


Distributed Games with a Central Decision Maker

Bharat Adsul 

Indian Institute of Technology Bombay, India

Nehul Jain 

Indian Institute of Technology Bombay, India

Abstract

We study distributed games played on non-deterministic asynchronous automata which feature a central decision maker process that participates in all key decision making tasks. In these partial-information games, processes use their causal past to respond to scheduling choices made by the scheduler and cooperatively strategize as a team to achieve the winning objective. We show that the problem of deciding the existence of a distributed winning strategy is efficiently solvable for global safety and local parity objectives. We provide algorithmic solutions that match their computational hardness. We formulate the notion of a finite-state distributed strategy which allows to quantify its distributed memory requirements. For the aforementioned objectives, we establish that finite-state distributed winning strategies always exist. In fact, we provide novel constructions of such winning strategies which are shown to have almost optimal amount of distributed memory. We also show that a natural extension of the model with two decision making processes is undecidable.

2012 ACM Subject Classification Theory of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases Mazurkiewicz traces, models of concurrency, distributed synthesis, game-theoretic models, asynchronous automata, distributed decision-making

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.5

1 Introduction

The design and analysis of concurrent programs and distributed protocols have always presented significant challenges. With the increasing pervasiveness of distributed applications, it is natural to investigate the automated synthesis of such protocols from their specifications. The foundational work on distributed synthesis [19] demonstrated the general undecidability of the problem when processes have access to only purely local information and limited communication capabilities. However, recent works [9, 14, 10, 18, 8, 11, 1] have introduced richer models where processes have access to their entire *causal past*, broadening the scope of problems that can be tackled. In particular, [9] introduced the notion of causal past and addressed the distributed controller synthesis problem in series-parallel systems and established that controlled reachability is decidable. The work [14] introduced systems with connectedly communicating processes and proved that the MSO theory in this setting is decidable which implies that the associated distributed synthesis problems are decidable.

Two more recent and relevant lines of work are Petri games and asynchronous games. Petri games [8] are distributed games played on Petri nets, where tokens represent players. The places in the underlying Petri net are divided into system and environment places, and tokens in system places make decisions based on their causal history. Several interesting classes of Petri games have been shown to be decidable [8, 7] but it was recently shown in [6] that Petri games with global winning conditions are undecidable, even with only two system players and one environment player.

Asynchronous games [10] are played on deterministic asynchronous automata and allow processes to control actions based on their causal past. These games are shown to be decidable in the interesting setting of acyclic architectures [10, 18] whose underlying process-



© Bharat Adsul and Nehul Jain;

licensed under Creative Commons License CC-BY 4.0

45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025).

Editors: C. Aiswarya, Ruta Mehta, and Subhajit Roy; Article No. 5; pp. 5:1–5:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

communication graph is acyclic. More general formulations like decomposable games [11] have also been identified and proven to be decidable. However, the undecidability results for asynchronous games with simple objectives such as local reachability/termination/deadlock-freeness, even in systems with six processes, from [12] highlights the complexity of distributed synthesis problems.

Clearly it is desirable to identify different settings of the distributed synthesis problems which are interesting, practically well motivated and decidable. In this work, we consider the distributed setting in which there is a designated process called the *central decision maker*. We refer to such distributed systems as CDM systems in which all the key decisions are taken in the presence of this process. In many server-client architectures the server process is primarily responsible for maintaining overall integrity and ensuring that the demands of the client processes are met. These systems serve as prototypical examples of CDM systems. A distributed version control system such as SVN may also be viewed as a CDM system in which a single authoritative main copy is maintained at the central repository. Users make concurrent changes to local copies; in order to effect these changes to the main copy, they acquire an exclusive access to the central repository using a “lock”. One can model this locking mechanism by a central decision maker process which synchronizes with different user processes. Another illustrative example of a CDM system is the working of an institution/organization consisting of multiple agents with a designated *head*. Various committees are formed to address different aspects of the organization, expedite work and increase overall efficiency. An agent is typically a member of several such committees. These committees conduct procedural meetings from time to time to discuss, gather relevant information and propose solutions. However, all the key decisions are taken in only those committee meetings in which the head participates. The head’s presence in all decision making activities ensures consistency and alignment with the organizational goals. Note that concurrent meetings of different committees are allowed in this example.

In this paper, we formulate and study a distributed CDM game which is played on a *non-deterministic* asynchronous automaton A involving a team \mathcal{P} of processes and assume the presence of a central decision maker, a special process in \mathcal{P} which participates in *every* action which is *not* deterministic. In these games, the environment manifests itself as a *scheduler* of actions. Each scheduling decision gives rise to an *event* which is distributed across the participating processes. These scheduling *events* inherit a natural partial order in which, for every process, the events in which this process participates, are totally ordered. The processes participating in a scheduled event must respond to the corresponding scheduling choice, purely based on their *collective causal past*, by providing a *matching local transition* of A . It is important to keep in mind that the environment is simply an *abstract* scheduler of (possibly concurrent) actions/events and it *does not reveal* any information about concurrent scheduling choices. In particular, the causal past of the central decision maker contains information only about some scheduling events and gets updated with possibly *unbounded* information about concurrent scheduling events as the play proceeds. Further, the causal past of any other process may *not* contain complete information about the decisions taken by the central decision maker. It has access to only those decisions which it learns directly or indirectly through other processes and is oblivious to concurrent scheduling decisions made by the central decision maker.

We show that the key problem of solving a CDM game, that is, deciding the existence of a distributed winning strategy, is efficiently solvable for global safety and local parity winning objectives. We also provide hardness results for these problems which demonstrate that our algorithmic solutions are optimal. More precisely, we show that the problems of solving

CDM games for the above objectives are EXPTIME-complete. Our EXPTIME-completeness result for global safety CDM games has some resemblance with a similar result for the Petri game model with one system token studied in [7]. It is important to note that [7] is only concerned with safety objectives. In contrast, we provide a uniform method to address both safety as well as parity objectives. Towards this, we build upon the theory of Mazurkiewicz traces and the theory of two-player games. More precisely, given a CDM game, we associate with it a natural two-player full-information game. Our key result allows to “extract” from a winning strategy in this two-player game, a distributed winning strategy in the CDM game. This is achieved by crucially relying on novel *special* linearizations of traces that we develop in this work. We believe that these special linearizations have much wider applicability.

This work also formulates a natural notion of a *finite-state distributed memory strategy*. Such a strategy is modelled as a deterministic asynchronous automaton in which each process keeps track of some additional key information in its *local memory states* and the responses to scheduling choices are computed with the help of the joint-memory states. The amount of additional local memory provides a measure of the *distributed memory complexity* of such a finite-state strategy. Recall that, in the standard full-information two-player graph games central to reactive synthesis (see [13]), safety and parity objectives can be achieved by positional/zero-memory winning strategies. In contrast, non-trivial *distributed* memory is required, in general, for winning global safety and local parity objectives in the CDM setting. For the safety and local parity objectives on CDM games, we establish that existence of distributed winning strategies implies existence of finite-state distributed winning strategies. In fact, we provide a novel construction of a finite-state winning strategy in which each process keeps in its local memory the latest global-state that is part of its causal past. We use the gossip automaton from [17] to maintain and update this information. It turns out that the distributed memory complexity of our construction is exponential in $|\mathcal{P}|$ - the number of processes. We also show that this exponential dependence on $|\mathcal{P}|$ is unavoidable.

Let us mention that our analysis applies to the more general setting of distributed games in which the “decision” events are causally ordered. In this general setting, any pair of actions, which are *not deterministic*, must have *some* process participating in both of them. We thus rule out concurrent/independent actions which are not deterministic. Clearly, this setting includes CDM games as the central decision maker participates in *all* actions which are not deterministic. Throughout the paper, we mainly address the CDM setting for convenience. Our final result concerns an extension of the CDM model which permits two decision making processes. In this natural extension, in every action which is not deterministic, at least one of the two decision makers participates. We build on the technical machinery of [12] to show the surprising result that safety games with two decision makers are undecidable.

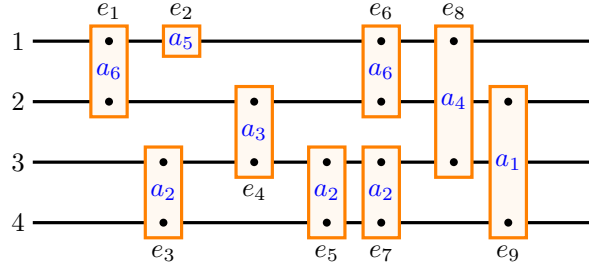
2 Preliminaries

In this section, we set up basic notations about (Mazurkiewicz) traces (see [4, 3, 16]). A trace is a well-established model of a concurrent behaviour in which the causality and concurrency information between events is represented in the form of a labelled partial order.

