled or disabled depending on previous actions and observations. Most importantly, the communication

structure is not hard-wired as in the classical setting, but instead implemented via a finite-state device that can dispatch information in different ways depending on the previous run.

A *process* is represented by an automaton $\mathcal{P} := (Q, A, B, q_0, \delta)$ on a set Q of states, with output alphabet A, input alphabet B, initial state $q_0 \in Q$, and a partial transition function $\delta : Q \times A \times B \to Q$. The symbols of the output alphabet A are called actions and those of the input alphabet B observations. We say that an action $a \in A$ is *enabled* at a state $q \in Q$, if $(q, a, b) \in \mathrm{dom}(\delta)$ for some $b \in B$. We assume that for every state $q \in Q$ the set $\mathrm{act}(q)$ of enabled actions is nonempty, and that for each enabled action $a \in \mathrm{act}(q)$ and each observation $b \in B$, the transition $\delta(q, a, b)$ is defined.

The automaton describes the possible behaviour of the process as follows: Every run starts in the initial state q_0. In any state q, the process chooses one of the available actions $a \in \mathrm{act}(q)$, then it receives an observation $b \in B$ and switches into state $\delta(q, a, b)$ to proceed. From a local perspective, the automaton expresses which actions of the process are enabled or disabled, depending on the sequence of previous actions and observations. From an external perspective, $\mathcal{P}$ defines a (local) *behaviour* relation $R_{\mathcal{P}} \subseteq (A \times B)^\omega$ consisting of the sequences of action-observation symbols in possible runs, which represents how the process may act in response to the observation sequence received as input.[3] In the following, we will not distinguish between the automaton $\mathcal{P}$ and the defined behaviour relation $R_{\mathcal{P}}$.

A *black-box* process is one with a single state, i.e., the set of enabled actions is independent of previous actions or observations. A *white-box* process is one where every state has precisely one enabled action – hence, a Moore machine, or a finite-state strategy in the game setting. A *program*, or strategy for process $\mathcal{P}$ is a white-box $\mathcal{S}$ with behaviour $\mathcal{S} \subseteq \mathcal{P}$.

A *monitored architecture* is represented by two elements: a collection of processes $\mathcal{P}^1, \ldots, \mathcal{P}^n$ plus a distinguished black-box process $\mathcal{P}^0$ called *Environment* on the one hand, and a specification of the communication infrastructure, called *view monitor*, on the other hand. When we refer to process i, we identify all associated elements with a superscript and write $\mathcal{P}^i = (Q^i, A^i, B^i, q_0^i, \delta^i)$. The set of *global actions* is the product $\Gamma := A^0 \times A^1 \times \cdots \times A^n$ of the action sets of all processes, including the Environment. Then, a *view monitor* is a Mealy machine $\mathcal{M} = (M, \Gamma, B, m_0, \mu, \nu)$ with input alphabet Γ and an output alphabet consisting of the product $B := B^1 \times \cdots \times B^n$ of the observation alphabets of all processes.

Hence, the monitor $\mathcal{M}$ transforms sequences of global actions into a tuple of observations, one for each process $i \in \{1, \ldots n\}$. The observations of the Environment are irrelevant, we always assume that $B^0 = \{0\}$. Since the Environement is a black box, it is completely specified by its set of actions, which already appears in the description of the view monitor. Therefore, $(\mathcal{P}^1, \ldots, \mathcal{P}^n, \mathcal{M})$ yields a complete description of a monitored architecture.

A *global behaviour* in a monitored architecture is an infinite sequence of global actions and observations $\pi := (a_0, b_0)(a_1, b_1), \cdots \in (\Gamma \times B)^\omega$ such that the observations corresponds to the output of the view monitor, $b_t = \mu(a_0, \ldots a_t)$ for all $t \geq 0$ and, for every process i, the corresponding action-observation sequence

[3] Such automata are called synchronous sequential transducers or nondeterministic generalised sequential machines in the literature.

$(a_0^i, b_0^i)(a_1^i, b_1^i)\ldots$ represents a behaviour in $\mathcal{P}^i$. We refer to the sequence of $a_0 a_1 \ldots$ of global actions in a global behaviour as a (global) *run*.

A *distributed program* is a collection $\mathcal{S} = (S^1, \ldots, S^n)$ of programs, one for every process. A global behaviour π is *generated* by a distributed program $\mathcal{S}$, if for every process i, the action-observation sequence $(a_0^i, b_0^i)(a_1^i, b_1^i)\ldots$ represents a behaviour in $\mathcal{S}^i \subseteq \mathcal{P}^i$.

The *run tree* of a distributed program $\mathcal{S}$ is the set $T_{\mathcal{S}} \subseteq \Gamma^*$ of prefixes of global runs generated by $\mathcal{S}$. Properties of runs are described by a linear-time or branching-time specification, given by an ω-word automaton or an ω-tree automaton over Γ, respectively. A run tree T satisfies a linear-time specification if all branches are accepted, and it satisfies a branching-time specification, if the tree T is accepted by the specification automaton. We say that a distributed program $\mathcal{S}$ is *correct* with respect to a specification if the generated run tree $T_{\mathcal{S}}$ satisfies the specification. Given an architecture together with a specification Φ, the distributed synthesis problem asks whether there exists a distributed program $\mathcal{S}$ that is correct with respect to the specification Φ.

