Controlling a Population

Nathalie Bertrand[1], Miheer Dewaskar[2], Blaise Genest[3], and Hugo Gimbert[4]

1 **Inria, IRISA, Rennes, France**
2 **CMI, Chennai, India**
3 **CNRS, IRISA, Rennes, France**
4 **CNRS, LaBRI, Bordeaux, France**

Abstract

We introduce a new setting where a population of agents, each modelled by a finite-state system, are controlled uniformly: the controller applies the same action to every agent. The framework is largely inspired by the control of a biological system, namely a population of yeasts, where the controller may only change the environment common to all cells. We study a synchronisation problem for such populations: no matter how individual agents react to the actions of the controller, the controller aims at driving all agents synchronously to a target state. The agents are naturally represented by a non-deterministic finite state automaton (NFA), the same for every agent, and the whole system is encoded as a 2-player game. The first player (Controller) chooses actions, and the second player (Agents) resolves non-determinism for each agent. The game with m agents is called the m-population game. This gives rise to a parameterized control problem (where control refers to 2 player games), namely the *population control problem*: can Controller control the m-population game for all $m \in \mathbb{N}$ whatever Agents does?

In this paper, we prove that the population control problem is decidable, and it is a EXPTIME-complete problem. As far as we know, this is one of the first results on parameterized control. Our algorithm, not based on cut-off techniques, produces winning strategies which are symbolic, that is, they do not need to count precisely how the population is spread between states. We also show that if there is no winning strategy, then there is a population size M such that Controller wins the m-population game if and only if $m \leq M$. Surprisingly, M can be doubly exponential in the number of states of the NFA, with tight upper and lower bounds.

1 Introduction

Finite-state controllers, implemented by software, find applications in many different domains: telecommunication, planes, etc. There have been many theoretical studies from the model checking community to show that finite-state controllers are sufficient to control systems in idealised settings. Usually, the problem would be modeled as a game: some players model the controller, and some players model the system [5], the game settings (number of players, their power, their observation) depending on the context.

Lately, finite-state controllers have been used to control living organisms, such as a population of yeasts [23]. In this application, microscopy is used to monitor the fluorescence level of a population of yeasts, reflecting the concentration of some molecule, which differs from cell to cell. Finite-state systems can model a discretisation of the population of yeasts [3,23]. The frequency and duration of injections of a sorbitol solution can be controlled, being injected uniformly into a solution in which the yeast population is immerged. However, the response of each cell to the osmotic stress induced by sorbitol varies, influencing the concentration of the fluorescent molecule. The objective is to control the population to drive it through a sequence of predetermined fluorescence states.

In this paper, we model this system of yeasts in an *idealised* setting: we require the (perfectly-informed) controller to surely lead synchronously all agents of a population to a

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

state (one of the predetermined fluorescence states). Such a population control problem does not fit in traditional frameworks from the model checking community. We thus introduce the *m-population game*, where a population of m identical agents is controlled uniformly. Each agent is modeled as a nondeterministic finite-state automaton (NFA), the same for each agent. The first player, called Controller, applies the same action, a letter from the NFA alphabet, to every agent. Its opponent, called Agents, chooses the reaction of each individual agent. These reactions can be different due to non determinism. The objective for Controller is to gather all agents synchronously in the target state (which can be a sink state w.l.o.g.), and Agents seeks the opposite objective. While this idealised setting may not be entirely satisfactory, it constitutes a simple setting, as a first step towards more complex settings.

Dealing with large populations *explicitly* is in general intractable due to the state-space explosion problem. We thus consider the associated *symbolic parameterized control problem*, asking to reach the goal independently of the population size. We prove that this problem is decidable. While *parameterized verification* received recently quite some attention (see related work), our results are one of the first on *parameterized control*, as far as we know.

Our results. We first show that considering an infinite population is not equivalent to the parameterized control problem for all non zero integer m: there are cases where Controller cannot control an infinite population but can control every finite population. Solving the ∞-population game reduces to checking a reachability property on the support graph [21], which can be easily done in PSPACE. On the other hand, solving the parameterized control problem requires new proof techniques, data structures and algorithms.

We easily obtain that when the answer to the population control problem is negative, there exists a population size M, called the *cut-off*, such that Controller wins the m-population game if and only if $m \leq M$. Surprisingly, we obtain a lower-bound on the cut-off doubly exponential in the number of states of the NFA. Following usual cut-off techniques would thus yield an inefficient algorithm of complexity at least 2EXPTIME.

To obtain better complexity, we developped new proof techniques (*not* based on cut-off techniques). Using them, we prove that the population control problem is EXPTIME-complete. As a byproduct, we obtain a doubly exponential upper-bound for the cut-off, matching the lower-bound. Our techniques are based on a reduction to a parity game with exponentially many states and a polynomial number of priorities. The parity game gives some insight on the winning strategies of Controller in the m-population games. Controller selects actions based on a set of *transfer graphs*, giving for each current state the set of states at time i from which agent came from, for different values of i. We show that it suffices for Controller to remember at most a quadratic number of such transfer graphs, corresponding to a quadratic number of indices i. If Controller wins this parity game then he can uniformly apply his winning strategy to all m-population games, just keeping track of these transfer graphs, independently of the exact count in each state. If Agents wins the parity game then he also has a uniform winning strategy in m-population games, for m large enough, which consists in splitting the agents evenly among all transitions of the transfer graphs.

Related work. Parameterized verification of systems with many identical components started with the seminal work of German and Sistla in the early nineties [16], and received recently quite some attention. The decidability and complexity of these problems typically depend on the communication means, and on whether the system contains a leader (following a different template) as exposed in the recent survey [13]. This framework has been extended to timed automata templates [1, 2] and probabilistic systems with Markov decision processes templates [6, 7]. Another line of work considers population protocols [4, 15]. Close in spirit, are broadcast protocols [14], in which one action may move an arbitrary number of agents from one state to another. Our model can be modeled as a subclass of broadcast protocols, where broadcasts emissions are self loops at a unique state, and no other synchronisation allowed. The parameterized reachability question considered for broadcast protocols is trivial in our framework, while our parameterized control question would be undecidable for broadcast protocols. In these different works, components interact directly, while in our work, the interaction is indirect via the common action of the controller. Further, the problems considered in related work are pure verification questions, and do not tackle the difficult issue of synthesising a controller for all instances of a parameterized system, which we do.

There are very few contributions pertaining to parameterized games with more than one player. The most related is [20], which proves decidability of control of mutual exclusion-like protocols in the presence of an unbounded number of agents. Another contribution in that domain is the one of broadcast networks of identical parity games [7]. However, the game is used to solve a verification (reachability) question rather than a parametrized control problem as in our case. Also the roles of the two players are quite different.

The winning condition we are considering is close to *synchronising words*. The original synchronising word problem asks for the existence of a word w and a state q of a *deterministic* finite state automaton, such that no matter the initial state s, reading w from s would lead to state q (see [24] for a survey). Lately, synchronising words have been extended to NFAs [21]. Compared to our settings, the author assumes a possibly infinite population of agents, who could leak arbitrarily often from a state to another. The setting is thus not parametrized, and a usual support arena suffices to obtain a PSPACE algorithm. Synchronisation for probabilistic models [11, 12] have also been considered: the population of agents is not finite nor discrete, but rather continuous, represented as a distribution. The distribution evolves deterministically with the choice of the controller (the probability mass is split according to the probabilities of the transitions), while in our setting, each agent can non deterministically move. This continuous model makes the parameterized verification question moot. In [11], the controller needs to apply the same action whatever the state the agents are in (like our setting), and then the existence of a controller is undecidable. In [12], the controller can choose the action depending on the state each agent is in (unlike our setting), and the existence of a controller reaching uniformly a set of states is PSPACE-complete.

Last, our parameterized control problem can be encoded as a 2-player game on VASS [9], with one counter per state of the NFA: the opponent gets to choose the population size (encoded as a counter value), and for each action chosen by the controller, the opponent chooses how to move each agent (decrementing a counter and incrementing another). However, such a reduction yields a symmetrical game on VASS in which both players are allowed to modify the counter values, in order to check that the other player did not cheat. Symmetrical games on VASS are undecidable [9], and their asymmetric variant (in which only one player is allowed to change the counter values) are decidable in 2EXPTIME [19], thus with higher complexity than our specific parameterized control problem.

2 The population control problem

2.1 The m-population game

A nondeterministic finite automaton (NFA for short) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ with Q a finite set of states, Σ a finite alphabet, $q_0 \in Q$ an initial state, and $\Delta \subseteq Q \times \Sigma \times Q$ the transition relation. We assume throughout the paper that NFAs are complete, that is, $\forall q \in Q, a \in \Sigma, \exists p \in Q : (q, a, p) \in \Delta$. In the following, incomplete NFAs, especially in figures, have to be understood as completed with a sink state.

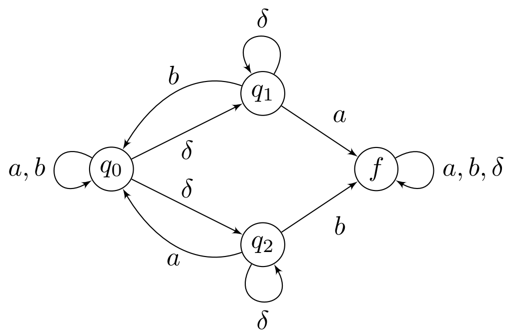


Figure 1 An exemple of NFA: The splitting gadget $\mathcal{A}_{\mathsf{split}}$.