Let \mathcal{P} be a finite non-empty set of processes, and let i, j range over \mathcal{P} . We write $\{X_i\}$ for a \mathcal{P} -indexed family $\{X_i\}_{i \in \mathcal{P}}$. A *distributed alphabet* over \mathcal{P} is a family $\tilde{\Sigma} = \{\Sigma_i\}$ of finite sets. Let $\Sigma = \bigcup_{i \in \mathcal{P}} \Sigma_i$ be the total alphabet, i.e., the set of all actions. For $a \in \Sigma$, we set $\text{loc}(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$ to be the set of processes which participate in it. We let $I = \{(a, b) \in \Sigma \times \Sigma \mid \text{loc}(a) \cap \text{loc}(b) = \emptyset\}$ and $D = (\Sigma \times \Sigma) \setminus I$ be the induced *independence* and *dependence* binary relations on Σ respectively. For a poset (X, \leq) and $Y \subseteq X$, we define $\downarrow Y = \{x \in X \mid x \leq y \text{ for some } y \in Y\}$; for $x \in X$, we set $\downarrow x = \downarrow \{x\}$ and $\Downarrow x = \downarrow x \setminus \{x\}$.

- **Definition 1.** A trace over $\tilde{\Sigma}$ is a Σ -labelled poset $t = (E, \leq, \lambda)$ where,
- E is a (possibly infinite) set of events and $\lambda : E \rightarrow \Sigma$ is a labelling function.
 - \leq is a partial order on E such that
 1. for each $e, e' \in E$, $(\lambda(e), \lambda(e')) \in D$ implies $e \leq e'$ or $e' \leq e$.
 2. for each $e, e' \in E$, $e \leq e'$ implies $(\lambda(e), \lambda(e')) \in D$, where $e \leq e'$ if $e < e'$ and for each e'' with $e \leq e'' \leq e'$, either $e'' = e$ or $e'' = e'$.
 3. for each $e \in E$, $\downarrow e$ is finite.

Let $t = (E, \leq, \lambda)$ be a trace over $\tilde{\Sigma}$. The elements of E are referred to as *events* in t . For $e \in E$, $\text{loc}(e)$ abbreviates $\text{loc}(\lambda(e))$. For $i \in \mathcal{P}$, the set of i -events in t is $E_i = \{e \in E \mid i \in \text{loc}(e)\}$, i.e., the events in which process i participates. Note that the first condition in the definition of a trace implies that E_i is totally ordered by \leq . We write $\text{loc}(t) = \{i \in \mathcal{P} \mid E_i \neq \emptyset\}$ to denote the set of processes which participate in *some* event in t . Note that $\text{loc}(t) = \cup_{e \in E} \text{loc}(e)$.



■ **Figure 1** A process line represents the sequence of actions in which the corresponding process participates. Each event is represented by a rectangle with its tied action inside.

► **Example 2.** A trace with 9 events over 4 processes with $\mathcal{P} = \{1, 2, 3, 4\}$ is shown in Figure 1 over a distributed alphabet $\tilde{\Sigma} = \{\Sigma_i\}_{i \in \mathcal{P}}$ where, $\Sigma_1 = \{a_4, a_5, a_6\}$ and $\Sigma_2 = \{a_1, a_3, a_6\}$ and so on. Each process is indicated by a horizontal line and time flows rightward. Each event is represented by a vertical box and is labelled by a letter in $\Sigma = \{a_1, a_2, a_3, a_4, a_5, a_6\}$. Some events such as e_2 are purely local to a single process. Dots in synchronizing events indicate the participating processes. It is easy to infer causality relation (\leq) from the figure, for example, $e_1 < e_4 < e_5 < e_7$. We can also infer which events are concurrent (unrelated by \leq): for example, event e_6 is concurrent to events e_5, e_7 ; e_8 and e_9 are also concurrent. The pairs of events e_4, e_5 and e_1, e_4 are causally ordered by process 3 and process 2, respectively.

Let $\text{TR}(\tilde{\Sigma})$ denote the set of all traces over $\tilde{\Sigma}$. As the distribution of Σ across processes will be clear from the context, we use $\text{TR}(\Sigma)$ (or simply TR) instead of $\text{TR}(\tilde{\Sigma})$. Henceforth a trace means a trace over $\tilde{\Sigma}$. A trace is finite if the underlying set of events is finite. We use $\text{TR}^*(\Sigma)$ and $\text{TR}^\omega(\Sigma)$ (or simply TR^* and TR^ω) to denote the set of all finite and infinite traces. Now we describe the concatenation operation on traces. Let $t = (E, \leq, \lambda)$ and $t' = (E', \leq', \lambda')$ be *finite* traces with disjoint event sets E and E' . We define $tt' \in \text{TR}^*$ to be the trace (E'', \leq'', λ'') where

- $E'' = E \cup E'$,
- \leq'' is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda'(e')) \in D\}$,
- $\lambda'' : E'' \rightarrow \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

Observe that, with a (resp. b) denoting the singleton trace whose only event is labelled a (resp. b), if $(a, b) \in I$ then $ab = ba$ in TR^* . The trace concatenation operation on finite traces can also be defined for infinite traces under certain conditions. We explain this now.

The above mentioned definition of tt' results in a valid trace if t is finite. That is, if $t \in \text{TR}^*$ and $t' \in \text{TR}$ then $tt' \in \text{TR}$. Moreover, if $t, t' \in \text{TR}$ are such that $\omega\text{loc}(t) \cap \text{loc}(t') = \emptyset$, then $tt' \in \text{TR}$ where $\omega\text{loc}(t)$ is the set of processes which participate in infinitely many events in t . It is easy to see that, in this case, $\omega\text{loc}(tt') = \omega\text{loc}(t) \cup \omega\text{loc}(t')$. We say that a trace t' is a *prefix* of a trace t if $t = t't''$ for some trace t'' . It is important to observe that this definition permits a prefix to be infinite.

We now come to the very important notion of a configuration of a trace $t = (E, \leq, \lambda)$. A subset $c \subseteq E$ is a *configuration* of t if c is *finite* and $\downarrow c = c$. For $e \in E$, $\downarrow e = \downarrow\{e\}$ is the causal past of e and $\Downarrow e = \downarrow e \setminus \{e\}$ is the strict causal past of e . Note that, if we restrict the trace t to a configuration c , we get another *finite* trace $t_c = (c, \leq, \lambda)$ which turns out to be a *prefix* of t . Conversely, given a finite prefix t' of t , we can identify a configuration c of t such that $t' = t_c$. In this sense, configurations of t are essentially finite prefixes of t . Examples of configurations (or, equivalently, finite prefixes) are the empty set, $\downarrow e$ or $\Downarrow e$, for every event $e \in E$. Note that E itself is a configuration of t iff t is finite. We let C_t denote the set of all configurations of t . The *action based successor relation* $\rightarrow_t \subseteq C_t \times \Sigma \times C_t$ is defined by $c \xrightarrow{a}_t c'$ if and only if there exists $e \in E$ such that $e \notin c$, $c \cup \{e\} = c'$ and $\lambda(e) = a$.

► **Example 3.** For the trace t in Figure 1, $c = \{e_1, e_3, e_4\}$ and $c' = \{e_1, e_2, e_3, e_4\}$ are configurations, and so is $c'' = \downarrow e_5 = \{e_1, e_3, e_4, e_5\}$. Observe that $c \xrightarrow{a_5}_t c'$ and $c \xrightarrow{a_2}_t c''$. The set $\{e_1, e_2, e_4\}$ is not a configuration since $\downarrow\{e_1, e_2, e_4\} = \{e_1, e_2, e_3, e_4\}$.

Asynchronous automata and the related transition systems are fundamental finite-state distributed devices due to Zielonka [20, 16] which operate/run on traces. In an asynchronous transition system, each process $i \in \mathcal{P}$ is equipped with a finite non-empty set S_i of *local* i -states. We set $S = \bigcup_{i \in \mathcal{P}} S_i$ and call S the set of local states. Further, we set $S_{\mathcal{P}} = \prod_{i \in \mathcal{P}} S_i$ and call it the set of *global* states. For a non-empty set $P \subseteq \mathcal{P}$ of processes, $S_P = \prod_{i \in P} S_i$ is the set of (joint) P -states. For a P -state $s \in S_P$ and $i \in P$, $s(i) \in S_i$ denotes the local i -state in the P -tuple s . More generally, for $Q \subseteq P$ and $s \in S_P$, we denote by s_Q the projection of s on Q – that is, s_Q is the unique Q -state such that $s_Q(i) = s(i)$ for all $i \in Q$. For $a \in \Sigma$, we use a to abbreviate $\text{loc}(a)$ when talking about states. Thus $S_a = S_{\text{loc}(a)}$ denotes the set of all a -states and if $\text{loc}(a) \subseteq P$ and s is a P -state, we write s_a for $s_{\text{loc}(a)}$.