6.2 From architectures to games and back

A monitored architecture $(\mathcal{P}^1, \ldots \mathcal{P}^n, \mathcal{M})$, can be translated into a distributed game $G = (V, E, \beta)$ for n players as follows. The set V of positions is the product $B \times M \times Q^1 \times \cdots \times Q^n$ of the global observation space with the state sets of the view monitor and of all processes (excluding the Environment), with initial position $(b_0, m_0, q_0^1, \ldots, q_0^n)$ for some (irrelevant) initial observation. There is a move $((b, m, q), a, (b', m', q')) \in E$, whenever the components are updated correctly, that is, the global observation $b' = \nu(m, a)$, the memory state of the view monitor $m' = \mu(m, a)$, and the local control state $q'^i := \delta(q^i, a^i, b'^i)$ for every process i. (Notice that the observation at the source does not matter). Finally, the observation function for each player i assigns $\beta^i(b, m, q) := b^i$.

Every prefix of a global run in the architecture induces a unique run prefix in the view monitor $\mathcal{M}$ and thus a history in the associated game, such that any program s^i for a process i corresponds to a strategy for player i and every specification induces a winning condition expressible by an automaton over the states of $\mathcal{M}$. Further, the outcome of a distributed finite-state strategy induces the run tree generated by the corresponding distributed program, hence every distributed programs that satisfies the specification correspond to winning strategy, and vice versa.

In this paper, we only considered games with linear-time winning conditions. Nevertheless, the automata technique of Kupferman and Vardi (2001) for solving games with hierarchical observation (Theorem 2) also applies to the branching-time setting. Thus, our decidability results for static and dynamic hierarchical information (Theorem 5 and 10) generalise immediately to branching-time winning conditions.

In summary, we can regard system architectures as a syntactic variant of games, modulo the transformation described above.

Proposition 18 *Every instance of the distributed synthesis problem over architectures can be reduced to an instance over games such that the finite-state solutions are*

preserved. For an architecture $(\mathcal{P}^1, \dots, \mathcal{P}^n, \mathcal{M})$ *and a specification given by a parity automaton* A_Φ*, the reduction runs in time* $O(\,|\mathcal{M}|\,(|A_\Phi| + |\mathcal{P}^1 \times \cdots \times \mathcal{P}^n|)\,)$.

For the converse, to translate a given game graph $G = (V, E, \beta)$ for n players into an architecture, we proceed as follows. For every player i, we first consider the localised variant $G^i = (V^i, E^i, \beta^i)$ of the game graph, obtained by replacing every action label a in a move $(u, a, v) \in E$ with the component a^i, and retaining only the observation function β^i. Then, we view the resulting one-player game graph as a nondeterministic finite-state automaton over the alphabet $A^i \times B^i$, by placing the observation symbol $b^i = \beta^i(v)$ from the target v of any move (u, a^i, v) on the incoming edge to yield the transition $(u, (a^i, b^i), v)$. Finally we determinise and minimise this automaton. The resulting automaton $(Q, A^i \times B^i, v_0, \delta)$ may not be complete in the second input component, as it is required for a process. Therefore, we add a fresh sink z with incoming transitions $(q, (a^i, b^i), z)$ from any state q where action a^i is enabled, but $\delta(q, (a^i, b^i))$ is undefined. The automaton $\mathcal{P}^i$ obtained in this way corresponds to a process.

To construct the associated view monitor, we expand the alphabet A of joint actions by adding a set A^0 of Environment actions, which we call *directions*. Intuitively, directions correspond to choices of Nature. Accordingly, we choose A^0 to be of size outdegree of G and expand the action profiles in the moves of E with directions, such that at any position, if there are two different outgoing moves, the direction in their action labels differs. Thus, we obtain a representation of G as a deterministic automaton with transition relation $\mu : V \times \Gamma \to V$ over the expanded alphabet $\Gamma := A^0 \times A^1 \times \cdots \times A^n$. Additionally, we define an output function $\nu : V \times \Gamma \to B$ that assigns to every position u and every direction-action profile $a \in \Gamma$, the profile of observations $\beta^i(v)$ for $v = \mu(u, a)$. Finally, we consider the Mealy automaton $\mathcal{M} = (V, \Gamma, B, v_0, \mu, \nu)$ as a view monitor for the collection of processes $\mathcal{P}^i$.

By way of this translation, the histories in G correspond to prefixes of runs in the architecture $(\mathcal{P}^1, \dots, \mathcal{P}^n, \mathcal{M})$, such that any finite-state strategy for a player i corresponds to a program for process i, and the outcome of any finite-state strategy profile s corresponds to the run tree generated by the distributed program corresponding to s. Since the game graph is deterministic in the action alphabet expanded with directions, every winning condition induces a specification, such that the solutions to the distributed synthesis problem are preserved. The specification is obtained as a product of the winning-condition automaton synchronised with the game graph (in the determinised variant with direction labels); in addition, any run that reaches a sink is deemed correct. Accordingly, the specification automaton is polynomial in the size of the game. However, as the process automata are constructed by determinisation and minimisation of the localised game graph, the overall translation involves a necessary exponential blowup (see van Glabbeek and Ploeger (2008)).

Proposition 19 *Every instance of the distributed synthesis problem over games can be reduced, in exponential time, to an instance over architectures such that the finite-state solutions are preserved.*

6.3 Pipelines, forks, and hierarchical observation

The classical setting of Pnueli and Rosner (1990) corresponds to the special case of architectures where all processes $\mathcal{P}^1, \ldots, \mathcal{P}^n$ are black boxes, and the action alphabet A^i of each process i consists of a product $X^i \times Y^i$ of sets of *control* and *communication* signals. The input alphabets and the communication infrastructure are described by a directed graph $((\{1, \ldots, n\}, \rightarrow)$ with arcs $i \rightarrow j$ expressing that, in every round, the communication signal y^i emitted with the action $a^i = (x^i, y^i)$ of process i is received by process j. Thus, each player i has an observation alphabet B^i formed by the product of the signal sets Y^j of all its predecessors $j \rightarrow i$ in the graph. In our framework, the communication graph corresponds to a view monitor $\mathcal{M}^i$ with only one state, which works as follows: upon input of a global action $((x^0, y^0), (x^1, y^1) \ldots, (x^n, y^n))$, output the global observation b composed of the collection of signals $b^i := (y^j |\ j \rightarrow i)$ for each player $i \in \{1, \ldots, n\}$.