For every integer m, we consider a system $\mathcal{A}^m$ with m identical agents $\mathcal{A}_1, \ldots, \mathcal{A}_m$ of the NFA $\mathcal{A}$. The system $\mathcal{A}^m$ is itself an NFA $(Q^m, \Sigma, q_0^m, \Delta^m)$ defined as follows. Formally, states of $\mathcal{A}^m$ are called configurations, and they are tuples $\mathbf{q} = (q_1, \ldots, q_m) \in Q^m$ describing the current state of each agent in the population. We use the shorthand $\mathbf{q}_0[m]$, or simply $\mathbf{q}_0$ when m is clear from context, to denote the initial configuration $(q_0, \ldots, q_0)$ of $\mathcal{A}^m$. Given a target state $f \in Q$, the f-synchronizing configuration is $f^m = (f, \ldots, f)$ in which each agent is in the target state.

The intuitive semantics of $\mathcal{A}^m$ is that at each step, the same action from Σ applies to all agents. The effect of the action however may not be uniform given the nondeterminism present in $\mathcal{A}$: we have $((q_1, \ldots, q_m), a, (q'_1, \ldots, q'_m)) \in \Delta^m$ iff $(q_j, a, q'_j) \in \Delta$ for all $j \leq m$. A (finite or infinite) play in $\mathcal{A}^m$ is an alternating sequence of configurations and actions, starting in the initial configuration: $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \cdots$ such that $(\mathbf{q}_i, a_i, \mathbf{q}_{i+1}) \in \Delta^m$ for all i.

This is the *m-population game* between Controller and Agents, where Controller chooses the actions and Agents chooses how to resolve non-determinism. The objective for Controller is to gather all agents synchronously in f while Agents seeks the opposite objective.

Our parameterized control problem asks whether Controller can win the m-population game for every $m \in \mathbb{N}$. A strategy of Controller in the m-population game is a function mapping finite plays to actions, $\sigma : (Q^m \times \Sigma)^* \times Q^m \to \Sigma$. A play $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \mathbf{q}_2 \cdots$ is said to *respect* σ, or is a *play under* σ, if it satisfies $a_i = \sigma(\mathbf{q}_0 a_0 \mathbf{q}_1 \cdots \mathbf{q}_i)$ for all $i \in \mathbb{N}$. A play $\pi = \mathbf{q}_0 a_0 \mathbf{q}_1 a_1 \mathbf{q}_2 \cdots$ is *winning* if it hits the f-synchronizing configuration, that is $\mathbf{q}_j = f^m$ for some $j \in \mathbb{N}$. Controller wins the m-population game if he has a strategy such that all plays under this strategy are winning. One can assume without loss of generality that f is a sink state. If not, it suffices to add a new action leading tokens from f to the new target sink state ☺ and tokens from other states to a losing sink state ☹. The goal of this paper is to study the following parameterized control problem:

Population control problem
Input: An NFA $\mathcal{A} = (Q, q_0, q_u, \Sigma, \Delta)$ and a target state $f \in Q$.
Output: Yes iff for every integer m Controller wins the m-population game.

For a fixed m, the winner of the m-population game can be determined by solving the underlying reachability game with $|Q|^m$ states, which is intractable for large values of m. On the other hand, the answer to the population control problem gives the winner of the m-population game for arbitrary large values of m. To obtain a decision procedure for this parameterised problem, new data structures and algorithmic tools need to be developed, much more elaborate than the standard algorithm solving reachability games.

▶ **Example 1.** We illustrate the population control problem with the example $\mathcal{A}_{\mathsf{split}}$ on alphabet $\{a, b, \delta\}$ in Figure 1. Here, to represent configurations we use a counting abstraction, and identify $\mathbf{q}$ with the vector (n_0, n_1, n_2, n_3), where n_0 is the number of agents in state q_0, and so on. Under these notations, there is a way to gather agents synchronously to f. We can give a symbolic representation of a memoryless winning strategy σ: $\forall k_0, k_1 > 0,\ \forall k_2, k_3 \geq 0,\ \sigma(k_0, 0, 0, k_3) = \delta,\ \sigma(0, k_1, k_2, k_3) = a,\ \sigma(0, 0, k_2, k_3) = b$. Indeed, the number of agents outside f decreases by at least one at every other step. The properties of this example will be detailed later and play a part in proving a lower bound (see Proposition 19).

2.2 Parameterized control and cut-off

A first observation for the population control problem is that $\mathbf{q}_0[m]$, f^m and Q^m are stable under a permutation of coordinates. A consequence is that the m-population game is also symmetric, and thus the set of winning configurations is symmetric and the winning strategy can be chosen uniform from symmetric winning configurations. Therefore, if Controller wins the m-population game then he has a positional winning strategy which only counts the number of agents in each state of $\mathcal{A}$ (the counting abstraction used in Example 1).

▶ **Proposition 2.** *Let $m \in \mathbb{N}$. If Controller wins the m-population game, then he wins the m'-population game for every $m' \leq m$.*

The idea to define $\sigma_{m'}$ is to simulate the missing $m - m'$ agents arbitrarily and apply σ_m.

Hence, when the answer to the population control problem is negative, there exists a *cut-off*, that is a value $M \in \mathbb{N}$ such that for every $m < M$, Controller has a winning strategy in $\mathcal{A}^m$, and for every $m \geq M$, he has no winning strategy.

▶ **Example 3.** To illustrate the notion of cut-off, consider the NFA on alphabet $A \cup \{b\}$ from Figure 2. Unspecified transitions lead to a sink state.

The cut-off is $M = O(|Q|)$ in this case. Indeed, we have the following two directions:

On the one hand, for $m < M$, there is a winning strategy σ_m in $\mathcal{A}^m$ to reach f^m, in just two steps. It first plays b, and because $m < M$, in the next configuration, there is at least one state q_i such that no agent is in q_i. It then suffices to play a_i to win.

Now, if $m \geq M$, there is no winning strategy to synchronize in f, since after the first b, agents can be spread so that there is at least one agent in each state q_i. From there, Controller can either play action b and restart the whole game, or play any action a_i, leading at least one agent to the sink state.

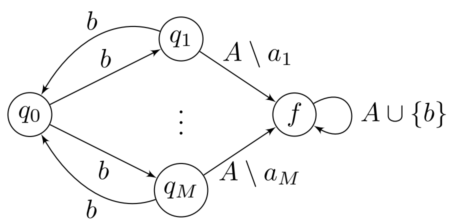


Figure 2 Illustration of the cut-off.

2.3 Main results

Our main result is the decidability of the population control problem:

▶ **Theorem 4.** *The population control problem is* EXPTIME*-complete.*

When the answer to the population control problem is positive, there exists a symbolic strategy σ, applicable to all instances $\mathcal{A}^m$, that does not need to count the number of agents in each state. This symbolic strategy requires exponential memory. Otherwise, the cut-off is at most doubly exponential, which is asymptotically tight.

▶ **Theorem 5.** *In case the answer to the population control problem is negative, the cut-off is at most* $\leq 2^{2^{O(|Q|^4)}}$. *There is a family of NFA* $(\mathcal{A}_n)$ *of size* $O(n)$ *and whose cut-off is* 2^{2^n}.

3 The capacity game

The objective of this section is to show that the population control problem is equivalent to solving a game called the *capacity game*. To introduce useful notations, we first recall the population game with infinitely many agents, as studied in [21] (see also [22] p.81).

3.1 The ∞-population game

To study the ∞-population game, the behaviour of infinitely many agents is abstracted into *supports* which keep track of the set of states in which at least one agent is. We thus introduce the *support game*, which relies on the notion of *transfer graphs*. Formally, a transfer graph is a subset of $Q \times Q$ describing how agents are moved during one step. The domain of a transfer graph G is $\mathrm{Dom}(G) = \{q \in Q \mid \exists (q,r) \in G\}$ and its image is $\mathrm{Im}(G) = \{r \in Q \mid \exists (q,r) \in G\}$. Given an NFA $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ and $a \in \Sigma$, the transfer graph G is compatible with a if for every edge (q,r) of G, $(q,a,r) \in \Delta$. We write $\mathcal{G}$ for the set of transfer graphs.

The *support game* of an NFA $\mathcal{A}$ is a two-player reachability game played by Controller and Agents on the *support arena* as follows. States are supports, *i.e.*, non-empty subsets of Q and the play starts in $\{q_0\}$. The goal support is $\{f\}$. From a support S, first Controller chooses a letter $a \in \Sigma$, then Agents chooses a transfer graph G compatible with a and such that $\mathrm{Dom}(G) = S$, and the next support is $\mathrm{Im}(G)$. A play in the support arena is described by the sequence $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ of supports and actions (letters and transfer graphs) of the players. Here, Agents best strategy is to play the maximal graph possible (this is not the case with discrete populations), and we obtain a PSPACE-complete algorithm [21]:

▶ **Proposition 6.** *Controller wins the* ∞*-population game iff he wins the support game.*

3.2 Realisable plays

If Controller wins the ∞-population game, then he wins every m-population game. Thus, if Controller wins the support game, then the answer to the population control problem is positive. The converse however is not true as demonstrated by the example from Figure 1: As we already shown, Controller wins any m-population game (with $m < \infty$). However, he loses the ∞-population game. Indeed, when Controller plays $(\delta \cdot (a \vee b))^*$, Agents has a counterstrategy which is to always split agents from q_0 to both q_1 and q_2. In this way, the sequence of supports is $\{q_0\}\{q_1, q_2\}(\{q_0, f\}\{q_1, q_2, f\})^*$, which never hits $\{f\}$.