Similar to the convention of writing a \mathcal{P} -indexed family as $\{X_i\}$, we use the shorthand $\{Y_a\}$ to denote the Σ -indexed family $\{Y_a\}_{a \in \Sigma}$.

► **Definition 4.** An *asynchronous transition system (ATS)* over $\tilde{\Sigma}$ is $A = (\{S_i\}, \{\xrightarrow{a}\})$ where,
 ■ For each process $i \in \mathcal{P}$, S_i is a finite non-empty set of local i -states.
 ■ For each action $a \in \Sigma$, $\xrightarrow{a} \subseteq S_a \times S_a$ is a non-deterministic transition relation on a -states.

Let $A = (\{S_i\}, \{\xrightarrow{a}\})$ be an ATS. Note that a transition in A on an action a is *local* in the sense that it involves only processes in $\text{loc}(a)$ and a -states. We extend these local transition relations naturally to global states. More precisely, for $a \in \Sigma$, we define $\xRightarrow{a} \subseteq S_{\mathcal{P}} \times S_{\mathcal{P}}$ as follows: for $s, s' \in S_{\mathcal{P}}$, $(s, s') \in \xRightarrow{a}$ if $(s_a, s'_a) \in \xrightarrow{a}$ and $s_{\mathcal{P} \setminus \text{loc}(a)} = s'_{\mathcal{P} \setminus \text{loc}(a)}$. An action a is said to be *enabled* at $s \in S_{\mathcal{P}}$ if there exists $s' \in S_{\mathcal{P}}$ such that $(s, s') \in \xRightarrow{a}$. Due to the *locality of transitions*, if a is enabled at s , then it is also enabled at s' with $s_a = s'_a$.

Now we define the important notion of a run of A on a trace t over $\tilde{\Sigma}$. Let us fix an initial global state $s_0 \in S_{\mathcal{P}}$. A run of A on $t \in \text{TR}$, starting from s_0 , is a map $\rho : C_t \rightarrow S_{\mathcal{P}}$ such that $\rho(\emptyset) = s_0$ and for every $c, c' \in C_t$, $c \xrightarrow{a}_t c'$ implies that $(\rho(c), \rho(c')) \in \xRightarrow{a}$. Note that, as A is non-deterministic, there can be multiple runs of A on the same trace. The following lemma follows directly from the locality of transitions of A .

► **Lemma 5.** Let $t = (E, \leq, \lambda)$ be a trace and $\rho : C_t \rightarrow S_{\mathcal{P}}$ be a run of A on t starting at s_0 . Then ρ is determined by $\rho(\emptyset)$ and $\rho(\downarrow e)$ for each $e \in E$.

3 The model

Now we describe the model called **asynchronous transition system** games or simply **ATS** games. ATS games on two processes are studied in [1]. An ATS game is played on a non-deterministic asynchronous transition system between an *environment* and a *distributed system* that comprises of a *team of cooperating processes*. The environment is a singular entity and manifests itself in the form of a *scheduler* of actions. Importantly, the environment *does not reveal* any information about prior/concurrent scheduling events.

We continue with the notation from the previous section. In particular, we fix a distributed alphabet $\tilde{\Sigma}$ over a fixed set \mathcal{P} of processes.

- **Definition 6.** An ATS game is of the form $\mathcal{G} = (A, s_0, \text{Win})$ where
- $A = (\{S_i\}, \{\xrightarrow{a}\})$ is an asynchronous transition system over $\tilde{\Sigma}$.
 - $s_0 \in S_{\mathcal{P}}$ is an initial global state of A .
 - Win is a specification of the winning condition.

A play of an ATS game is inherently distributed, capturing the ongoing, possibly non-terminating interaction between the distributed system and the environment. Let us now formalize this notion of a *distributed play*. To this end, fix an ATS game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$.

We begin with a description of the interleaved semantics of a distributed play. Such a play begins in the initial global state s_0 . Environment makes the first move by *scheduling* an action (say a) which is enabled at s_0 . The processes participating *in* a respond by selecting an available a -transition and thus advancing the “current” global state (say, to s). At this point, it is environment’s turn to schedule another action (say b) which is enabled at s . The processes participating *in* b respond by choosing a suitable b -transition at s . Let us assume that a and b are independent actions. Then the processes participating in b are oblivious to the “concurrent” scheduling of “prior” a -action. It is important to observe that, in this situation, thanks to the locality of transitions in A , b is also enabled at s_0 and environment also had the option of scheduling b first and then a later. On the other hand, if a and b are dependent, then there is a process which participates in both a and b and every process participating in b *comes to know* about the prior a -action which is “causally” before the current b -action.

So, in this interleaved semantics of a distributed play, a play consists of an alternate sequence of moves of the environment (which schedules actions) and the distributed system (where the participating processes respond by matching transitions). Observe that each process in the distributed system has only partial information; among all the actions which are already scheduled, it only *knows* those which are in its *causal past*. The causal past of a process contains the information that a process comes to know directly or indirectly through its interactions with the other processes. Further when two or more processes interact (by participating in a shared/joint action), they exchange their complete causal pasts.

- **Definition 7.** A (partial and distributed) play of the game \mathcal{G} is a tuple (t, ρ) where
- $t = (E, \leq, \lambda)$ is a trace over $\tilde{\Sigma}$.
 - $\rho : C_t \rightarrow S_{\mathcal{P}}$ is a run of A on t starting at s_0

The idea is that the (labelled) events in t represent *enabled* scheduler choices and the partial-order \leq captures the causality between these events arising out of the distribution of these events across processes. The fact that an event e is labelled a (that is, $\lambda(e) = a$) means that the environment scheduled the shared action a which lead the processes participating

in a to synchronize. Their *collective causal past-information* at e corresponds to events in $\Downarrow e$. Based on this common information, these processes respond to e by choosing a local transition on a which advances their prior joint a -state s_a to s'_a where $s = \rho(\Downarrow e)$ and $s' = \rho(\Downarrow e)$. Thus the run ρ correctly captures the decisions of the processes during the play. Note that by Lemma 5, ρ is completely determined by $\rho(\emptyset) = s_0$ and $\rho(\Downarrow e)$ for each $e \in E$. In summary, the event e models environment's scheduling of action $\lambda(e)$ at the global state $\rho(\Downarrow e)$; and the $\lambda(e)$ -transition from $\rho(\Downarrow e)$ to $\rho(\Downarrow e)$ models the collective response of the processes participating in e .

A (distributed) play (t, ρ) is said to be *maximal* if there is no play (t', ρ') such that t is a proper prefix of t' and $\rho = \rho'|_{C_t}$, the restriction of ρ' to the configurations of t . Note that if (t, ρ) is maximal then the processes which have moved only finitely often in t can not be further scheduled by the environment as no action involving *only them* is enabled. It is important to note that a maximal play could be either finite or infinite. If a maximal play $(t = (E, \leq, \lambda), \rho)$ is finite, then no action is enabled at the final global state $\rho(E)$.

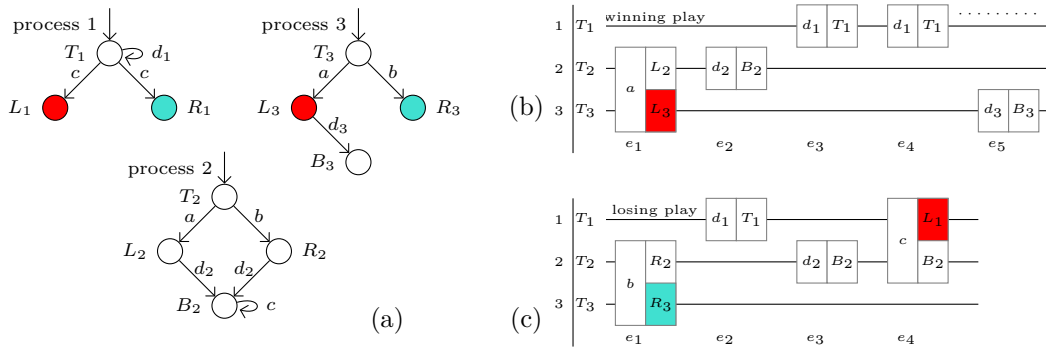
We next turn our attention to the winning condition Win which specifies the winner of a maximal play. At an abstract level, Win is simply the collection of all maximal plays in which the distributed system wins.

In this work we study ATS games from the viewpoint of *decision making processes*.

► **Definition 8.** Let $\mathcal{G} = (A, s_0, \text{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$. An action a of A is not deterministic if there exists $s_a, s'_a, s''_a \in S_a$ such that both (s_a, s'_a) and (s_a, s''_a) are a -transitions (that is, belong to \xrightarrow{a}) and $s'_a \neq s''_a$.