A *pipeline* architecture in the Pnueli-Rosner setting is one where the communication graph is a directed path $0 \rightarrow 1 \rightarrow 2 \cdots \rightarrow n$. The main result of Pnueli and Rosner (1990), later extended by Kupferman and Vardi (2001), shows that the distributed synthesis problem is solvable for pipelines architectures. We prove that games with hierarchical observation and regular winning conditions can be translated into pipeline architectures with regular specifications such that the solutions to the distributed synthesis problem are preserved, thus justifying our statement in Theorem 2 as a corollary of the cited results.

Theorem 20 *The distributed synthesis problem for games with hierarchical observations reduces to the corresponding problem for pipeline architectures.*

Proof Consider a game graph $G = (V, E, \beta)$ that yields hierarchical observation with a winning condition $W \subseteq V^\omega$ given by an automaton. For simplicity, let us assume that the information hierarchy among the n players follows the nominal order $1 \leq \cdots \leq n$. Accordingly, the observation of each player $i > 1$ is determined by the observation of player $i - 1$, in the sense that there exists a function $f^i : B^{i-1} \rightarrow B^i$ such that $\beta^i(v) = f^i(\beta^{i-1}(v))$ for every position $v \in V$. By defining $f^1 : B^1 \rightarrow B^1$ to be the identity, we can hence write $\beta^i(v) = f^i \circ f^{i-1} \circ \cdots \circ f^1(\beta^1(v))$.

First, we transform the given game instance into a monitored architecture $(\mathcal{P}^1, \ldots, \mathcal{P}^n, \mathcal{M})$ and a specification Φ constructed as in Proposition 19. As the game yields hierarchical information, the output function of the view monitor $\mathcal{M}$ satisfies $\nu^i(m, a) = f^i \circ f^{i-1} \circ \cdots \circ f^1(\nu^1(m, a))$, for each player i, every memory state m, and global action a. Recall that $\mathcal{M}$ is deterministic in the first component of its input alphabet, which corresponds to Environment actions (directions). Hence, we can write $\nu^1(m, a)$ as $\nu^1(m, a^0)$.

Next, we sequentialise the obtained architecture: For each process $i < n$, we expand the action alphabet A^i with the observation alphabet B^{i+1} of the next player. Thus, we set $\hat{A}^i := A^i \times B^{i+1}$ and consider the actions in A^i as internal signals and the observations B^{i+1} as communication signals. The actions of the last process carry no relevant communication symbols, we set $\hat{A}^n = A^n \times \{0\}$. Now, we modify each process $i < n$ to output in addition to his action a^i (now an internal signal), the observation value $f^{i+1}(b^i)$ as a communication symbol – where b^i refers to the observation received by process i in the previous period (or the default initial observation); to implement this, the value $f^{i+1}(b^i)$ is stored in the state of

the process when receiving b^i and then added to all outgoing transitions. Let us call the modified processes $\hat{\mathcal{P}}^i$. Again, the last process remains unchanged.

Notice that process $\hat{\mathcal{P}}^1$ relies on observations $\nu^1(m, a^0)$ rather than just environment action a^0. To fix this, we take its synchronised product with $\mathcal{M}$. The resulting process $\mathcal{M}\times\hat{\mathcal{P}}^1$ inputs only environment actions and updates the monitor state internally, to yield the same output $\nu^1(m, a^0)$ as $\hat{\mathcal{P}}^1$.

As a view monitor $\hat{\mathcal{M}}$, we take the standard monitor for the communication graph $0 \to 1 \to 2 \to \cdots \to n$, in the sense of Pnueli and Rosner (1990), as described in the beginning of the subsection.

To account for the latency introduced by sequentialisation, we adjust the specification Φ over the original set of global actions $\Gamma = A^0 \times A^1 \times \ldots A^n$. For every infinite word $\alpha = a_0 a_1 \cdots \in \Gamma^\omega$ formed of global actions $a_t = (a_t^0 a_t^1 \ldots a_t^n)$ for all $t \geq 0$, we denote by $\mathrm{pipe}(\alpha) := a'_0, a'_2, \cdots \in \Gamma^\omega$ the infinite sequence of global actions where $a'_t = (a_t^0, a_{t+1}^1, \ldots, a_{t+n}^n)$, for all $t > 0$. We can verify that for every global run α in $(\mathcal{P}, \mathcal{M})$, the sequence $\mathrm{pipe}(\alpha)$ represents a run in the pipeline $(\hat{\mathcal{P}}, \hat{\mathcal{M}})$. Finally, we define the specification $\hat{\Phi} := \{\mathrm{pipe}(\alpha) \in \Gamma^\omega \mid \alpha \in W\}$.

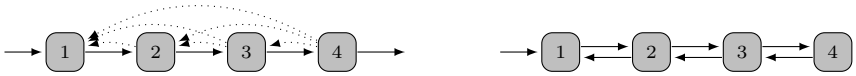
For the monitored architecture $(\mathcal{M}\times\hat{\mathcal{P}}^1, \hat{\mathcal{P}}^2, \ldots, \hat{\mathcal{P}}^n, \hat{\mathcal{M}})$ and the specification $\hat{\Phi}$, every correct distributed program corresponds to a finite-state distributed winning strategy in the original game, and vice versa.