In general, every play of the m-population game (for $m < \infty$) can be abstracted into a play in the support arena via the projection mapping $\Phi_m : Q^m \to 2^Q$ which associates to a configuration its support: $\Phi_m(\mathbf{q}) = \{q \in Q \mid \exists 1 \le i \le m, \mathbf{q}[i] = q\}$. However, as shown by the above example, the converse is not true: not every play in the support arena is *realisable*.

▶ **Definition 7** (Realisable plays)**.** A play of the support game is *realisable* if it is the projection by Φ_m of a play in an m-population game, for some $m < \infty$.

3.3 The capacity game

An obvious hint to obtain a game on the support arena equivalent with the population control problem is to make the winning condition tougher for Agents, letting him lose whenever the play is not realisable. We characterise realisability in terms of capacity:

▶ **Definition 8** (Plays with finite and bounded capacity)**.** Let $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ be a play in the support arena. An *accumulator* of ρ is a sequence $T = (T_j)_{j\in\mathbb{N}}$ such that for every integer j, $T_j \subseteq S_j$, and which is *successor-closed i.e.*, for every $j \in \mathbb{N}$,

$$(s \in T_j \wedge (s,t) \in G_{j+1}) \implies t \in T_{j+1} \quad .$$

For every $j \in \mathbb{N}$, an edge $(s,t) \in G_{j+1}$ is an *entry* to T if $s \notin T_j$ and $t \in T_{j+1}$.

A play has *finite capacity* if all its accumulators have finitely many entries, and it has *bounded capacity* if moreover the number of entries of its accumulators is bounded.

Figure 3 represents an NFA, two transfer graphs G and H, and a play $GHG^2HG^3\cdots$. Obviously, this play is not realisable because at least n agents are needed to realise n transfer graphs G in a row: at each G step, at least one agent moves from q_0 to q_1, and no new agent enters q_0. A simple analysis shows that there are only two kinds of non-trivial accumulators $(T_j)_{j\in\mathbb{N}}$ depending on whether their first non-empty T_j is $\{q_0\}$ or $\{q_1\}$. We call these top and bottom accumulators, respectively. All accumulators have finitely many entries, thus the play has finite capacity. However, for every $n \in \mathbb{N}$ there is a bottom accumulator with $2n$ entries. As an example, a bottom accumulator with 4 entries (in red) is depicted on the figure. Therefore, the capacity of this play is not bounded.

Bounded capacity is equivalent to realisability:

▶ **Lemma 9.** *A play is realisable iff it has bounded capacity.*

Bounded capacity does not seem to be a regular property. Even if it were decidable, it is likely to be of high complexity. On the other hand, the languages of plays with infinite capacity can be specified by a non-deterministic Büchi automaton: it suffices to guess on-the-fly an accumulator, and check that it has infinitely many entries. We thus define the capacity game by relaxing realisability into the finite capacity property.

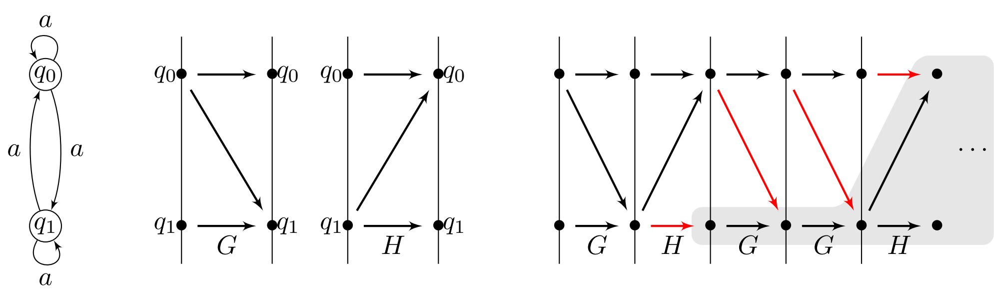


Figure 3 An NFA, two transfer graphs, and a play with finite yet unbounded capacity.

▶ **Definition 10** (Capacity game). The *capacity game* is the game played on the support arena, where Controller wins iff either the play reaches $\{f\}$ or it does not have finite capacity.

Assuming that Controller wins the capacity game, then he also wins the m-population game for all $m < \infty$. Indeed, for $m < \infty$, it suffices for Controller to play his winning strategy from the capacity game in the m-population game. By doing so, for any infinite play, its projection is realisable, thus it has bounded capacity, and in particular it has finite capacity. Because it is winning, it must reach $\{f\}$. We show that the converse implication holds as well, provided that Agents plays with finite memory.

▶ **Proposition 11.** *If Agents has a winning strategy* with finite memory M *in the capacity game, then he has a winning strategy in the* $|Q|^{1+|\mathsf{M}|\cdot 4^{|Q|}}$*-population game.*

3.4 Finite memory strategies are sufficient, not positional strategies.

In this subsection, we prove that memoryless strategies are not sufficient to win the capacity game, but finite memory strategies are sufficient. Consider the example of Figure 4, where the only way for Controller to win is to reach a support without q_2 and play c. With a memoryless strategy, Controller cannot win the capacity game. There are only two memoryless strategies from support $S = \{q_1, q_2, q_3, q_4\}$. If Controller only plays a from S, the support remains S and the play has bounded capacity. If he only plays b's from S, then Agents can split tokens from q_3 to both q_2, q_4 and the play remains in support S, with bounded capacity. In both cases, the play has finite capacity (even bounded) and Controller loses.

However, Controller can win the capacity game. His winning (finite-memory) strategy σ consists in first playing c, and then playing alternatively a and b, until the support does not contain $\{q_2\}$, in which case he plays c to win. Two consecutive steps ab send q_2 to q_1, q_1 to q_3, q_3 to q_3, and q_4 to either q_4 or q_2. To prevent Controller from playing c and win, Agents needs to spread from q_4 to both q_4 and q_2 every time ab is played. Consider the accumulator T defined by $T_{2i} = \{q_1, q_2, q_3\}$ and $T_{2i-1} = \{q_1, q_2, q_4\}$ for every $i > 0$. It has an infinite number of entries (from q_4 to T_{2i}). Hence Controller wins if this play is executed. Else, Agents eventually keeps all agents from q_4 in q_4 when ab is played, implying the next support does not contain q_2. Strategy σ is thus a winning strategy for Controller.

Finite memory is however sufficient in general to win the capacity game. Indeed, the Büchi

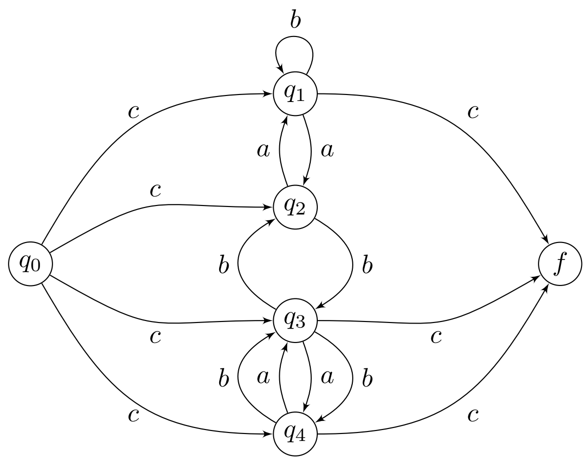


Figure 4 Population game where Controller needs memory to win the associated capacity game.

automaton accepting plays of infinite capacity can be determinised into parity automata (*e.g.* using Safra's construction). Since positional strategies are sufficient for both players to win parity games, finite memory strategies are sufficient to win capacity games, in particular for Agents. Thus we can apply Proposition 11 (and the discussing before it) to obtain:

▶ **Proposition 12.** *Controller wins the capacity game iff he wins the population control game.*

The usual Safra determinisation on the capacity game produces parity automata doubly exponential in the size of the input NFA. This allows us to solve the capacity game in 2EXPTIME and to obtain a triply exponential upper bound on the cut-off using Proposition 11 (the memory M being doubly exponential). We do not detail this construction as we provide an exponentially smaller construction in the next section.

4 Solving the capacity game in EXPTIME

To solve efficiently the capacity game, we build an equivalent exponential size parity game with a polynomial number of parities. To do so, we enrich the support arena with a *tracking list* responsible to check whether the play has finite capacity. The tracking list is a list of transfer graphs, which are used to detect certain patterns called *leaks*.

4.1 Leaking graphs

In order to detect whether a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \dots$ has finite capacity, it is enough to detect *leaking* graphs (characterising entries of accumulators). Further, leaking graphs have special *separation* properties which will allow us to track a small number of graphs. For G, H two graphs, we denote $(a, b) \in G \cdot H$ iff there exists z with $(a, z) \in G$, and $(z, b) \in H$.

▶ **Definition 13** (Leaks and separations)**.** Let G, H be two transfer graphs. We say that G *leaks at* H if there exist states q, x, y with $(q, y) \in G \cdot H$, $(x, y) \in H$ and $(q, x) \notin G$. We say that G *separates* a pair of states (r, t) if there exists $q \in Q$ with $(q, r) \in G$ and $(q, t) \notin G$.

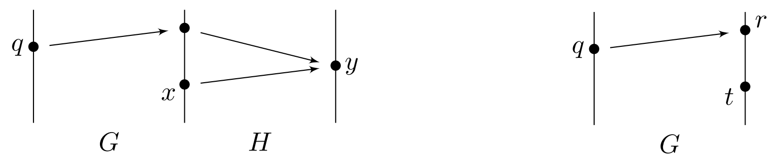


Figure 5 Left: G leaks at H; Right: G separates (r, t).