A subset $Q \subset \mathcal{P}$ of processes is said to be *decision makers* in \mathcal{G} if for every a which is not deterministic, $Q \cap \text{loc}(a) \neq \emptyset$. In other words, if no process in Q participates in an action b , then the transition relation \xrightarrow{b} of A is a (partial) deterministic function from S_b to S_b .

A **central decision maker (CDM)** game is an ATS game with decision makers $\{\ell\}$ for some process $\ell \in \mathcal{P}$. This central decision maker process ℓ participates in every action of A which is not deterministic.



■ **Figure 2** A safety CDM game with unsafe set $\{(L_1, B_2, R_3), (R_1, B_2, L_3)\}$ and its plays.

► **Example 9.** Consider a CDM game illustrated in Figure 2 with $\mathcal{P} = \{1, 2, 3\}$ and process 1 as the central decision maker. Note that only action c is *not* deterministic and it is shared by 1 and 2. Process i has purely local action d_i and actions a and b are shared by 2 and 3.

All 3 processes start in top local states and want to avoid states where colors mismatch. This can be captured as a global safety winning condition where color-mismatch signifies occurrence of an unsafe global state. Observe that, initially, the scheduler can play exactly

one of the deterministic actions a or b forcing the next joint state of process 2 and process 3 to be either (L_2, L_3) or (R_2, R_3) . It can concurrently schedule any number of d_1 actions. Afterwards, the scheduler can force process 2 to the bottom local state B_2 by playing local action d_2 . At this point, actions c and d_1 are enabled and the scheduler is allowed to play any of these actions. Note that, if process 3 is in local state L_3 then the scheduler must also schedule d_3 in a maximal play. Some plays of this CDM game are depicted in Figure 2.

If action c is never scheduled, the distributed system wins along such maximal plays. However, if action c is ever scheduled then the resulting response determines the winner. In order to win in such situations, the central decision maker process 1 needs to inspect its causal past to find out which of the actions a or b was played in the past. With this information, process 1 makes the correct decision and ensures a win for the team.

Now we are ready to define the important and crucial notion of a distributed strategy for an ATS game. Intuitively speaking, a distributed strategy is an advice function that the team of processes in the distributed system uses to respond to environment's actions. More importantly, this response can only depend on the collective causal history of the processes participating in this action. Further, the advice function does not restrict the choices available to the environment in all situations.

► **Definition 10.** A distributed strategy in \mathcal{G} is a partial function $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ with the smallest domain such that

- the domain of σ is prefix closed, that is, if $t \in \text{TR}^*$ is such that $\sigma(t)$ is defined and $t' \in \text{TR}^*$ is a prefix of t then $\sigma(t')$ is also defined.
- $\sigma(\epsilon) = s_0$ where ϵ denotes the empty trace.
- for every $t \in \text{TR}^*$ if $\sigma(t)$ is defined and a is enabled at $\sigma(t)$, then $\sigma(ta)$ is also defined, and $(\sigma(t), \sigma(ta)) \in \xrightarrow{a}$ (here ta denotes the concatenation of t with the singleton trace a).

It is easy to see that the first condition in the above definition, that the domain of σ is prefix-closed, is redundant. By requiring σ to have the smallest domain, the second and third conditions in the above definition ensure that σ becomes defined exactly on traces built by iteratively extending the empty trace along enabled actions, making the domain of σ prefix-closed.

A finite trace $t = (E, \leq, \lambda) \in \text{TR}^*$ is said to be *prime* if t has a unique maximum event. In other words, a prime trace t has an event $e \in E$ such that $E = \downarrow e$; we let $\text{last}(t)$ denote this maximum event of t . The next lemma states that a distributed strategy is determined by its effect on the prime traces.

► **Lemma 11.** Let $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ be a distributed strategy in \mathcal{G} . Then σ is completely determined by $\sigma(t)$ for every prime trace t in the domain of σ . Moreover, if \mathcal{G} is a CDM game with ℓ as a central decision maker then σ is completely determined by $\sigma(t)$ for every prime trace t in the domain of σ whose last action $\text{last}(t)$ involves ℓ .

► **Definition 12.** Let $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ be a distributed strategy in \mathcal{G} . A distributed play $(t, \rho : C_t \rightarrow S_{\mathcal{D}})$ of \mathcal{G} is said to conform σ if, for all $c \in C_t$, $\rho(c) = \sigma(c)$. Recall that, for a configuration c of t , c also denotes the induced finite trace $t_c = (c, \leq, \lambda)$.

A distributed strategy $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ is winning in \mathcal{G} if all maximal plays conforming it belong to Win. In other words, all maximal plays where the distributed team employs σ are won by the distributed system.

The description at the end of Example 9, for the CDM game in Figure 2, can be readily converted into a winning distributed strategy as in the above definition.

Given an ATS game $\mathcal{G} = (A, s_0, \text{Win})$, the key algorithmic question is to decide if there exists a distributed winning strategy in \mathcal{G} . We answer this for CDM games with global safety and local parity conditions. Note that a CDM game is a partial-information game and different processes have mutually incomparable partial informations about scheduler's actions; a process is oblivious to concurrent scheduling decisions on other processes and there is no bound on the number of such concurrent scheduling decisions. Clearly the decisions taken by the central decision maker (based solely on its causal past) influence the future behaviour of other processes. Observe that ATS games with only one process ($|\mathcal{P}| = 1$) are CDM games and are essentially full-information two-player games played on a finite graph [13].

4 Extraction of distributed strategies via special linearizations

We now develop a general construction which aids in the solution of CDM games. Let us fix a CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and process $1 \in \mathcal{P}$ as the central decision maker. Towards solving \mathcal{G} , we introduce a standard *full-information two-player* game G_{seq} played on a finite graph A_{seq} between two players called Sys and Env. See [13] for a study of these games and its relevance to reactive synthesis in the sequential setting.

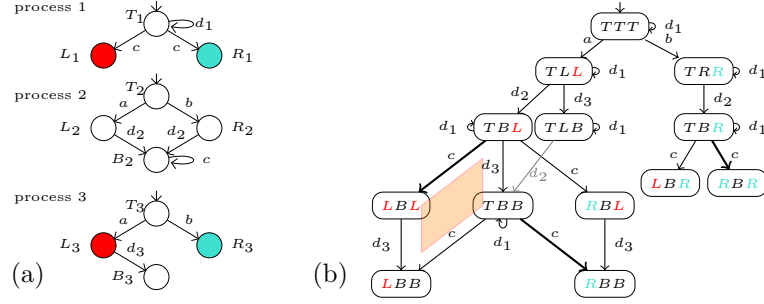
► **Definition 13.** *The game arena A_{seq} is a bipartite graph $A_{\text{seq}} = (V_{\text{env}}, V_{\text{sys}}, \rightsquigarrow \subseteq V_{\text{env}} \times V_{\text{sys}} \cup V_{\text{sys}} \times V_{\text{env}})$ whose vertices and directed edges (that is, elements of \rightsquigarrow) are as follows:*

- $V_{\text{env}} = S\mathcal{P}$ and $V_{\text{sys}} = \{(s, a) \in S\mathcal{P} \times \Sigma \mid a \text{ is enabled at } s\}$
- For $s \in V_{\text{env}}, (s', a) \in V_{\text{sys}}$, we have an edge from s to (s', a) iff $s = s'$
- For $(s', a) \in V_{\text{sys}}$ and $s \in V_{\text{env}}$, we have an edge from (s', a) to s iff $s' \xrightarrow{a} s$ in the ATS A . Recall that \xrightarrow{a} naturally extends the local transition relation \xrightarrow{a} of A to global-states.

We define $G_{\text{seq}} = (A_{\text{seq}}, s_0, \text{Win}_{\text{seq}})$ to be a standard two-player graph game of complete information whose winning condition Win_{seq} is currently *unspecified*. Note that $s_0 \in V_{\text{env}}$. The game G_{seq} is played by players Sys and Env by alternately moving a “token” along the edges of the bipartite graph A_{seq} . Player Env makes moves from V_{env} and player Sys makes moves from V_{sys} . Initially the token is at vertex $s_0 \in V_{\text{env}}$. Thus a play of G_{seq} is a sequence (possibly infinite) of vertices visited by the token starting with vertex s_0 and it is of the form: $s_0, (s_0, a_0), s_1, (s_1, a_1), s_2, \dots$. A maximal play is either infinite, or finite in which case it ends in $s \in V_{\text{env}}$ such that no action is enabled at s and as a result, there is no outgoing-edge in A_{seq} from the vertex s . As mentioned above, we do not specify the winning condition Win_{seq} in this section and it will be suitably instantiated in later sections.