To conclude the reduction, we restrict the specification $\hat{\Phi}$ to the set of global runs generated by the processes of this architecture, and replace the processes with black boxes, thus obtaining a hard-wired pipeline architecture in the framework of Pnueli and Rosner (1990), which preserves the solutions of the distributed synthesis problem for the game G with hierarchical observation from the outset. □

Interestingly, the direct translation of pipeline architectures into games does not necessarily result in games with hierarchical observation. Consider, for instance a pipeline with three processes $1 \leq 2 \leq 3$, each with a one-bit communication alphabet. In the corresponding game, while player receives an input bit from the environment, player 2 can play an action that reveals a bit to process 3 which is not revealed to player 1, thus the information sets of player 1 and player 3 become incomparable. Indeed, the architecture model does not a priori prevent processes to emit signals that are independent of the received input. The condition is ensured only in the implemented system, where every process follows its prescribed program. For arbitrary architectures (already in the Pnueli-Rosner framework), it seems hard to incorporate this condition into the process of designing a correct distributed program.

For the case of pipelines, it is easy to work around this circularity. The idea is to send the input of every player to all the previous players, as illustrated in Figure 5(a). Formally, this amounts to transforming a given architecture $\mathcal{A}$ on the communication graph $(\{1, \ldots, n\}, \to)$ by adding *feedback links* $i \to j$ from any process i to all processes $j < i$. Clearly, the resulting architecture $\mathcal{A}'$ corresponds to a game with hierarchical observation. We argue that the distributed synthesis problem is invariant under this transformation

Lemma 21 *Every pipeline can be reduced, by adding feedback links, to an architecture that corresponds to a game with hierarchical observation and admits the same solutions to the distributed synthesis problem.*



(a) pipeline with added feedback links (b) two-way chain

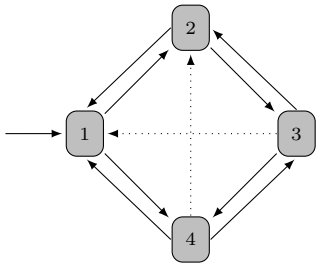
Fig. 5 Pipelines and chains (original links solid, added feedback links dotted)

Proof Any distributed program for a pipeline architecture $\mathcal{A}$ generates the same run in the architecture $\mathcal{A}'$ with added feedback links (which are ignored), hence every solution for $\mathcal{A}$ is also a correct distributed program for $\mathcal{A}'$. Conversely, given a distributed program $\mathcal{S}'$ for an architecture $\mathcal{A}'$ with feedback links added to a pipeline architecture $\mathcal{A}$ as above, we can construct a distributed program $\mathcal{S}$ for $\mathcal{A}$ by considering, for every process $i < n$, the synchronised product of $\mathcal{S}'^i$ with $\mathcal{S}'^{i+1} \times \cdots \times \mathcal{S}'^n$; hence, the current observation of each player $j \geq i$ is maintained in the control state. The program $\mathcal{S}^i$ is built from the product automaton, by using the observation data from the state rather than the signals from the incoming feedback links. Accordingly, $\mathcal{S}$ is a distributed program for the pipeline $\mathcal{A}$ that generates the same runs as $\mathcal{S}'$, hence the transformation preserves correctness under any specification. □

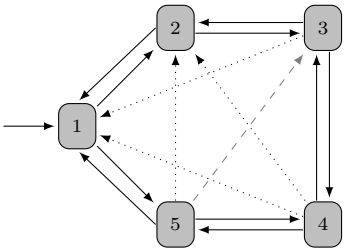
In terms of games, the argument of Lemma 21 can be rephrased as follows: in the game corresponding to a pipeline architecture, the game graph induced by any finite-state strategy profile yields hierarchical information. Then, we view the output of a hypothetical finite-state strategy (program) of every player i as a finite-state signal. Due to the pipeline structure, this signal is information-consistent for each receiving player $j \leq i$, hence it can be made observable (across the feedback links). Thus, we obtain a game with hierarchical observation that is equivalent to the one that corresponds to the pipeline at the outset.

Using this idea, we can recover further results on decidable architectures presented by Kupferman and Vardi (2001) and Finkbeiner and Schewe (2005). Two-way chains, for instance, that is, pipelines where every process $i > 1$ has an additional link to process $i - 1$ as pictured in Figure 5(b), lead to the same game as the underlying pipeline under the transformation of Lemma 21. The case of rings with up to four processes is similar: a ring is a two-way pipeline with an additional two-way link between the first and the last process. In a four-process ring as pictured in Figure 6(a), for any distributed program, process 1 can infer the signals emitted by 3 (for 2 and 4), and process 2 the signals emitted by 4. Hence, we can add feedback links from 3 to 1 and from 4 to 2 without changing the set of runs generated by distributed programs. By doing so, we obtain a game with hierarchical observation in the order $1 \preceq 2 \approx 4 \preceq 3$.