The tracking list will be composed of concatenated graphs *tracking* i of the form $G[i, j] = G_{i+1} \cdots G_j$ relating S_i with S_j: $(s_i, s_j) \in G[i, j]$ if there exists $(s_k)_{i<k<j}$ with $(s_k, s_{k+1}) \in G_{k+1}$ for all $i \leq k \leq j$. Infinite capacity relates to leaks in the following way:

▶ **Lemma 14.** *A play has infinite capacity iff there exists an index i such that $G[i, j]$ leaks at G_{j+1} for infinitely many indices j.*

In this case, we say that index i *leaks infinitely often*. Note that if G separates (r, t), and r, t have a common successor by H, then G leaks at H. To link leaks with separations, we consider for each index k, the pairs of states that have a common successor, in possibly several steps, as expressed by the symmetric relation R_k: $(r, t) \in R_k$ iff there exists $j \geq k$ and $y \in Q$ such that $(r, y) \in G[k, j] \wedge (t, y) \in G[k, j]$.

▶ **Lemma 15.** *For $i < n$ two indices, the following three properties hold:*

1. *If $G[i,n]$ separates $(r,t) \in R_n$, then there exists $m \geq n$ such that $G[i,m]$ leaks at G_{m+1}.*
2. *If index i does not leak infinitely often, then the number of indices j such that $G[i,j]$ separates some $(r,t) \in R_j$ is finite.*
3. *If index i leaks infinitely often, then for all $j > i$, $G[i,j]$ separates some $(r,t) \in R_j$.*

4.2 The tracking list

The *tracking list* exploits the relationship between leaks and separations. It is a list of transfer graphs which altogether separate all possible pairs, and are sufficient to detect when leaks occur. Notice that telling at step j whether the pair (r,t) belongs to R_j cannot be performed by a deterministic automaton. We thus *a priori* have to consider every pair $(r,t) \in Q^2$ for separation. The tracking list $\mathcal{L}_n$ at step n is defined inductively as follows. $\mathcal{L}_0$ is the empty list, and for $n > 0$, the list $\mathcal{L}_n$ is computed in three stages:

1. first, every graph H in the list $\mathcal{L}_{n-1}$ is concatenated with G_n, yielding $H \cdot G_n$;
2. second, G_n is added at the end of the obtained list;
3. last, the list is filtered: a graph H is kept if and only if it separates a pair of states $(p,q) \in Q^2$ which is not separated by any graph that appears earlier in the list.

Because of the third item, there are at most $|Q|^2$ graphs in the tracking list. The list may become empty if no pair of states is separated by any graph, for example if all the graphs are complete. Let $\mathcal{L}_n = \{H_1, \cdots, H_\ell\}$ be the tracking list at step n. Then each transfer graph $H_r \in \mathcal{L}_n$ is of the form $H_r = G[t_r, n]$. We say that r is the *level* of H_r, and t_r the *index tracked* by H_r. Observe that the lower the level of a graph in the list, the smaller the index it tracks. When we consider the sequence of tracking lists $(\mathcal{L}_n)_{n\in\mathbb{N}}$, for every index i, either it eventually stops to be tracked or it is tracked forever from step i, *i.e.* for every $n \geq i$, $G[i,n]$ belongs to $\mathcal{L}_n$. In the latter case, i is said to be *remanent* (because it will never disappear).

Using Lemma 14 and the second and third statements of Lemma 15, we obtain:

▶ **Lemma 16.** *A play has infinite capacity iff there exists an index i such that i is remanent and leaks infinitely often.*

4.3 The parity game

We now describe a parity game $\mathcal{PG}$, which extends the support arena with on-the-fly computation of the tracking list.

Priorities. By convention, lowest priorities are the most important and the odd parity is good for Controller, so Controller wins iff the $\liminf$ of the priorities is odd. With each level $1 \leq r \leq |Q|^2$ of the tracking list are associated two priorities $2r$ and $2r+1$, and on top of that are added priorities 1 and $2|Q|^2+2$, hence the set of all priorities is $\{1, \ldots, 2|Q|^2+2\}$.

When Agents chooses a transition labelled by a transfer graph G, the tracking list is updated with G and the priority of the transition is determined as the smallest among: priority 1 if a support in $\{f\}$ has ever been visited, priority $2r+1$ for the smallest r such that H_r (from level r) leaks at G, priority $2r$ for the smallest level r where a graph was removed, and in all other cases priority $2|Q|^2+2$.

States and transitions. $\mathcal{G}^{\leq|Q|^2}$ denotes the set of list of at most $|Q|^2$ transfer graphs.

- States of $\mathcal{PG}$ form a subset of $\{0,1\} \times 2^Q \times \mathcal{G}^{\leq |Q|^2}$, each state being of the form $(b, S, H_1, \ldots, H_\ell)$ with $b \in \{0,1\}$ a bit indicating whether a support in $\{f\}$ has been seen, S the current support and $(H_1, \ldots, H_\ell)$ the tracking list. The initial state is $(0, \{q_0\}, \emptyset)$.
- Transitions in $\mathcal{PG}$ are all $(b, S, H_1, \ldots, H_\ell) \xrightarrow{\mathbf{p},a,G} (b', S', H'_1, \ldots, H'_{\ell'})$ where $\mathbf{p}$ is the priority, and such that $S \xrightarrow{a,G} S'$ is a transition of the support arena, and
  1. $(H'_1, \ldots, H'_{\ell'})$ is the tracking list obtained by updating the tracking list $(H_1, \ldots, H_\ell)$ with G, as explained in subsection 4.2;
  2. if $b = 1$ or if $S' \subseteq F$, then $\mathbf{p} = 1$ and $b' = 1$;
  3. otherwise $b' = 0$. In order to compute the priority $\mathbf{p}$, we let $\mathbf{p}'$ be the smallest level $1 \leq r \leq \ell$ such that H_r leaks at G and $\mathbf{p}' = \ell + 1$ if there is no such level, and we also let $\mathbf{p}''$ as the minimal level $1 \leq r \leq \ell$ such that $H'_r \neq H_r \cdot G$ and $\mathbf{p}'' = \ell + 1$ if there is no such level. Then $\mathbf{p} = \min(2\mathbf{p}' + 1, 2\mathbf{p}'')$.

We are ready to state the main result of this paper, which yields an EXPTIME complexity for the population control problem. This entails the first statement of Theorem 4, and together with Proposition 11, also the first statement of Theorem 5.

▶ **Theorem 17.** *Controller wins the game $\mathcal{PG}$ if and only if Controller wins the capacity game. Solving these games can be done in time $O(2^{(1+|Q|+|Q|^4)(2|Q|^2+2)})$. Strategies with $2^{|Q|^4}$ memory states are sufficient to both Controller and Agents.*

Proof. The state space of parity game $\mathcal{PG}$ is the product of the set of supports with a deterministic automaton computing the tracking list. As the state space of the capacity game is also the set of supports, there is a natural correspondence between plays and strategies in the parity game $\mathcal{PG}$ and in the capacity game.

Controller can win the parity game $\mathcal{PG}$ in two ways: either the play visits the support $\{f\}$, or the priority of the play is $2r + 1$ for some level $1 \leq r \leq |Q|^2$. By design of $\mathcal{PG}$, this second possibility occurs iff r is remanent and leaks infinitely often. According to Lemma 16, this occurs if and only if the corresponding play of the capacity game has infinite capacity. Thus Controller wins $\mathcal{PG}$ iff he wins the capacity game.

In the parity game $\mathcal{PG}$, there are at most $2^{1+|Q|}\left(2^{|Q|^2}\right)^{|Q|^2} = 2^{1+|Q|+|Q|^4}$ states and $2|Q|^2 + 2$ priorities, implying the complexity bound using state-of-the-art algorithms [18]. Actually the complexity is even pseudo-polynomial according to the algorithms in [10]. Notice however that this has little impact on the complexity of the population control problem, as the number of priorities is logarithmic in the number of states of our parity game.

Further, it is well known that the winner of a parity game has a positional winning strategy [18]. A *positional* winning strategy σ in the game $\mathcal{PG}$ corresponds to a *finite-memory* winning strategy σ' in the capacity game, whose memory states are the states of $\mathcal{PG}$. Actually in order to play σ', it is enough to remember the tracking list, *i.e.* the third component of the state space of $\mathcal{PG}$. Indeed, the second component, in 2^Q, is redundant with the actual state of the capacity game and the bit in the first component is set to 1 when the play visits $\{f\}$ but in this case the capacity game is won by Controller whatever is played afterwards. Since there at most $2^{|Q|^4}$ different tracking lists, we get the upper bound on the memory. ◀

5 Lower bounds

The proofs of Theorems 4 and 5 are concluded by the proofs of lower bounds.

▶ **Theorem 18.** *The population control problem is* EXPTIME*-hard.*

Proof. We first prove a PSPACE-hard lower bound, reducing from the halting problem for polynomial space Turing machines. We will extend the result to an EXPTIME-hard lower bound, by reducing from the halting problem for polynomial space *alternating* Turing machines. Let $\mathcal{M} = (S, \Gamma, T, s_0, s_f)$ be a Turing machine with $\Gamma = \{0, 1\}$ as tape alphabet. By assumption, there exists a polynomial P such that, on initial configuration $x = x_1 \cdots x_n \in \{0, 1\}^n$, $\mathcal{M}$ uses at most $P(n)$ tape cells. We write transitions $t \in T$ under the form $t = (s, s', b, b', d)$, where s and s' are, respectively, the source and the target control states, b and b' are, respectively, the symbols read from and written on the tape, and $d \in \{\leftarrow, \rightarrow, -\}$ indicates the move of the tape head.