It will be useful to view a play of G_{seq} as simply a sequence $s_0, a_0, s_1, a_1, s_2, \dots$ starting at s_0 such that, for each i , a_i is enabled at s_i and $(s_i, s_{i+1}) \in \xrightarrow{a_i}$. In this viewpoint, a_i corresponds to player Env moving the token from s_i to (s_i, a_i) and, s_{i+1} corresponds to player Sys moving the token subsequently from (s_i, a_i) to s_{i+1} .

► **Example 14.** Figure 3 shows an ATS A and a simplified view of the derived game arena A_{seq} . The simplified view represents the *sequential automaton* underlying A_{seq} . For instance, in the graph A_{seq} we have two edges from TBL (short for $T_1B_2L_3$) to (TBL, c) and (TBL, d_3) corresponding to two enabled actions, namely c and d_3 , at TBL . We also have two edges in A_{seq} from (TBL, c) to LBL and RBL corresponding to two global c -transitions $(TBL, LBL), (TBL, RBL)$ of A . A play $s_0, a_0, s_1, a_1, s_2, \dots$ starting at s_0 in G_{seq} described above, can be simply viewed as a *run* of this sequential automaton on $a_0a_1a_2, \dots$ from s_0 .



■ **Figure 3** An ATS A and the derived sequential game arena A_{seq} .

A (sequential full-information) strategy for player Sys in G_{seq} maps its history leading to current vertex to the next valid vertex. We formalize it as follows.

► **Definition 15.** A strategy for player Sys in G_{seq} is a partial function $\tau : \Sigma^* \rightarrow S_{\mathcal{D}}$ with the smallest domain such that

- $\tau(\epsilon) = s_0$ where ϵ denotes the empty word.
- for every $w \in \Sigma^*$ if $\tau(w)$ is defined and a is enabled at $\tau(w)$, then $\tau(wa)$ is also defined and furthermore $(\tau(w), \tau(wa)) \in \xRightarrow{a}$. In other words, we have a directed edge in A_{seq} from $(\tau(w), a) \in V_{\text{Sys}}$ to $\tau(wa) \in V_{\text{Env}}$.

It is easy to check that the domain of τ is a prefix-closed subset of Σ^* .

Henceforth, by a strategy in G_{seq} we always mean a strategy for the player Sys in G_{seq} . It turns out that every distributed strategy in \mathcal{G} naturally gives rise to a sequential strategy in G_{seq} . We develop some more notation to explain this. Each finite word $w \in \Sigma^*$ naturally induces a finite trace $t \in \text{TR}^*$. More precisely, if $w = a_1 a_2 \dots a_n$, we associate with w the finite trace $t = (E, \leq, \lambda)$ which is defined as follows: $E = \{e_1, e_2, \dots, e_n\}$ has n events corresponding to n positions in the word w ; we set $\lambda(e_i) = a_i$ and the partial order \leq on E is the transitive-closure of the relation $\{(e_i, e_j) \mid i \leq j \text{ and } (a_i, a_j) \in D\}$. It is easy to check that t is indeed a trace over $\tilde{\Sigma}$. Let us denote this association by the map $\eta : \Sigma^* \rightarrow \text{TR}^*$. It turns out [4, 3, 16] that for a trace $t \in \text{TR}^*$, the set $\eta^{-1}(t) \subseteq \Sigma^*$ is precisely all those words which correspond to different linearizations of t . In fact, η is a surjective monoid morphism and $\eta(w) = \eta(w')$ iff w can be obtained from w' by a sequence of operations where each operation is an *exchange* of two *adjacent independent* letters.

The next proposition states that, a distributed strategy σ in \mathcal{G} naturally induces a (sequential) strategy σ_{seq} in G_{seq} .

► **Proposition 16.** Let $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ be a distributed strategy in \mathcal{G} . Consider the induced partial function $\sigma_{\text{seq}} : \Sigma^* \rightarrow S_{\mathcal{D}}$ defined as follows: for $w \in \Sigma^*$, $\sigma_{\text{seq}}(w) = \sigma(\eta(w))$. So w belongs to the domain of σ_{seq} iff $\eta(w)$ belongs to the domain of σ . Then $\sigma_{\text{seq}} : \Sigma^* \rightarrow S_{\mathcal{D}}$ is in fact a strategy for player Sys in G_{seq} . Furthermore, σ_{seq} is diamond-closed; that is, for $w, w' \in \Sigma^*$ and $(a, b) \in I$, $\sigma_{\text{seq}}(wabw') = \sigma_{\text{seq}}(wbaw')$.

So, the strategies in G_{seq} induced by distributed strategies in \mathcal{G} are diamond-closed. However, an arbitrary strategy in G_{seq} is not necessarily diamond-closed.

► **Example 17.** In Figure 3(b), we have used **bold** edges to depict the response of a *positional* strategy in G_{seq} on the action c which is not deterministic. This strategy responds to c at TBL by using the bold edge to LBL , at TBB by using the bold edge to RBB and at TBR

by using the bold edge to RBR . As a result, at TBL the response to cd_3 ends at LBB while that for d_3c ends at RBB . As c and d_3 are independent, this strategy is not diamond-closed.

Our main goal in this section is to extract a distributed strategy in \mathcal{G} from an arbitrary strategy crucially exploiting the presence of a central decision maker. Towards this, we now develop special linearizations of finite traces.

We first fix a total order \triangleleft_Σ on Σ such that $\Sigma \setminus \Sigma_1 \triangleleft_\Sigma \Sigma_1$. Note that, in the total order \triangleleft_Σ , if $a, b \in \Sigma$ are such that $1 \notin \text{loc}(a)$ and $1 \in \text{loc}(b)$ then $a \triangleleft_\Sigma b$. With our implicit assumption that process 1 is the central decision maker, every action in which it does not participate comes before every action in which it participates in the order \triangleleft_Σ . We now outline an inductive procedure to compute a special linearization map $\text{Lin} : \text{TR}^* \rightarrow \Sigma^*$.

► **Definition 18.** Let $t = (E, \leq, \lambda) \in \text{TR}^*$. If $t = \epsilon$, then $\text{Lin}(t) = \epsilon$. Otherwise, let M be the set of all maximal (wrt \leq) events of the trace t and e be the unique event in M whose label is \triangleleft_Σ -least among the labels in $\{\lambda(f) \mid f \in M\}$. Note that no two events in M can have the same label as events with the same label are ordered/comparable wrt \leq .

It is easily verified that with t' as the trace corresponding to the configuration $E \setminus \{e\}$, we have $t = t'a$ where $a = \lambda(e)$. We now define $\text{Lin}(t) = \text{Lin}(t')a$.

In short, we compute $\text{Lin}(t)$ starting from its last action/letter. Towards this, we peel off that maximal event of t which is \triangleleft_Σ -least labelled and put its label as the last letter in $\text{Lin}(t)$ and continue this process until all events are covered.

Now we exploit the key consequences of the \triangleleft_Σ requirement that $\Sigma \setminus \Sigma_1 \triangleleft_\Sigma \Sigma_1$. It is very important to observe that, in general, t' is a trace-prefix of t does not imply that $\text{Lin}(t')$ is a word-prefix of $\text{Lin}(t)$. The following lemma plays a crucial role later.

► **Lemma 19.** Let $t = (E, \leq, \lambda) \in \text{TR}^*$ and $e \in E$ be such that process 1 participates in e , that is $\lambda(e) \in \Sigma_1$. Then $\text{Lin}(\downarrow e)$ is prefix of $\text{Lin}(t)$. Here we identify the configuration $\downarrow e$ with the corresponding trace-prefix $(\downarrow e, \leq, \lambda)$ of t .

► **Example 20.** We revisit the trace t from Figure 1 with process 1 as the central decision maker and event set $E = \{e_1, e_2, \dots, e_9\}$. Let the ordering \triangleleft_Σ be given by $a_i \triangleleft_\Sigma a_j$ iff $i \leq j$. Note that $\Sigma \setminus \Sigma_1 \triangleleft_\Sigma \Sigma_1$. We now compute $\text{Lin}(t)$ for the trace t . Maximal events of t are e_8 and e_9 and as $\lambda(e_9) = a_1 \triangleleft_\Sigma a_4 = \lambda(e_8)$, $\text{Lin}(t) = \text{Lin}(t').a_1$ where t' is given by $E' = E \setminus \{e_9\}$. Successively the maximal event sets are $\{e_8\}, \{e_6, e_7\}, \{e_6, e_5\}, \{e_6\}, \{e_2, e_4\}, \{e_2, e_3\}, \{e_2\}, \{e_1\}$. This results in $\text{Lin}(t) = a_6a_5a_2a_3a_6a_2a_4a_1$ with event-linearization $e_1e_2e_3e_4e_6e_5e_7e_8e_9$.