However, two-way rings with five or more black-box processes can in general not be transformed into games with hierarchical observation by adding feedback links. For any linear ordering $\leq$ of the processes, the addition of feedback links from each process i to all $j \leq i$ leads to an architecture that allows spurious runs, which cannot be generated by distributed programs in the original architecture. Figure 6(a) illustrates that hierarchical observation cannot be attained with the ordering $1 \leq 2 \leq 3 \leq 4 \leq 5$, for instance, as the feedback link from 5 to 3 (dashed in the picture) represents a signal that is not observation-consistent for process 3. Indeed, as pointed out by Mohalik and Walukiewicz (2003) and Finkbeiner and



(a) four processes attain hierarchical observation



(b) link from 5 to 3 (and from 4 to 2) is not observation-consistent

Fig. 6 Two-way rings with information-consistent feedback links (dotted)

Schewe (2005), the synthesis problem for two-way rings with at least five black-box processes is undecidable.

In general, achitectures in the classical framework of Pnueli and Rosner, where the communication links are hard-wired, present the following dichotomy: either there exists a total ordering $\preceq$ among processes, such that the addition of feedback links $\{i \to j \mid j < i\}$ leaves the set of runs generated by any distributed program unchanged, or the architecture contains an information fork, in the sense defined by Finkbeiner and Schewe (2005) – in the latter case, the authors of the cited paper show that there always exists a specification under which the distributed synthesis problem is undecidable.

7 Effective synthesis for monitored architectures

By relaxing the condition of hierarchical information to include changing or intermittent hierarchies, we obtained more general classes of games on which the distributed synthesis problem is effectively solvable. How does this generalisation carry over to distributed architectures? In the standard framework of architectures with hard-wired communication links, there is little hope for redrawing the decidability frontier for distributed synthesis by exploiting patterns of dynamic or recurring hierarchical information: According to the information-fork criterion of Finkbeiner and Schewe (2005), any solvability condition on architectures must either restrict the specifications, or request that the communication graph is essentially a pipeline. Since hierarchical information — in all considered variants — is a property of the game graph, independent of the winning condition, it would be unnatural to cast it as a restriction on the system specification. On the other hand, solvability of pipeline architectures is already covered by the basic condition of hierarchical observation.

Nevertheless, we encounter situations in practice where the components of a distributed system coordinate successfully without being restricted to communicate along the links of a fixed pipeline, or even a fixed communication graph. Consider, for instance, a system that operates in time phases, each with a specific workflow among component processes that are organised in a pipeline, but just for the duration of one phase; in the next phase, the workflow may change and follow another pipeline. If, at the end of each phase, all players are updated with the

information received in the global system, we obtain a situation corresponding to a game with dynamic hierarchical information. Even if the architecture allows any two processes to communicate in any direction, the described phase design ensures that dynamic hierarchical information is maintained, hence the synthesis problem is solvable. Our shadow-player construction can be understood as an instance of this idea. Beyond maintaining reconfigurable pipelines, one may furthermore allow workflows that propagate incomparable information, as long as this occurs for a bounded number of rounds.

7.1 Hierarchical architectures

One way to ensure that the synthesis problem is solvable on a class of instances is by restricting the architecture so that the corresponding game graph yields (static, dynamic, or recurring) hierarchical information. To verify whether a monitored architecture satisfies the condition, we can use the procedures from Lemma 9 and Theorem 16 directly, without constructing the corresponding game. Given an architecture $(\mathcal{P}^1, \ldots, \mathcal{P}^n, \mathcal{M})$ the procedure for dynamic (or static) hierarchical information runs in space $O(\sum_{i=1}^{n} \log(|\mathcal{P}^i|) + \log(|\mathcal{M}|))$, and for the case of recurring hierarchical information, it runs in space $O((n|\times_{i=1}^{n} \mathcal{P}^i \times \mathcal{M}|)^2)$.

In the fixed-architecture framework, solvable classes are characterised by a basic pattern of the communication graph: a pipeline where no process i is allowed to receive signals from any process $j < i-1$. This ensures that every run maintains hierarchical observation, hence the game graph corresponding to the architecture yields hierarchical observation, which implies that the synthesis problem is solvable. We can identify similar solvability patterns for monitored architectures by restricting to view monitors that never deliver signals which would violate the condition of dynamic hierarchical information. For any architecture equipped with such a monitor, the synthesis problem is solvable with respect to every specification. Deciding whether a given view monitor $\mathcal{M}$ yields dynamic or recurring hierarchical information (in the corresponding game graph) with every matching collection of processes amounts to deciding whether the architecture formed of black-box processes and the view monitor $\mathcal{M}$ yields dynamic or recurring hierarchical information, and can hence be done in logarithmic or polynomial space, respectively.

Thus, we obtain classes of monitored architectures on which the distributed synthesis problem is solvable as a direct application of our game-theoretic analysis.

7.2 Maintaining hierarchical information through strategies

Our second proposal for automated synthesis in the framework of monitored architectures relies on enforcing hierarchical information strategically, at the program level, rather than restricting the architecture a-priori. In this way, the task of avoiding incomparable information is put in the hands of the designer of the distributed system. We describe an automated method to help the system designer accomplish this task.

Let us first detail our argument in terms of games. Note that, in view of recurring hierarchical information, it is undecidable whether, for a given game, there

exists a strategy that is winning and also avoids infinite gaps with incomparable information: this follows by adapting the standard reduction from the Halting Problem to the synthesis of reachability strategies in two-player games with imperfect information (see, e.g., Peterson and Reif (1979), Berwanger and Kaiser (2010)). Therefore, we restrict our attention to dynamic hierarchical information.

Definition 22 Given a game, a strategy s maintains hierarchical information if every history that follows s yields hierarchical information.

One straighforward, but important insight is that the synthesis problem restricted to strategies that maintain hierarchical information is effectively solvable.