We derive an NFA $\mathcal{A} = (Q, \Sigma, q_0, \Delta)$ with a distinguished state ☺ such that, $\mathcal{M}$ terminates in q_f on input x if anf only if $(\mathcal{A}, ☺)$ is a positive instance of the population control problem. The high-level description of $\mathcal{A}$ is as follows: The states of $\mathcal{A}$ are of several types: contents of each of the $P(n)$ cells (one state per content and per cell), position of the tape head (one state per possible position), control state of the Turing machine (one state per control state), and three special states, namely an initial state q_0, a sink winning state ☺, and a sink losing state ☹. To each transition t in the Turing machine and each position p of the tape, we associate an action $a_{t,p}$ in the NFA. Intuitively, $a_{t,p}$ moves tokens from the source state to the target state, update the tape head position, as well as the current cell contents at positions p. Playing $a_{t,p}$ from head position $q \neq p$ sends tokens from q to ☹, ensuring that Controller only plays actions associated with the current head position, and similarly for the cell contents b at positions p and state s expected by t. Last, an *init* action is available from q_0 to encode the initial configuration x.

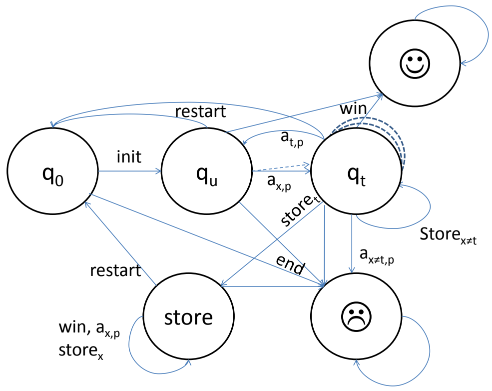
The NFA also has winning actions, that allow one to check that there are no tokens in a subset of states, and send the remaining one to the target. One such action should be played when tokens encoding the state of the Turing machine lie in q_f, indicating that $\mathcal{M}$ accepted. Another winning action is played whenever there are not enough tokens to encode the initial configuration: Agents needs at least $P(n) + 2$ tokens to place in the tape configuration ($P(n)$ tokens), the control state and the head position (one token each). The sink losing state is used to pinpoint an error by Controller in the simulation of the Turing machine.

In order to encode an alternating Turing machine, we use an additional state q_u, with an additional transition labelled *init* from q_0 to q_u. We assume in a first step that q_u is filled with a unique agent. We will explain later how to simulate this.

We add one state q_t per transition t of the Turing machine. We assume that the states of the Turing machine alternate between states of Controller (first) and then states of Agents. From state q_u, Agents can choose to place the agent to any of the state q_t associated with a transition t of the Turing machine, whatever action is chosen by Controller. From state q_t, only actions $a_{t,p}$ are allowed, bringing back the token to q_u. That is, actions $a_{t',p}$ with $t' \neq t$ leads from q_t to the sink state ☹. Hence, Controller needs to follow the transition t chosen by Agents. To punish Agents in the case where the tape content is not the one expected by transition $t = (s, s', b, b', d)$, there are trashing actions $trash_s$ and $trash_{p,b}$ allowed from state q_t. Action $trash_s$ sends the token from q_t to ☺, and tokens from $control = s$ to ☹. In this way, the token from q_t cannot be used by Agents and Controller can win more easily. Similarly, $trash_{p,b}$ leads to ☺ the token from q_t, and leads to ☹ tokens from a position state $\neq p$ and from the content b at position p. In this way, Agents has to place the token only in a q_t which agrees with the configuration.

We add one action *end* which sends all tokens from state ☺, q_u and any of the q_t to the target state ☺, and which sends any other state (in particular the one encoding the Turing Machine configuration) to ☹. In this way, assuming that there is a single token in q_u after the first transition, Controller can choose the transition from a Controller state of the Turing machine, and Agents can chose the transition from an Agents state, and we have reduced from an alternating polynomial space Turing machine, yielding the EXPTIME-hardness.

Now, we add another gadget depicted on the figure below to adapt to cases where there are more than one token in q_u, and Agents would split tokens to several q_t. We add a state *store* and actions $(store_t)_{t \text{ a transition}}$ and *restart* which can be played by Controller at any time. Action $store_t$ keeps all tokens where they are but tokens in q_t which are sent to state *store*. Also, action *restart* sends all tokens but those in ☺ and ☹ to state q_0. Last, the winning action as well as *end* are changed: the winning action sends tokens from q_u or any q_t to ☺, tokens from the winning Turing machine configuration stay where they are and tokens from other Turing machine configurations goes to ☹. Action *end* sends tokens from the Turing machine configuration (and from ☺) to ☺, and all tokens from q_0, q_u, q_t and *store* to ☹.



Assume that Agents has a winning strategy in the Turing machine and the number of tokens is at least $P(n) + 3$. He can thus play his winning strategy placing a single token in q_u at start. If Controller plays $store_t$ (whatever t), either no tokens is stored, or the unique token is stored in *store*. Thus Controller cannot play *end* and thus cannot reach ☺ with the tokens from the Turing machine configuration until it plays a *restart*, which places all the tokens back to q_0 (as no token reached ☺). Hence playing $store_t$ is useless and Agents wins.

Conversely, if Agents has no winning strategy in the Turing machine, then to win, he would need to have more than one token in q_u and split them at some point between $q_{t_1}, \ldots, q_{t_n}$. Then, Controller can play the associated $store_{t_2}, \ldots, store_{t_n}$ actions placing most tokens (but from q_{t_1}) in *store*, and he plays his winning strategy from q_{t_1} which places some tokens in ☺. Then Controller plays *restart* and proceed inductively with strictly less tokens from q_0, as some tokens are safely in ☺. Eventually, he can play *end* and he wins. ◀

We now show that surprisingly, the cut-off can be as high as doubly exponential in the size of the NFA.

▶ **Proposition 19.** *There exists a family of NFA $(\mathcal{A}_n)_{n\in\mathbb{N}}$ such that $|\mathcal{A}_n| = 2n + 7$, and for $M = 2^{2^n+1} + n$, there is no winning strategy in $\mathcal{A}_n^M$ and there is one in $\mathcal{A}_n^{M-1}$.*

Proof. Let $n \in \mathbb{N}$. The NFA $\mathcal{A}_n$ we build is the product of *two* NFAs with different properties: $\mathcal{A}_n = \mathcal{A}_{\mathsf{split}} \times \mathcal{A}_{\mathsf{count},n}$. On the one hand, for $\mathcal{A}_{\mathsf{split}}$, winning the game with m agents requires $\Theta(\log m)$ steps. On the other hand, $\mathcal{A}_{\mathsf{count},n}$ implements a usual counter over n bits (as used in many different publications), such that Controller can avoid to lose during $O(2^n)$ steps. The combination of these two gadgets ensures a cut-off for $\mathcal{A}_n$ of 2^{2^n}.

Figure 6 in the appendix shows the counting gadget that implements a counter with states l_i (meaning bit i is 0) and h_i (for bit i is 1). It enjoys the following properties: $(c1)$ there is a strategy in $\mathcal{A}_{\mathsf{count},n}$ to ensure avoiding ☹ during 2^n steps, by playing α_i whenever the counter suffix from bit i is $01\cdots1$; $(c2)$ for $m \geq n$, no strategy of $\mathcal{A}^m_{\mathsf{count},n}$ avoid ☹ for 2^n steps.

Recall Figure 1, which presents the splitting gadget that has the following properties. In $\mathcal{A}^m_{\mathsf{split}}$ with $m \in \mathbb{N}$ agents, $(s1)$ there is a winning strategy ensuring to win in $2\lfloor \log_2 m \rfloor + 2$ steps; $(s2)$ no strategy can ensure to win in less than $2\lfloor \log_2 m \rfloor + 1$ steps.

The two gadgets (splitting and counting) are combined into their product $\mathcal{A}_n = \mathcal{A}_{\mathsf{split}} \times \mathcal{A}_{\mathsf{count},n}$, with actions consisting of pairs of action, one for each gadget: $\Sigma = \{a, b, \delta\} \times \{\alpha_i \mid 1 \leq i \leq n\}$. The initial state is the q_0 of $\mathcal{A}_{\mathsf{count},n}$, and we add transitions labelled $\alpha_1, \ldots, \alpha_n$ from this initial state to the q_0 of $\mathcal{A}_{\mathsf{split}}$. We add an action $*$ which can be played from any state of $\mathcal{A}_{\mathsf{count},n}$ but ☹, and only from f in $\mathcal{A}_{\mathsf{split}}$, leading to the global target state ☺.

Let $M = 2^{2^n+1} + n$. We deduce that the cut-off is $M - 1$ as follows:

- For M agents, a winning strategy for Agents is to first split n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, in order to fill each l_i with 1 token, and 2^{2^n+1} tokens to the q_0 of $\mathcal{A}_{\mathsf{split}}$. Then Agents splits evenly tokens between q_1, q_2 in $\mathcal{A}_{\mathsf{split}}$. In this way, Controller needs at least $2^n + 1$ steps to reach the final state of $\mathcal{A}_{\mathsf{split}}$ $(s2)$, but Controller reaches ☹ after these $2^n + 1$ steps in $\mathcal{A}_{\mathsf{count},n}$ $(c2)$.
- For $M - 1$ agents, Agents needs to use at least n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, else Controller can win easily. But then there are less than 2^{2^n+1} tokens in the q_0 of $\mathcal{A}_{\mathsf{split}}$. And thus by $(s1)$, Controller can reach f within 2^n steps, after which he still avoids ☹ in $\mathcal{A}_{\mathsf{count},n}$ $(c1)$. And then Controller sends all agents to ☺ using $*$.