As an illustration of Lemma 19, consider event $e_6 \in E_1$. We have $\downarrow e_6 = \{e_1, e_2, e_3, e_4, e_6\}$ and $\text{Lin}(\downarrow e_6) = a_6a_5a_2a_3a_6$ which is a prefix of $\text{Lin}(t)$. However, $e_7 \notin E_1$ and one can verify that $\text{Lin}(\downarrow e_7) = a_6a_5a_2a_3a_2a_2$, which is not a prefix of $\text{Lin}(t)$.

Now we use special linearizations to lift a strategy in G_{seq} to a distributed strategy in \mathcal{G} . Intuitively, in the special linearization of a trace, all the events of process 1 appear the earliest. Thus the response of a sequential strategy along the special linearization is based on the least amount of information to the central decision maker. We lift these responses to construct a distributed strategy.

► **Definition 21.** Let $\tau : \Sigma^* \rightarrow \mathcal{S}_{\mathcal{D}}$ be a strategy in G_{seq} . We define a partial function $\tau_{\text{dstr}} : \text{TR}^* \rightarrow \mathcal{S}_{\mathcal{D}}$ as follows: for $t \in \text{TR}^*$, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$. Note that, t belongs to the domain of τ_{dstr} iff $\text{Lin}(t)$ belongs to the domain of τ .

Let us illustrate the above definition for the non-diamond-closed strategy τ from Example 17 with t being the trace induced by the word ad_2d_3c . As $\text{Lin}(t) = ad_2cd_3$, we set $\tau_{\text{dstr}}(t) = \tau(ad_2cd_3) = LBB$. Note that $\tau(ad_2d_3c) = RBB$.

We now state the following main proposition which shows that *partial function* τ_{dstr} from Definition 21 is in fact a distributed strategy in \mathcal{G} . The proof crucially uses Lemma 19 which implies that, restricted to central decision maker events, special linearization of a later event *extends* that of an earlier one.

► **Proposition 22.** *Let $\tau : \Sigma^* \rightarrow S_{\mathcal{D}}$ be a strategy for player Sys in G_{seq} . Then $\tau_{\text{dstr}} : \text{TR}^* \rightarrow S_{\mathcal{D}}$ is a distributed strategy in \mathcal{G} .*

5 Global safety and local parity CDM games

We turn our attention to CDM games with global safety and local parity objectives. For an ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$, $\max_i |S_i|$ denotes the maximum number of local states per process. For complexity analysis, we make the realistic assumption that every action involves at most a *fixed constant* number of participating processes. This is reasonable because in most distributed systems, each action involves only a few processes interacting at a time, e.g., reading/writing shared resources or coordinating with immediate “neighbors”. Thanks to this assumption and the local nature of transitions of A , it is easy to check that the size of A , denoted by $\|A\|$, is *polynomial* in $\max_i |S_i|$, $|\Sigma|$ and $|\mathcal{P}|$. Besides $\|A\|$, the other component which contributes to the size of a CDM game \mathcal{G} , denoted $\|\mathcal{G}\|$, is the specification of the winning condition. Observe that the number of vertices in the game arena A_{seq} (see Definition 13) is atmost $2 * |\Sigma| * (\max_i |S_i|)^{|\mathcal{P}|}$. Note that $(\max_i |S_i|)^{|\mathcal{P}|}$ is the maximum number of global-states of A . As a result, the size of A_{seq} , denoted by $\|A_{\text{seq}}\|$ is polynomial in $(\max_i |S_i|)^{|\mathcal{P}|}$ and $|\Sigma|$.

5.1 Global safety objective

Let us fix a CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and a global-safety winning condition. A global-safety winning condition is given by a subset $F \subseteq S_{\mathcal{D}}$ of *safe* global-states. A maximal play (t, ρ) is won by the distributed system if, for *all* $c \in C_t$, $\rho(c) \in F$. By abuse of notation, we also write $\mathcal{G} = (A, s_0, F)$ and refer to it as a safety CDM game.

Our solution for the safety CDM game uses the full information two-player token game $G_{\text{seq}} = (A_{\text{seq}}, s_0, \text{Win}_{\text{seq}})$ from the previous section. We instantiate the winning condition Win_{seq} and the resulting G_{seq} as follows:

► **Definition 23.** *Let $G_{\text{seq}} = (A_{\text{seq}}, s_0, F_{\text{seq}})$ be the full-information two-player safety graph game where the safety objective for player Sys is given by*

$$F_{\text{seq}} = \{s \in V_{\text{sys}} \mid s \in F\} \cup \{(s, a) \in V_{\text{env}} \mid s \in F\}$$

In order to win G_{seq} , player Sys must have a strategy to ensure that the token never leaves the safe-set F_{seq} of vertices of A_{seq} .

Recall that a strategy in G_{seq} (winning or not) always means a strategy for player Sys.

► **Definition 24.** *Let $\tau : \Sigma^* \rightarrow S_{\mathcal{D}}$ be a strategy in G_{seq} . A play $\alpha = s_0, a_0, s_1, a_1, s_2, \dots$ of G_{seq} conforms τ if, for all i , $\tau((a_0 a_1 \dots a_i)) = s_{i+1}$. The fact that τ is a strategy ensures that we have a directed edge in A_{seq} from (s_i, a_i) to s_{i+1} . The strategy τ is winning if all maximal plays $\alpha = s_0, a_0, s_1, a_1, s_2, \dots$ conforming it have the property that, for all i , $s_i \in F_{\text{seq}}$.*

► **Theorem 25.** *There is a distributed winning strategy in the safety CDM game \mathcal{G} iff there is a winning strategy in G_{seq} . Moreover, it can be decided in time polynomial (more*

accurately, quadratic) in the size of A_{seq} whether player Sys has a winning strategy in G_{seq} . The running time complexity of the resulting decision procedure for \mathcal{G} is polynomial in $(\max_i |S_i|)^{|\mathcal{P}|}, |\Sigma|, |F|$.

We consider the decision problem of determining whether a CDM game with global safety objectives admits a distributed winning strategy; the procedure above is EXPTIME due to the exponential dependence on $|\mathcal{P}|$. These games are also EXPTIME-hard. The proof is similar to the corresponding result for the Petri game model with one system token studied in [7]. Together these results imply the following theorem.

► **Theorem 26.** *Global safety CDM games are EXPTIME-complete.*

5.2 Local parity objective

Now we analyze CDM games with local parity objectives. Parity winning conditions are central to the theory of two-player graph games and are studied extensively in literature [13]. A standard parity game on a graph of size n and m colors can be solved in time $n^{m+O(1)}$ [15]. A recent advance from [2] brings it down to $n^{\log(m)+6}$ - a quasi-polynomial bound.

A local-parity winning condition is given by a color function $\chi : S_1 \rightarrow \{0, 1, \dots, k\}$ that assigns each CDM local state a color from the finite color set $C = \{0, 1, \dots, k\}$. Given a maximal play (t, ρ) with $t = (E, \leq, \lambda)$, we define $\text{inf}(t, \rho) = \{s \in S_1 \mid \exists^\infty e \in E_1, \rho(\downarrow e)(1) = s\}$. Note that if E_1 - the set of events in which process 1 participates, is finite then $\text{inf}(t, \rho) = \emptyset$. The system wins if $\text{inf}(t, \rho)$ is empty or $\max\{\chi(s)_{s \in \text{inf}(\rho)}\}$ is even. We also denote such a game by $\mathcal{G} = (A, s_0, \chi)$ and refer to it as a (local) parity CDM game. Towards solving \mathcal{G} , we instantiate G_{seq} by an appropriate *sequential* parity condition.

► **Definition 27.** *Let $G_{\text{seq}} = (A_{\text{seq}}, s_0, \chi_{\text{seq}})$ be the standard two-player parity game where $\chi_{\text{seq}} : V_{\text{sys}} \cup V_{\text{env}} \rightarrow \{0, 1, \dots, k\}$ is defined as: $\chi_{\text{seq}}(s) = 0$ and*

$$\begin{aligned} \chi_{\text{seq}}((s, a)) &= \chi(s(1)) \text{ if } a \in \Sigma_1 \\ &= 0 \text{ if } a \notin \Sigma_1 \end{aligned}$$

We now describe the winning condition Win_{seq} of G_{seq} using χ_{seq} . As discussed in Section 4, a maximal play of G_{seq} may be viewed as a sequence $\alpha = s_0, a_0, s_1, a_1, s_2, \dots$ and it corresponds to the maximal movement α' of the token along the path $s_0, (s_0, a_0), s_1, (s_1, a_1), s_2, \dots$ in A_{seq} . On this maximal sequence α , player Sys wins if it is finite or the highest color occurring infinitely often in $\chi_{\text{seq}}^{\alpha'} = \chi_{\text{seq}}(s_0), \chi_{\text{seq}}((s_0, a_0)), \chi_{\text{seq}}(s_1), \chi_{\text{seq}}((s_1, a_1)), \chi_{\text{seq}}(s_2), \dots$ is even. The following lemma brings out the connection between χ_{seq} and χ .