Theorem 23 *For any finite game, it is decidable whether there exists a distributed winning strategy that maintains hierarchical information, and if so, we can synthesise one.*

Proof Let $\mathcal{G}$ be an arbitrary finite game with an ω-regular winning condition. According to Lemma 8, the set of game histories that yield hierarchical information is regular. Let A be a deterministic automaton that recognises this set and consider its synchronised product $G \times A$ with the game graph G. From this product, we construct a new game graph G' by adding a sink position $\ominus$ with a fresh observation to be received by all players, and by replacing all moves $((v, q), a, (v, q'))$ in $G \times A$ where the automaton state q' at the target is non-accepting with $((v, q), a, \ominus)$. Hence, all histories in G' yield hierarchical information and every history in G that does not yield hierarchical information, maps to one in G' that ends at $\ominus$. Finally, we adjust the winning condition of $\mathcal{G}$ by expanding each play with the corresponding run of A and by excluding all plays that reach $\ominus$.

The game $\mathcal{G}'$ constructed in this way yields dynamic hierarchical information, hence the synthesis problem is effectively solvable. Moreover, every distributed winning strategy in $\mathcal{G}'$ corresponds to a distributed strategy in $\mathcal{G}$ that is winning and maintains hierarchical information in the sense of Definition 22, and conversely, every winning strategy that maintains hierarchical information in $\mathcal{G}$ corresponds to a winning strategy in $\mathcal{G}'$. □

The above theorem gives raise to an effective, sound, and incomplete method for solving the synthesis problem for monitored architectures: Given a problem instance consisting of an architecture and a specification, solve the synthesis problem for the corresponding game restricted to strategies that maintain hierarchical information, and translate any resulting finite-state winning strategy back into a distributed program. This approach omits solutions that involve runs which do not yield hierarchical information. However, if there exist correct distributed programs that also maintain hierarchical information, the procedure will construct one.

Alternatively, the method based on Theorem 23 can be viewed as a complete procedure for solving the synthesis problem on instances with arbitrary architectures, but with specifications restricted to run trees in which every run (corresponds to a play that) yields dynamic hierarchical information – a regular property according to Lemma 8.

7.3 Hierarchical routing

To conclude, we present a concrete example of an architecture design that supports the application of our method without otherwise restricting the solution procedure.

We set out with the observation that the condition of dynamic (or static) hierarchical information is a safety condition that can be supervised by the view monitor. Our proposal is to incorporate into view monitors the ability to send hierarchy-related data to the processes, which can be used by the programs to ensure that hierarchical information is maintained whenever possible.

To illustrate the idea, we consider architectures of a particular format, which we call *routed* architectures. Intuitively each processes can emit signals addressed to any other process, and the view monitor, called *router*, either delivers a signal or denies it, according to a deterministic rule. In case of denial, the sending process receives a notification. The intention is to maintain hierarchical information on every run as far as possible. However, signals sent by the Environment cannot be denied, and inter-process signals may also be forced for delivery by the emitting process. This may lead to violations of the condition of hierarchical information, in which case the monitor sends a panic signal to all processes and henceforth simply delivers all emitted signals.

Formally, a routed architecture for n processes and the Environment features action of a special format. Each action has a control and a communication component: the control component is a single symbol as in the case of static architectures; the communication component is a (possibly empty) list of signals, each formed of a body – a single symbol – and a header, which contains the identities i, j of the sending and the receiving process, and a priority number. In Environment actions, all signals carry top priority (which forces delivery). We assume that no process appears twice as a receiver in one action. The observations of each process are formed of a communication component, which consists of a list of signals, at most one from every other process, and a notification component, which consists of a panic flag and a list of delivery flags, one for every other process.

Priority numbers are used to determine the observations delivered in response to a global action. Towards this, the priorities of all signals in the communication component of an observation are aggregated. The aggregation function is monotonous with respect to the inclusion between sets of signals, and each set of observations has a unique element of maximum value (to break ties, we may use process identifiers). Since the observation space is finite, it is always possible to define a priority aggregation function with these properties.

The *router* for a given collection of processes is a view monitor that operates as follows. There is one sink state called panic state: here the router reads the communication components of the global action and delivers to each process i the list of all signals addressed to i as a receiver, also setting the panic flag and delivery flags for each signal emmited by process i. In any other states, including the initial one, the router reads the global action a and considers the set of admissible observations: a global observation b is admissible if (1) the communication component consists of signals emitted in the global action a and contains all signals with top priority, and (2) the delivery flags are set correctly, and the panic flag is reset, if and only if, the run prefix that would be reached by delivering the observation b (corresponds to a play that) yields hierarchical information. If there exist admissible observations that do not raise the panic flag, the monitor picks

the one of maximal aggregated priority, delivers it to the processes, and switches into a non-panic successor state. Else, if all admissible observations raise the panic flag, the monitor delivers all received signals and switches into the panic state. As the condition of hierarchical observation corresponds to a finite-state signal, the described operation of the router can be implemented by a finite-state monitor. Essentially, the states store the data needed to supervise the information hierarchy ordering.

In terms of expressiveness, routed architectures lie between hard-wired and monitored architectures. As in hard-wired architectures, signals are routed unaltered between processes, the difference being that contents and receiver of a signal are chosen by the emitting process. In contrast, the signals delivered by an arbitrary view monitor to different processes can be highly correlated depending on on the global action and the state of the monitor. For routed architectures, the correlation is restricted to non-delivery of signals due to violating the condition of hierarchical information.

Given a routed architecture and a (linear or branching-time) specification, we can use the incomplete synthesis method presented in Subsection 8 to synthesise a distributed program that maintains hierarchical information. If this succeeds, we obtain a correct distributed program for which no panic or non-delivery flag will ever be raised.