Thus, the family $(\mathcal{A}_n)$ of NFA exhibits a doubly exponential cut-off. ◀

6 Discussion

Obtaining an EXPTIME algorithm for the control problem of a population of agents was challenging. We have a matching EXPTIME-hard lower-bound. Further, the surprising doubly exponential matching upper and lower bounds on the cut-off imply that the alternative technique, checking that Controller wins all m-population game for m up to the cut-off, is far from being efficient. This compares favourably with the exponential gap for the parameterized verification of almost-sure reachability for a population communicating via a shared register [8] (the latter problem is in EXPSPACE and PSPACE-hard).

The idealised formalism we describe in this paper is not entirely satisfactory: for instance, while each agent can move in a non-deterministic way, unrealistic behaviours can happen, *e.g.* all agents synchronously taking infinitely often the same choice. An almost-sure control problem in a probabilistic formalism should be studied, ruling out such extreme behaviours. As the population is discrete, we may avoid the undecidability that holds for distributions [11] and is inherited from the equivalence with probabilistic automata [17]. Abstracting continuous distributions by a discrete population of arbitrary size could thus be seen as an approximation technique for undecidable formalisms such as probabilistic automata.

Acknowledgement: We would like to thank Gregory Batt for fruitful discussions concerning the biological setting. We thanks also Mahsa Shirmohammadi for interesting discussions. This work was partially supported by ANR projet STOCH-MC (ANR-13-BS02-0011-01).

References

1 Parosh Abdulla, Giorgio Delzanno, Othmane Rezine, Arnaud Sangnier, and Riccardo Traverso. On the verification of timed ad hoc networks. In *Proceedings of Formats'11*, volume 6919 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2011.

2 Parosh Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–263, 2003.

3 S. Akshay, Blaise Genest, Bruno Karelovic, and Nikhil Vyas. On regularity of unary probabilistic automata. In *Proceedings of STACS'16*, volume 47 of *Leibniz International Proceedings in Informatics*, pages 8:1–8:14. Leibniz-Zentrum für Informatik, 2016.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of PODC'04*, pages 290–299. ACM, 2004.

5 André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 1(303):7–34, 2003.

6 Nathalie Bertrand and Paulin Fournier. Parameterized verification of many identical probabilistic timed processes. In *Proceedings of FSTTCS'13*, volume 24 of *Leibniz International Proceedings in Informatics*, pages 501–513. Leibniz-Zentrum für Informatik, 2013.

7 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Playing with probabilities in reconfigurable broadcast networks. In *Proceedings of FoSSaCS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2014.

8 Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. In *Proceedings of ICALP'16*, volume 55 of *Leibniz International Proceedings in Informatics*, pages 106:1–106:14. Leibniz-Zentrum für Informatik, 2016.

9 Tomás Brázdil, Petr Jančar, and Antonín Kučera. Reachability games on extended vector addition systems with states. In *Proceedings of ICALP'10*, volume 6199 of *Lecture Notes in Computer Science*, pages 478–489. Springer, 2010.

10 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of STOCS'17*. ACM, 2017.

11 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata (erratum). Technical report, CoRR abs/1206.0995, 2012.

12 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Limit synchronization in Markov decision processes. In *Proceedings of FoSSaCS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2014.

13 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of STACS'14*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, 2014.

14 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proceedings of LICS'99*, pages 352–359. IEEE Computer Society, 1999.

15 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. In *Proceedings of CONCUR'15*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 470–482. Leibniz-Zentrum für Informatik, 2015.

16 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

17 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *Proceedings of ICALP'10*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010.

18 Marcin Jurdzinski. Small progress measures for solving parity games. In *Proceedings of STACS'00*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

19 Marcin Jurdziński, Ranko Lazić, and Sylvain Schmitz. Fixed-dimensional energy games are in pseudo polynomial time. In *Proceedings of ICALP'15*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015.

20 Panagiotis Kouvaros and Alessio Lomuscio. Parameterised Model Checking for Alternating-Time Temporal Logic. In *Proceedings of ECAI'16*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1230–1238. IOS Press, 2016.

21 Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304, 2014.

22 Mahsa Shirmohammadi. *Qualitative analysis of synchronizing probabilistic systems.* PhD thesis, ULB, 2014.

23 Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt. In silico control of biomolecular processes. In *Computational Methods in Synthetic Biology*, chapter 13, pages 277–285. Humana Press, Springer, 2015.

24 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In *Proceedings of LATA'08*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008.

A Appendix

This Appendix contains proofs ommitted in the core of the paper due to space contraints.

A.1 Proof of Section 2

▶ **Proposition 2.** *Let $m \in \mathbb{N}$. If Controller wins the m-population game, then he wins the m'-population game for every $m' \leq m$.*

Proof. Let $m \in \mathbb{N}$, and assume σ is a winning strategy for Controller in $\mathcal{A}^m$. For $m' \leq m$ we define σ' as a strategy on $\mathcal{A}^{m'}$, inductively on the length of finite plays. Initially, σ' chooses the same first action as σ: $\sigma'(q_0^{m'}) = \sigma(q_0^m)$. We then arbitrarily choose that the missing $m-m'$ agents would behave similarly as the first agent. This is indeed a possible move for the adversary in $\mathcal{A}^m$. Then, for any finite play under σ' in $\mathcal{A}^{m'}$, say $\pi' = \mathbf{q}_0^{m'} a_0 \mathbf{q}_1^{m'} a_1 \mathbf{q}_2^{m'} \cdots \mathbf{q}_n^{m'}$, there must exist an extension π of π' obtained by adding $m - m'$ agents, all behaving as the first agent in $\mathcal{A}^{m'}$, that is consistent with σ. Then, we let $\sigma'(\pi') = \sigma(\pi)$. Obviously, since σ is winning in $\mathcal{A}^m$, σ' is also winning in $\mathcal{A}^{m'}$. ◀

A.2 Proofs of Section 3

▶ **Lemma 9.** *A play is realisable iff it has bounded capacity.*

Proof. Let $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \cdots$ be a realisable play in the support arena and $\pi = \mathbf{q}_0\mathbf{q}_1\mathbf{q}_2\cdots$ a play in the m-population game for some m, such that $\Phi_m(\pi) = \rho$. For any accumulator $T = (T_j)_{j\in\mathbb{N}}$ accumulator of ρ, let us show that T has less than m entries. For every $j \in \mathbb{N}$, we define $n_j =\mid \{1 \leq k \leq m \mid \mathbf{q}_j(k) \in T_j\} \mid$ as the number of agents in the accumulator at index j. By definition of the projection, every edge (s,t) in G_j corresponds to the move of at least one agent from state s in $\mathbf{q}_j$ to state t in $\mathbf{q}_{j+1}$. Thus, since the accumulator is successor-closed, the sequence $(n_j)_{j\in\mathbb{N}}$ is non-decreasing and it increases at each index j where the accumulator has an entry. The number of entries is thus bounded by m the number of agents.

Conversely, assume that a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \cdots$ has bounded capacity, and let m be an upper bound on the number of entries of its accumulators. Let us show that ρ is the projection of a play $\pi = \mathbf{q}_0\mathbf{q}_1\mathbf{q}_2\cdots$ in the $(|S_0||Q|^{m+1})$-population game. In the initial configuration $\mathbf{q}_0$, every state in S_0 contains $|Q|^{m+1}$ agents. Then, configuration $\mathbf{q}_{n+1}$ is obtained from $\mathbf{q}_n$ by splitting evenly the agents among all edges of G_{n+1}. As a consequence, for every edge $(s,t) \in G_{n+1}$ at least a fraction $\frac{1}{|Q|}$ of the agents in state s in $\mathbf{q}_n$ moves to state t in $\mathbf{q}_{n+1}$. By induction, $\pi = \mathbf{q}_0\mathbf{q}_1\mathbf{q}_2\cdots$ projects to some play $\rho' = S'_0 \xrightarrow{a_1,G'_1} S'_1 \xrightarrow{a_2,G'_2} \cdots$ such that for every $n \in \mathbb{N}$, $S'_n \subseteq S_n$ and $G'_n \subseteq G_n$. To prove that $\rho' = \rho$, we show that for every $n \in \mathbb{N}$ and state $t \in S_n$, at least $|Q|$ agents are in state t in $\mathbf{q}_n$. For that let $(U_j)_{j\in 0\ldots n}$ be the sequence of subsets of Q defined by $U_n = \{t\}$, and for $0 < j < n$,

$$U_{j-1} = \{s \in Q \mid \exists t' \in U_j, (s,t') \in G_j\} \ .$$

Let $(T_j)_{j\in\mathbb{N}}$ be the sequence of subsets of states defined by $T_j = Q \setminus U_j$ if $j \leq n$ and $T_j = Q$ otherwise. Then $(T_j)_{j\in\mathbb{N}}$ is an accumulator: if $s \notin U_j$ and $(s,s') \in G_j$ then $s' \notin U_{j+1}$. As a consequence, $(T_j)_{j\in\mathbb{N}}$ has at most m entries, thus there are at most m indices $j \in \{0\ldots n-1\}$ such that some agents in the states of U_j in configuration $\mathbf{q}_j$ may move to states outside of U_{j+1} in configuration $\mathbf{q}_{j+1}$. In other words, if we denote M_j the number of agents in the states of U_j in configuration $\mathbf{q}_j$ then there are at most m indices where the sequence

$(M_j)_{j\in 0\ldots n}$ decreases. By definition of π, even when $M_j > M_{j+1}$ at least a fraction $\frac{1}{|Q|}$ of the agents moves from U_j to U_{j+1} along the edges of G_{j+1}, thus $M_{j+1} \geq \frac{M_j}{|Q|}$. Finally, the number of agents M_n in state t in $\mathbf{q}_n$ satisfies $M_n \geq \frac{|S_0||Q|^{m+1}}{|Q|^m} \geq |Q|$. Hence ρ and ρ' coincide, so that ρ is realisable. ◀

We prove one implication of the characterization of the population control problem by capacity game.