► **Lemma 28.** *If α is finite then player Sys wins in α . We now assume that α is infinite. If actions from Σ_1 occur finitely often in α then only color 0 occurs infinitely often in $\chi_{\text{seq}}^{\alpha'}$ and player Sys wins in α . If actions from Σ_1 occur infinitely often in α , consider the infinite subsequence β' obtained from α' by restricting it to vertices in $V_{\text{sys}} \cap (S_{\mathcal{P}} \times \Sigma_1)$. Suppose $\beta' = (s_{i_1}, a_{i_1}), (s_{i_2}, a_{i_2}), \dots$. Then the highest color occurring infinitely often in $\chi_{\text{seq}}^{\beta'} = \chi_{\text{seq}}((s_{i_1}, a_{i_1})), \chi_{\text{seq}}((s_{i_2}, a_{i_2})), \dots$ is same as that of $\chi_{\text{seq}}^{\alpha'}$. Furthermore it is equal to the highest color occurring infinitely often in the sequence $\chi(s_{i_1}(1)), \chi(s_{i_2}(1)), \dots$*

Now we state our main result for local parity CDM games.

► **Theorem 29.** *There is a distributed winning strategy in the local parity CDM game \mathcal{G} iff the player Sys has a winning strategy in G_{seq} .*

Given a local parity CDM game \mathcal{G} , the decision problem is to determine whether the distributed system has a winning strategy. Thanks to the assumption that each action involves at most a fixed constant number of processes, the size $||\mathcal{G}||$ is polynomial in $\max_i |S_i|$, $|\Sigma|$, $|\mathcal{P}|$, and $|C|$, where C is the color set. The derived standard parity game G_{seq} has arena size polynomial in $(\max_i |S_i|)^{|\mathcal{P}|}$ and $|\Sigma|$ and it uses only $|C|$ colors. Known algorithms for parity games on arenas of size n with m colors run in time $n^{m+O(1)}$ [15] or $n^{\log(m)+6}$ [2], and applying them to G_{seq} gives an EXPTIME procedure for \mathcal{G} . Moreover, we can also show that local parity CDM games are EXPTIME-hard even with two colors, yielding the following theorem.

► **Theorem 30.** *Local parity CDM games are EXPTIME-complete.*

6 Finite-state distributed strategies

In this section we investigate distributed strategies in CDM games which admit a finite-state implementation. Let $\mathcal{G} = (A, s_0, \text{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$.

► **Definition 31.** *A memory automaton for \mathcal{G} is a deterministic asynchronous automaton $\mathcal{A} = (\{S_i \times M_i\}, \{\delta_a : S_a \times M_a \rightarrow S_a \times M_a\}, (s_0, m_0))$ over $\tilde{\Sigma}$ such that*

- *For $a \in \Sigma$, $(s_a, m_a) \in S_a \times M_a = \Pi_{i \in \text{loc}(a)}(S_i \times M_i)$, if a is enabled at s_a in A then a is also enabled at (s_a, m_a) in \mathcal{A} . Further, with $\delta_a((s_a, m_a)) = (s'_a, m'_a)$, $(s_a, s'_a) \in \xrightarrow{a}$; that is, there is an a -transition from s_a to s'_a in A .*
- *$(s_0, m_0) \in S_{\mathcal{D}} \times M_{\mathcal{D}}$ is a global state of \mathcal{A} where s_0 is the initial global state of A .*

In the memory automaton \mathcal{A} , process i has access to local memory states M_i . The memory automaton is a distributed automaton which starts in the initial global state (s_0, m_0) and provides a deterministic response to *every* enabled scheduler action using the current joint memory-state. Further, it also updates the joint memory-state deterministically.

A memory automation is *zero-memory* if for each i , M_i is a singleton set. A zero-memory automaton may be viewed as a deterministic *asynchronous sub-automaton* of A .

► **Definition 32.** *Let \mathcal{A} be a memory automaton for \mathcal{G} and $\sigma : \text{TR}^* \rightarrow S_{\mathcal{D}}$ be a distributed strategy in \mathcal{G} . We say that σ is realized by \mathcal{A} if, for every trace t in the domain of σ , $\sigma(t) = s$ where $\mathcal{A}(t) = (s, m)$ – here $\mathcal{A}(t)$ is the final global state attained on the unique run of \mathcal{A} on t .*

A distributed strategy σ in \mathcal{G} is said to be finite-state if there exists \mathcal{A} which realizes it. Thus the responses of σ are determined by the finite-state device \mathcal{A} . A distributed strategy σ realized by a zero-memory automaton is called a zero-memory distributed strategy.

It is well-known (see [13]) that for standard two-player full-information sequential games with safety and parity objectives, existence of a winning strategy implies existence of a positional/“zero-memory” winning strategy. Unfortunately, this is not so for CDM games. It is easy to verify that the safety CDM game from Figure 2 does *not* admit a *zero-memory distributed* winning strategy. However, we have already exhibited a valid distributed winning strategy for this game in Example 9.

Let us fix a safety/parity CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$. Recall that in Sections 5.1 and 5.2, we have shown that we can extract a distributed winning strategy τ_{dstr} in \mathcal{G} from a sequential winning strategy τ in G_{seq} . The key property shared by G_{seq} in both instantiations is that: if player Sys has a winning strategy in G_{seq} then it also has a positional winning strategy. Our main goal now is to obtain a memory automaton realizing the distributed strategy τ_{dstr} in \mathcal{G} which is extracted from a positional strategy τ in G_{seq} .

► **Definition 33.** Let $\tau : \Sigma^* \rightarrow S_{\mathcal{P}}$ be a strategy in G_{seq} . We call τ *positional* if there exists a function $f_{\tau} : V_{\text{sys}} \rightarrow V_{\text{env}}$ such that, for every w in the domain of τ and for every a which is enabled at $\tau(w)$, $\tau(wa) = f_{\tau}((\tau(w), a))$. We refer to f_{τ} as the *witness function* of τ .

► **Proposition 34.** Let $\tau : \Sigma^* \rightarrow S_{\mathcal{P}}$ be a positional strategy in G_{seq} with $f_{\tau} : V_{\text{sys}} \rightarrow V_{\text{env}}$ as the witness function of τ . Further, let $\tau_{\text{dstr}} : \text{TR}^* \rightarrow S_{\mathcal{P}}$ be the “extracted” distributed strategy in \mathcal{G} defined as: for $t \in \text{TR}^*$, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$. Then, for a prime trace t of the form $t = t'a$, $\tau_{\text{dstr}}(t) = f_{\tau}((\tau_{\text{dstr}}(t'), a))$.

The above proposition shows that the response of τ_{dstr} at the last a -event e of a prime trace $t = t'a$ can be determined (using f_{τ}) by the processes participating in e provided they can compute $\tau_{\text{dstr}}(t')$. Note that $t' = \Downarrow e$ is their collective causal past at e . Hence $\tau_{\text{dstr}}(\Downarrow e)$ represents the *last/latest global-state* about the entire system that they are aware of at e . This suggests that, in order to realize a finite-state implementation of τ_{dstr} , each process should keep track of the latest global-state that it is aware of, in its local memory. When a subset of processes synchronize on a shared action, they need a *finite-state* mechanism to compute the best global-state that they are *collectively* aware of. It turns out that this can be achieved with the help of the *gossip automaton* from [17].

We first develop some more notation. Let $t = (E, \leq, \lambda)$ be a finite trace, $i \in \mathcal{P}$ and $P \subseteq \mathcal{P}$. Recall that E_i is the set of events in which process i participates. We define $\partial_i(t)$ to be the trace induced by $\downarrow E_i$ – the events in the causal past of process i . Similarly, $\partial_P(t)$ is the trace induced by $\cup_{j \in P} \downarrow E_j$ and it represents the collective causal past of processes in P . We define function $\text{latest}_Q : \text{TR}^*(\Sigma) \times \mathcal{P} \rightarrow 2^Q$ that gives which processes in the set Q have the latest information about a given process k in a given trace t . For $Q = \{i_1, i_2, \dots, i_l\}$, $j \in \text{latest}_Q(t, k)$ iff $\partial_k(\partial_Q(t)) = \partial_k(\partial_j(t))$. We now state the key result from [17].