However, assuming that the given specification is insensitive to signals that are not delivered, it is sufficient to synthesise a distributed strategy that avoids the panic signal, but allows non-delivery of signals. In game-theoretic terms, this corresponds to viewing signalling attempts as cheap-talk actions. The option of sending signals that may be denied does not enlarge the class of solvable problem instances: in principle, each process i can infer from its observation sequence that a certain signal will not be delivered, since this can occur only when the receiving process is less informed than i – in other words, the non-delivery notification is information-consistent for process i, so it yields no new information. Nevertheless, in practice, the warning mechanism against driving the system into a non-hierarchical state offers the system designer more freedom in programming the processes than the setting where such attempts are forbidden.

Finally, for problem instances that do not admit a complete hierarchical solution, our model of routed architectures allows to combine strategies synthesised via the method from Subsection 8 applied on the prefix of the run tree on which hierarchical information is maintained with other strategies that are triggerred when the panic signal is raised.

8 Discussion

The task of coordinating several players with imperfect information towards attaining a common objective is quintessential in the design of computational systems with multiple interacting components. Known automated methods for accomplishing this task rely on a basic pattern of hierarchical information flow. Here, we showed that this pattern can be relaxed considerably, towards allowing the hierarchy to rearrange in the course of the interaction, or even to vanish temporarily.

Our technical analysis is based on finite-state games with perfect recall and synchronous dynamics. The model has the advantage of exposing the fundamental

connection between knowledge and action as put forward by Moses (2015). Indeed, our results on solvable cases for the coordination problem in games rely on the key concept of identifying finite-state signals that provide sufficient knowledge for triggering winning strategies.

The game model has shortcomings as well. The assumption of perfect recall is rather uncritical, as we finally synthesise strategies implementable by finite-state automata. However, the assumption of synchronous dynamics does restrict the scope of our results: the uncertainty about the ordering of event occurrences in asynchronous systems cannot be captured directly by imperfect information in games, and different methods are required to approach the synthesis problem. (See, e.g., Madhusudan and Thiagarajan (2002), Gastin et al. (2009), Muscholl and Walukiewicz (2014).) Insights on information and coordination in games may shed light on the asynchronous setting as well, however, at the current state of research the areas appear divided.

Perhaps the greatest challenge is to make concrete how the game-theoretic results can help in designing real-world systems. In terms of operational models, our games are close to the distributed reactive systems of Pnueli and Rosner (1990), described by a communication architecture and a regular specification: Instances of the synthesis problem can be translated back and forth between the game and the system model. Nevertheless, it turns out that games with (static, dynamic, or recurring) hierarchial information, do not correspond to natural classes of distributed architectures. This is because most of the interaction structure described by a game graph is incorporated into the specification on the side of the architecture. While the game graph expresses which actions *will be* available or not in a certain state, the specification expresses which actions *should* occur or not in a run of a correct program. Since the original model of distributed reactive systems is too permissive in describing possible behaviours of processes, a meaningful classification in terms of information flow patterns would need to refer to behaviours of correct programs – which we aim yet to construct.

To overcome the cleavage between game and architecture representations, we introduced the model of monitored architectures where the hard-wired communication graph of the Pnueli-Rosner model is replaced by a transducer that transforms global actions into signals sent to the processes, in a dynamic way, depending on the previous run. This yields a faithful representation of the information flow in the operational model, which allows to translate the decidability results on games with hierarchical information. Thus, we obtain rich classes of distributed architectures on which the automated synthesis problem is solvable.

In continuation of our study, we see three directions for further research. One promising approach is to refine the classification of solvable classes according to the structure of the winning condition (or the specification) rather than considering only the set of possible behaviour represented by the game graph (or the architecture). In the classical framework of Pnueli and Rosner, Madhusudan and Thiagarajan (2001) considered specifications that are *local*, in the sense that they require each process to satisfy a condition expressed on the state space of its own automaton. The authors show that, under such specification, the class of solvable architectures includes clean pipelines, that is pipelines where the last process receives private signals from the environment. Technically, the games corresponding to clean pipelines do not yield hierarchical information, nevertheless, the synthesis problem can be solved by a straightforward adaptation of the automata-theoretic

method of Pnueli and Rosner (1990); Kupferman and Vardi (2001). As another example, our method for synthesising distributed programs that maintain hierarchical information, presented in Subsection , can be interpreted as a complete solution for arbitrary architectures with specifications restricted to runs that yield dynamic hierarchical information.

A second concrete direction relies on assessing the condition of hierarchical information at runtime rather than considering the raw system model. In terms of games, this corresponds to asserting that the game graph induced by every strategy yields hierarchical information, although this may not be true for the original game. Such an approach would allow to capture functional dependencies in runs generated by distributed programs, and lead to even more liberal classes of solvable architectures.

Finally, we may include the view monitor in the synthesis process. So far, we modeled the view monitor as a white-box process that represent the available communication infrastructure. However, in practical applications, the infrastructure does not need to be fixed, the system designers may be able to configure certain of its parameters. On the other hand, as we showed in Subsection 7.3, it can be helpful to use the abilities of the view monitor as a global observer to provide the processes with signals about the global run, even if they are observation-consistent and hence could be deduced locally by the receiver. Therefore, it will seems appropriate to consider view monitors as grey-box processes, and synthesise white-box implementations using automated procedures.

At bottomline, we are confident that the quest for effective automated synthesis is worth pursuing. To correct the discouraging picture drawn by the fundamental undecidability results, it is important to identify natural classes of models on which the problem is solvable. We believe that hierarchical information offers a convincing example in this direction.