▶ **Proposition 20.** *If Controller wins the capacity game, then for all m Controller has a winning strategy in the m-population game.*

Proof. Let σ be a strategy for Controller winning the capacity game, and σ_m the strategy in the m-population game which plays like σ: from a configuration $\mathbf{q}$ with support S the strategy σ_m plays the action $\sigma(S)$.

The strategy σ_m is winning: every play consistent with σ_m projects in the support arena to a play consistent with σ. According to Lemma 9, the projection has finite capacity. Since σ is winning the projection reaches a support in 2^F, thus the play of the m-population game reaches a configuration in F^m. ◀

We now prove the more challenging reverse implication.

▶ **Proposition 11.** *If Agents has a winning strategy* with finite memory M *in the capacity game, he has a winning strategy in the $|Q|^{1+|\mathsf{M}|\cdot 4^{|Q|}}$-population game.*

Proof. Let τ be a winning strategy for Agents in the capacity game with finite-memory M. We define and $m = |Q|^{1+|\mathsf{M}|\cdot 4^{|Q|}}$ and consider the m-population game.

A winning strategy τ_m for Agents in the m-population game can be designed using τ as follows. When it is Agents's turn to play in the m-population game, the play so far $\pi = \mathbf{q}_0 \xrightarrow{a_1} \mathbf{q}_1 \cdots \mathbf{q}_n \xrightarrow{a_{n+1}}$ is projected via Φ_m to a play $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \cdots S_n \xrightarrow{a_{n+1}}$ in the capacity game. Let $G_{n+1} = \tau(\rho)$ be the decision of Agents at this point in the capacity game. Then, to determine $\mathbf{q}_{n+1}$, τ_m splits evenly the agents in $\mathbf{q}_n$ along every edge of G_{n+1}. This guarantees that for every edge $(q,r) \in G_{n+1}$, at least a fraction $\frac{1}{|Q|}$ of the agents in state q in $\mathbf{q}_n$ moves to state r in $\mathbf{q}_{n+1}$. Assuming that τ_m is properly defined, then it is winning for Agents. Indeed, τ guarantees that $\{f\}$ is never reached in the capacity game, thus τ_m guarantees that not all agents are simultaneously in the target state f.

Now, strategy τ_m is properly defined as long as the projection ρ is consistent with τ, which in turns holds as long as at least one agent actually moves along every edge of G_{n+1}. To establish that τ_m is well-defined, it is enough to show that:

(†) for every $n \in \mathbb{N}$ and every state $r \in S_n$, at least $|Q|$ agents are in state r in $\mathbf{q}_n$.

To show (†), we consider $\rho = S_0 \xrightarrow{a_1,G_1} S_1 \xrightarrow{a_2,G_2} \ldots S_n$ the projection in the support arena of a play $\pi = \mathbf{q}_0 \xrightarrow{a_1} \mathbf{q}_1 \xrightarrow{a_2} \ldots \mathbf{q}_n$ consistent with τ_m. Let $(U_j)_{j\in 0\ldots n}$ be the sequence of subsets of Q defined by $U_n = \{r\}$, and for $0 < j < n$,

$$U_{j-1} = \{s \in Q \mid \exists t \in U_j, (s,t) \in G_j\} \ .$$

Let $T = (T_j)_{j\in\mathbb{N}}$ be the sequence of complement subsets: $T_j = Q \setminus U_j$ if $j \leq n$ and $T_j = Q$ otherwise. Then, T is an accumulator: if $s \notin U_j$ and $(s,s') \in G_j$ then $s' \notin U_{j+1}$.

Assume that there are two integers $0 \leq i < j \leq n$ such that at step i and j

- the memory state of τ coincide: $\mathsf{m}_i = \mathsf{m}_j$;

- the supports coincide: $S_i = S_j$; and
- the supports in the accumulator T coincide: $T_i = T_j$.

Then we show that there is no entry in the accumulator between indices i and j. The play π_* identical to π up to date i and which repeats ad infinitum the subplay of π between dates i and j, is consistent with τ, because $\mathsf{m}_i = \mathsf{m}_j$ and $S_i = S_j$. The corresponding sequence of transfer graphs is $G_0, \ldots, G_{i-1}(G_i, \ldots, G_{j-1})^\omega$ and $T_0, \ldots, T_{i-1}(T_i \ldots T_{j-1})^\omega$ is a "periodic" accumulator of π_*. By periodicity, this accumulator has either no entry, or infinitely many entries after date $i-1$. Since τ is winning, π_* has finite capacity, thus the periodic accumulator has no entry after date $i-1$, and there is no entry in the accumulator $(T_j)_{j\in\mathbb{N}}$ between indices i and j.

Let I be the set of indices where there is an entry in the accumulator $(T_j)_{j\in\mathbb{N}}$. According to the above, for all pairs of distinct indices (i,j) in I, we have $m_i \neq m_j \vee S_i \neq S_j \vee V_i \neq V_j$. As a consequence,

$$|I| \leq |\mathsf{M}| \cdot 4^{|Q|} \quad .$$

Denote a_i the number of agents in U_i at date i. If $i \notin I$, *i.e.* if there is no entry to T_i at date i then all agents in U_i at date i are in U_{i+1} at date $i+1$ hence $a_{i+1} = a_i$. In the other case, when $i \in I$, strategy τ_m sends at least a fraction $\frac{1}{|Q|}$ of the agents from U_i to U_{i+1} thus $a_{i+1} \geq \frac{a_i}{|Q|}$. Finally

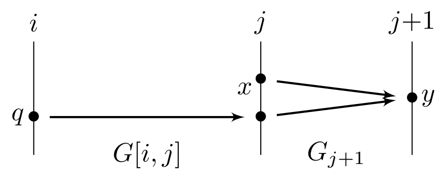
$$a_n \geq \frac{m}{|Q|^{|I|}} \geq m \cdot |Q|^{-|\mathsf{M}|\cdot 4^{|Q|}} = |Q|^{1+|\mathsf{M}|\cdot 4^{|Q|}} \cdot |Q|^{-|\mathsf{M}|\cdot 4^{|Q|}} = |Q| \quad .$$

Since $U_n = \{r\}$ then property (†) holds. As a consequence τ_m is well-defined and, as already discussed, τ_m is a winning strategy for Agents in the m-population game. ◀

Proofs of Section 4

▶ **Lemma 14.** *A play has infinite capacity iff there exists an index i such that $G[i,j]$ leaks at G_{j+1} for infinitely many indices j.*

Proof. To prove the right-to-left implication, assume that there exists an index i such that $G[i,j]$ leaks at G_{j+1} for an infinite number of indices j. As the number of states is finite, there exist a state q with an infinite number of indices j such that we have some $(x_j, y_{j+1}) \in G_{j+1}$ with $(q, y_{j+1}) \in G[i, j{+}1]$, $(q, x_j) \notin G[i,j]$. The accumulator generated by $T_i = \{q\}$ has an infinite number of entries, and we are done with this direction.



For the left-to-right implication, assume that there is an accumulator $(T_j)_{j\geq 0}$ with an infinite number of entries.

For X a subset of vertices of the DAG, $|X|_n$ denotes the number of vertices of X of rank n, and we define the *width* of X as $\text{width}(X) = \limsup_n |X|_n$. We use several times the following property of the width.

(†) If $X_0 \neq \emptyset$ and X_1 are two disjoint successor-closed sets, and if $X_0 \cup X_1 \subseteq X$, then then $\mathrm{width}(X_1) < \mathrm{width}(X)$.

Let us prove property (†). Let r be the minimal rank of vertices in X_0. Since X_0 is successor-closed and there is no dead-end in the DAG, for every $n \geq r$, X_0 contains at least one vertex of rank n. Because X_0 and X_1 are disjoint, we derive $|X_1|_n + 1 \leq |X|_n$. Taking the limsup of this inequality we obtain (†).

We pick X a successor-closed set of nodes with infinitely many incoming edges, of minimal width with this property. Let v be a vertex of X of minimal rank and denote $S(v)$ for the set of successors of v. Let us show that $S(v)$ has infinitely many incoming edges. Define $T(v)$ as the set of predecessors of vertices in $S(v)$ and $Y = X \setminus T(v)$. Then Y is successor-closed because $T(v)$ is predecessor-closed and X is successor-closed. Applying property (†) to $X_0 = S(v) \subseteq X$ and $X_1 = Y \subseteq X$, we obtain $\mathrm{width}(Y) < \mathrm{width}(X)$. By width minimality of X among successor-closed sets with infinitely many incoming edges, Y must have finitely many incoming edges only. Since $Y = X \setminus T(v)$ and X has infinitely many incoming edges, then $T(v)$ has infinitely many incoming edges. Thus there are infinitely many edges connecting a vertex outside $S(v)$ to a vertex of $S(v)$, so that $S(v)$ has infinitely many incoming edges. ◀

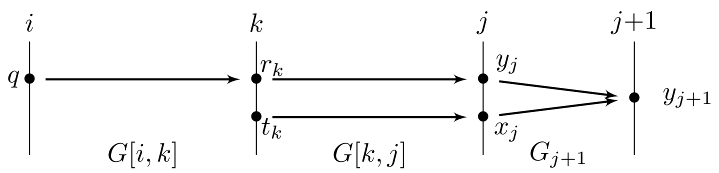
▶ **Lemma 15.** *For $i < n$ two indices, the following three properties hold:*