► **Theorem 35.** There exists an effectively constructible deterministic complete asynchronous automaton $A_{\text{gossip}} = (\{V_i\}, \{\nabla_a\}, v_0)$ over $\tilde{\Sigma}$, called the *gossip automaton*, and, for each $Q = \{i_1, i_2, \dots, i_l\} \subseteq \mathcal{P}$ there exists an effectively computable function $\text{gossip}_Q : V_{i_1} \times \dots \times V_{i_l} \times \mathcal{P} \rightarrow 2^Q$ such that, for every trace t and every process k , $\text{latest}_Q(t, k) = \text{gossip}_Q(v_Q, k)$, where v is the global state of A_{gossip} reached on the unique run of A_{gossip} on t .

We now describe the memory automaton \mathcal{A} which realizes τ_{dstr} from Proposition 34. We set $\mathcal{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ where for each i , $M_i = S_{\mathcal{P}} \times V_i$. So, a local memory i -state $m_i = (q_i, v_i) \in M_i$ of process i contains a local gossip i -state v_i and a global state $q_i \in S_{\mathcal{P}}$ in \mathcal{A} . Intuitively, q_i is the best global state that process i is aware of. The initial global memory state $m_0 \in M_{\mathcal{P}}$ is defined as: for each $i \in \mathcal{P}$, $m_0(i) = (s_0, v_0(i))$. Note that s_0 (resp. v_0) is the initial global state of \mathcal{A} (resp. A_{gossip}).

We now turn our attention to the definition of the transition function δ_a of \mathcal{A} . Let us suppose that $\text{loc}(a) = Q = \{i_1, i_2, \dots, i_l\}$. Fix $(s_a, m_a) \in S_a \times M_a$ such that a is enabled at s_a in \mathcal{A} . For each $i \in \text{loc}(a)$, $m_a(i) = (q_i, v_i) \in S_{\mathcal{P}} \times V_i$. Recall that ∇_a is the a -transition function of A_{gossip} and let $v_a \in V_a$ be such that, for each $i \in \text{loc}(a)$, $v_a(i) = v_i$ and $v'_a = \nabla_a(v_a)$. We now define $\delta_a((s_a, m_a))$ as follows. We first compute the best global state $\hat{s} \in S_{\mathcal{P}}$ that processes in $Q = \text{loc}(a)$ are aware of. For each $k \in \mathcal{P}$, we set

$$\hat{s}(k) = q_j(k) \text{ where } j \in Q \text{ is such that } j \in \text{gossip}_Q((v_{i_1}, v_{i_2}, \dots, v_{i_l}), k)$$

We next compute $\hat{s}' = f_{\tau}((\hat{s}, a))$ and define $\delta_a((s_a, m_a))$ to be (\hat{s}', m'_a) where, for each $i \in \text{loc}(a)$, $m'_a(i) = (\hat{s}', v'_a(i))$.

► **Theorem 36.** The distributed strategy τ_{dstr} extracted from the positional sequential strategy τ is finite-state. In fact, it is realized by the memory automaton \mathcal{A} defined above.

Proof idea. We prove by induction on the size of the trace t that if $\mathcal{A}(t) = (s, m)$ with $m = (q, v)$, then $q_i = \tau_{\text{dstr}}(\partial_i(t))$ and $s = \tau_{\text{dstr}}(t)$. The base case is immediate.

For the induction step, let t get extended by a . By induction, q_i correctly tracks τ_{dstr} on the views of each process. Since τ is memoryless, $\tau_{\text{dstr}}(ta)_a = f_\tau(\tau_{\text{dstr}}(\partial_a(t)), a)_a$. Using the gossip construction, let $j = \text{gossip}_a(v_a, k)$. Then, by Theorem 35, $j = \text{latest}_a(t, k)$ and $\partial_k(\partial_a(t)) = \partial_k(\partial_j(t))$. Thus, the intermediate state \hat{s} satisfies $\hat{s}(k) = \tau_{\text{dstr}}(\partial_k(\partial_a(t)))_k$ for each k , hence $\hat{s} = \tau_{\text{dstr}}(\partial_a(t))$. Updating \mathcal{A} along a then correctly sets $q'_i = \tau_{\text{dstr}}(\partial_i(ta))$ and $s' = \tau_{\text{dstr}}(ta)$ for all i , completing the induction. \blacktriangleleft

When applied to safety/parity CDM games, the above theorem provides a memory automaton realizing a distributed winning strategy. Note that A_{gossip} does not depend on the specification of the game \mathcal{G} at all. *Distributed memory complexity* $\max_i |M_i|$ of this memory automaton is bounded by $(\max_i |S_i|)^{|\mathcal{P}|} \cdot (\max_i |V_i|)$. This exponential dependence on \mathcal{P} is due to the fact that each process stores in its local memory state a *global-state* of A . Our next result shows that this exponential dependence on \mathcal{P} is necessary. Its proof is based on an adaptation of the memory lower bound argument for *generalized reachability* in sequential games from [5].

► **Theorem 37.** *For every n , there exists a global safety CDM game $\mathcal{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ with $|\mathcal{P}| = n + 1$ such that the central decision maker process has $O(n)$ local states and every other process has $O(1)$ local states. Further, there exists a distributed winning strategy in \mathcal{G} . However, any memory automaton realizing a winning distributed strategy in \mathcal{G} must have at least 2^n local memory states for the central decision maker.*

7 Discussion

For CDM games – where a single decision maker process participates in all non-deterministic choices – we have established two main results: (i) the problems of deciding the existence of winning strategies for global safety and local parity objectives are EXPTIME-complete, and (ii) constructions of distributed finite-state implementations of these strategies (when they exist) with optimal memory structures.

It is natural to ask about the decidability status of ATS games in the presence of multiple decision making processes. Towards this, we establish the following theorem.

► **Theorem 38.** *ATS games with two decision makers are undecidable.*

Our proof of the above theorem uses the key ideas from [12] which showed that six-process asynchronous control games are undecidable. More precisely, we use the intermediate problem *infinite bipartite coloring* introduced in [12] for our reduction. We then revisit the ideas in the proof of the undecidability of six-process asynchronous control games and adapt them to the setting of fourteen-process safety ATS games with only two decision making processes. The undecidability proof relies crucially on the ability of the two decision makers to take truly *concurrent* decisions.

Interestingly, if decision events are causally ordered, even across disjoint sets of processes, the arguments presented in this work can be adapted to recover decidability; we presented them in the CDM setting for convenience. It would be interesting to generalize the results in the CDM setting to arbitrary ω -regular objectives. Identifying natural decidable extensions of the CDM games is also a potential future direction.

References

- 1 Bharat Adsul and Nehul Jain. Asynchronous transition system games for two processes and their analysis. In *11th Indian Conference on Logic and Its Applications (ICLA 2025)*, volume 15402 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2025. doi:10.1007/978-3-031-89610-1_5.
- 2 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022.
- 3 Volker Diekert and Yves Métivier. Partial commutation and traces. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 457–533. Springer, 1997. doi:10.1007/978-3-642-59126-6_8.
- 4 Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific Publishing Co., Inc., USA, 1995.
- 5 Nathanaël Fijalkow and Florian Horn. The surprising complexity of generalized reachability games, 2012. arXiv:1010.2420.
- 6 Bernd Finkbeiner, Manuel Giesekeing, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, 2022. doi:10.4230/LIPICS.CSL.2022.20.
- 7 Bernd Finkbeiner and Paul Gözl. Synthesis in Distributed Environments. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:14, 2018. doi:10.4230/LIPICS.FSTTCS.2017.28.
- 8 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation*, 253:181–203, 2017. doi:10.1016/J.IC.2016.07.006.
- 9 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.
- 10 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.
- 11 Hugo Gimbert. On the Control of Asynchronous Automata. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, 2018. doi:10.4230/LIPICS.FSTTCS.2017.30.
- 12 Hugo Gimbert. Distributed asynchronous games with causal memory are undecidable. *Logical Methods in Computer Science*, Volume 18, Issue 3, September 2022. doi:10.46298/lmcs-18(3:30)2022.
- 13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.
- 14 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International*

- Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005. doi:10.1007/11590156_16.
- 15 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.
 - 16 Madhavan Mukund. Automata on distributed alphabets. In *Modern Applications of Automata Theory*, IISc Research Monographs Series 2, pages 257–288. World Scientific, 2012. doi:10.1142/9789814271059_0009.
 - 17 Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Computing*, 10(3):137–148, 1997. doi:10.1007/s004460050031.
 - 18 Anca Muscholl and Igor Walukiewicz. Distributed Synthesis for Acyclic Architectures. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, volume 29 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 639–651, 2014. doi:10.4230/LIPICS.FSTTCS.2014.639.
 - 19 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.
 - 20 Wiesław Zielonka. Notes on finite asynchronous automata. *RAIRO-Theoretical Informatics and Applications*, 21(2):99–135, 1987. doi:10.1051/ITA/1987210200991.