References

Azhar, Salman, Gary Peterson, and John Reif. 2001. Lower bounds for multiplayer noncooperative games of incomplete information. *Journal of Computers and Mathematics with Applications* 41: 957–992.

Béal, Marie-Pierre, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. 2000. Squaring transducers: An efficient procedure for deciding functionality and sequentiality of transducers. In *Proc. of latin american symposium on theoretical informatics (latin 2000)*, eds. Gaston H. Gonnet and Alfredo Viola. Vol. 1776 of *Lecture notes in computer science*, 397–406. Springer. ISBN 978-3-540-67306-4.

Berwanger, Dietmar, and Laurent Doyen. 2008. On the power of imperfect information. In *Proceedings of the 28th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'08)*, eds. Ramesh Hariharan, Madhavan Mukund, and V. Vinay. Vol. 2 of *Leibniz international proceedings in informatics*. Bangalore, India: Leibniz-Zentrum für Informatik.

Berwanger, Dietmar, and Łukasz Kaiser. 2010. Information tracking in games on graphs. *Journal of Logic, Language and Information* 19 (4): 395–412.

Berwanger, Dietmar, Łukasz Kaiser, and Bernd Puchala. 2011. Perfect-information construction for coordination in games. In *Proc. of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*. Vol. 13 of *Lipics*, 387–398. Leibniz-Zentrum für Informatik.

Büchi, J. Richard, and Lawrence H. Landweber. 1969. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society* 138: 295–311.

Finkbeiner, B., and S. Schewe. 2005. Uniform distributed synthesis. In *Logic in Computer Science (LICS'05), proc*, 321–330. IEEE.

Gastin, Paul, Nathalie Sznajder, and Marc Zeitoun. 2009. Distributed synthesis for well-connected architectures. *Formal Methods in System Design* 34 (3): 215–237.

Grädel, E., W. Thomas, and T. Wilke, eds. 2002. *Automata, Logics, and Infinite Games. Lecture notes in computer science*. Springer.

Halpern, Joseph Y., and Moshe Y. Vardi. 1989. The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences* 38 (1): 195–237.

Immerman, Neil. 1988. Nondeterministic space is closed under complementation. *SIAM J. Comput.* 17 (5): 935–938.

Janin, David. 2007. On the (high) undecidability of distributed synthesis problems. In *Proc. of theory and practice of computer science (sofsem 2007)*. Vol. 4362 of *Lecture notes in computer science*, 320–329. Springer.

Jones, Neil D. 1975. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences* 11 (1): 68–85.

Kupferman, Orna, and Moshe Y. Vardi. 2001. Synthesizing distributed systems. In *Proc. of logic in computer science (lics'01)*, 389–398. IEEE Computer Society Press.

Madhusudan, P., and P. S. Thiagarajan. 2001. Distributed controller synthesis for local specifications. In *Proceedings of the 28th international colloquium on automata, languages and programming, icalp '01*. Vol. 2076 of *Lncs*, 396–407. Springer.

Madhusudan, P., and P. S. Thiagarajan. 2002. A decidable class of asynchronous distributed controllers. In *Concur 2002 concurrency theory*, eds. Lubo Brim, Mojmr Ketnsk, Antonn Kuera, and Petr Janar. Vol. 2421 of *Lecture notes in computer science*, 145–160. Springer. ISBN 978-3-540-44043-7.

Meyer, Albert R., and Larry J. Stockmeyer. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. annual symposium on switching and automata theory (swat'72)*, 125–129. IEEE Computer Society.

Mohalik, Swarup, and Igor Walukiewicz. 2003. Distributed games. In *Foundations of software technology and theoretical computer science (fsttcs 2003), proc.*, 338–351. Springer.

Moses, Yoram. 2015. Relating knowledge and coordinated action: The knowledge of preconditions principle. In *Theoretical aspects of rationality and knowledge, TARK 2015, proc.* Vol. 215 of *EPTCS*, 231–245.

Muscholl, Anca, and Igor Walukiewicz. 2014. Distributed synthesis for acyclic architectures. In *Foundation of software technology and theoretical computer science, FSTTCS 2014, proc.* Vol. 29 of *Lipics*, 639–651. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Peterson, Gary L., and John H. Reif. 1979. Multiple-Person Alternation. In *Proc 20th annual symposium on foundations of computer science, (focs 1979)*, 348–363. IEEE.

Pnueli, Amir, and Roni Rosner. 1990. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st annual symposium on foundations of computer science, focs '90*, 746–757. IEEE Computer Society Press.

Puchala, Bernd. 2013. Synthesis of winning strategies for interaction under partial information. PhD diss, RWTH Aachen University.

Rabin, Michael O. 1972. *Automata on infinite objects and Church's thesis. Regional conference series in mathematics*. American Mathematical Society.

Schewe, Sven. 2008. Synthesis of distributed systems. PhD diss, Universität des Saarlandes.

Szelepcsényi, Róbert. 1988. The method of forced enumeration for nondeterministic automata. *Acta Inf.* 26 (3): 279–284.

Thomas, Wolfgang. 1995. On the synthesis of strategies in infinite games. In *Proc. of symposium on theoretical aspects of computer science (stacs 1995)*, 1–13.

van Glabbeek, Rob, and Bas Ploeger. 2008. Five determinisation algorithms, eds. Oscar H. Ibarra and Bala Ravikumar, 161–170. Berlin, Heidelberg: Springer. 978-3-540-70844-5.

Weber, Andreas, and Reinhard Klemm. 1995. Economy of description for single-valued transducers. *Inf. Comput.* 118 (2): 327–340.