1. *If $G[i,n]$ separates $(r,t) \in R_n$, then there exists $m \geq n$ such that $G[i,m]$ leaks at G_{m+1}.*
2. *If index i does not leak infinitely often, then the number of indices j such that $G[i,j]$ separates some $(r,t) \in R_j$ is finite.*
3. *If index i leaks infinitely often, then for all $j > i$, $G[i,j]$ separates some $(r,t) \in R_j$.*

Proof. We start with the proof of the first item. Assume that $G[i,n]$ separates a pair $(r,t) \in R_n$. Hence there exists q such that $(q,r) \in G[i,n]$, $(q,t) \notin G[i,n]$. Now, from $(r,t) \in R_n$, we derive the existence of an index $k > n$ and a state y such that $(r,y) \in G[n,k]$ and $(t,y) \in G[n,k]$. Hence, there exists a path $(t_j)_{n \leq j \leq k}$ with $t_n = t$, $t_k = y$, and $(t_j, t_{j+1}) \in G_{j+1}$ for all $n \leq j < k$. Moreover, there is a path from q to y because there are paths from q to r and from r to y. Let $\ell \leq k$ be the minimum index such that there is a path from q to t_ℓ. As there is no path from q to $t_n = t$, necessarily $\ell \geq n+1$. Obviously, $(t_{\ell-1}, t_\ell) \in G_\ell$, and by definition and minimality of ℓ, $(q, t_{\ell-1}) \notin G[i, \ell-1]$ and $(q, t_\ell) \in G[i,\ell]$. That is, $G[i, \ell-1]$ leaks at G_ℓ.

Let us now prove the second item, using the first one. Assume that i does not leak infinitely often, and towards a contradiction suppose that there are infinitely many j's such that $G[i,j]$ separates some $(r,t) \in R_j$. To each of these separations, we can apply item ***1.*** to obtain infinitely many indices m such that $G[i,m]$ leaks at G_{m+1}, a contradiction.

We now prove the last item. Since there are finitely many states in Q, there exists $q \in Q$ and an infinite set J of indices such that for every $j \in J$, $(q, y_{j+1}) \in G[i, j+1]$, $(q, x_j) \notin G[i,j]$, and $(x_j, y_{j+1}) \in G_{j+1}$ for some x_j, y_{j+1}. The path from q to y_{j+1} implies the existence of y_j with $(q, y_j) \in G[i,j]$, and $(y_j, y_{j+1}) \in G_{j+1}$. We thus found separated pairs (x_j, y_j) for every $j \in J$. To exhibit separations at other indices $k > j$ with $k \notin J$, the natural idea is to consider predecessors of the x_j's and y_j's.



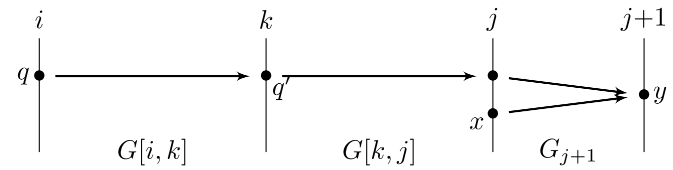
We define sequences $(r_k, t_k)_{k \geq i}$ inductively as follows. To define r_k, we take a $j \geq k+1$ such that $j \in J$; this is always possible as J is infinite. There exists a state r_k such that $(q, r_k) \in G[i,k]$ and $(r_k, y_j) \in G[k,j]$.
Also, as x_j belongs to $\mathrm{Im}(G[1,j])$, there must exist a state t_k such that $(t_k, x_j) \in G[k,j]$. Clearly, $(q, t_k) \notin G[i,k]$, else $(q, x_j) \in G[i,j]$, which is not true. Last, y_{j+1} is a common successor of t_k and r_k, that is $(t_k, y_{j+1}) \in G[k, j+1]$ and $(r_k, y_{j+1}) \in G[k, j+1]$. Hence $G[i,k]$ separates $(r_k, t_k) \in R_k$. ◀

▶ **Lemma 16.** *A play has infinite capacity iff there exists an index i such that i is remanent and leaks infinitely often.*

Proof. The direction from right-to-left is trivial. Assume the play has finite capacity, and let i be a remanent index. By Lemma 14, i does not leak infinitely oten.

For the other direction, assume that the play has infinite capacity. By Lemma 14, there exists an index i that leaks infinitely often. We choose i minimal with this property.

We first show that for all $k \geq i$, k leaks infinitely often as well. There are infinitely many indices $j > i$ such that $G[i,j]$ leaks at G_{j+1}. For each such index j, there are states q, x, y such that $(q, y) \in G[i, j{+}1]$, $(q, x) \notin G[i,j]$ and $(x, y) \in G_{j+1}$. Consider any index $i \leq k \leq j$. There exists a state q' such that $(q, q') \in G[i,k]$ and $(q', y) \in G[k, j{+}1]$. We thus have $(q', y) \in G[k, j{+}1]$, $(q', x) \notin G[k,j]$ and $(x, y) \in G_{j+1}$. Thus $G[k,j]$ leaks at G_{j+1}. This holds for all $j > i$ and $i \leq k \leq j$, so that for all $k \geq i$, $G[k,j]$ leaks at G_{j+1} for infinitely many indices j.



We prove now that some $k \geq i$ is remanent, which will finish the proof. Towards a contradiction, assume that it is not the case.

Let $\ell < i$. By minimality of i, ℓ leaks only finitely often. Applying the second statement of Lemma 15, there are only finitely many indices $j \geq \ell$ such that $G[\ell, j]$ separates some pair of R_j. We let j_ℓ the maximum of these indices, and $N = \max_{\ell < i} j_\ell$. By definition of N, for all $\ell < i$ and all $j > N$, $G[\ell, j]$ separates no pair of R_j.

Fix now $n > N$, the minimal index such that there exists j with $G[n,j] \in \mathcal{L}_j$ and for all $i \leq k \leq N$, $G[k,j] \notin \mathcal{L}_j$. The existence of n is guaranteed since we assumed for contradiction that no $k \geq i$ is remanent. Let J be the step at which index n is no longer tracked in the list. Just before the list is filtered to obtain $\mathcal{L}_J$, it starts with a prefix of the form: $G[i_1, J], \cdots, G[i_\ell, J], G[n, J]$. By definition of n, the indices $i_1, \cdots, i_\ell$ are smaller than i. That is, $i_1 < \cdots < i_\ell < i < N < n \leq J$.

Now, the choice of N guarantees that for all $1 \leq k \leq \ell$, $G[i_k, J]$ separates no pair in R_J. Moreover, $n \geq i$ thus n leaks infinitely often, and by the third statement of Lemma 15, $G[n, J]$ separates some pair of R_J, which cannot be separated by any $G[i_k, J]$. Therefore, during the third stage, $G[n, J]$ is not filtered. This contradicts the definition of J as the step after which index n is no longer tracked.

Thus some index larger than i is remanent, and leaks infinitely often. ◀

Proofs of Section 5

▶ **Proposition 19.** *There exists a family of NFA $(\mathcal{A}_n)_{n\in\mathbb{N}}$ such that $|\mathcal{A}_n| = 2n+7$, and for $M = 2^{2^n+1}+n$, there is no winning strategy in $\mathcal{A}_n^M$ and there is one in $\mathcal{A}_n^{M-1}$.*

Proof. Let $n \in \mathbb{N}$. The NFA $\mathcal{A}_n$ we build is the product of *two* NFAs with different properties: $\mathcal{A}_n = \mathcal{A}_{\mathsf{split}} \times \mathcal{A}_{\mathsf{count},n}$. On the one hand, for $\mathcal{A}_{\mathsf{split}}$, winning the game with m agents requires $\Theta(\log m)$ steps. On the other hand, $\mathcal{A}_{\mathsf{count},n}$ implements a usual counter over n bits (as used in many different publications), such that Controller can avoid to lose during $O(2^n)$ steps. The combination of these two gadgets ensures a cut-off for $\mathcal{A}_n$ of 2^{2^n}.

Figure 6 shows the counting gadget that implements a counter with states l_i (meaning bit i is 0) and h_i (for bit i is 1). It enjoys the following properties: $(c1)$ there is a strategy in $\mathcal{A}_{\mathsf{count},n}$ to ensure avoiding ☹ during 2^n steps, by playing α_i whenever the counter suffix from bit i is $01\cdots1$; $(c2)$ for $m \geq n$, no strategy of $\mathcal{A}_{\mathsf{count},n}^m$ avoid ☹ for 2^n steps.

Recall Figure 1, which presents the splitting gadget that has the following properties. In $\mathcal{A}_{\mathsf{split}}^m$ with $m \in \mathbb{N}$ agents, $(s1)$ there is a winning strategy ensuring to win in $2\lfloor \log_2 m \rfloor + 2$ steps; $(s2)$ no strategy can ensure to win in less than $2\lfloor \log_2 m \rfloor + 1$ steps.

The two gadgets (splitting and counting) are combined into their product $\mathcal{A}_n = \mathcal{A}_{\mathsf{split}} \times \mathcal{A}_{\mathsf{count},n}$, with actions consisting of pairs of action, one for each gadget: $\Sigma = \{a, b, \delta\} \times \{\alpha_i \mid 1 \leq i \leq n\}$. The initial state is the q_0 of $\mathcal{A}_{\mathsf{count},n}$, and we add transitions labeled $\alpha_1, \dots, \alpha_n$ from this initial state to the q_0 of $\mathcal{A}_{\mathsf{split}}$. We add an action $*$ which can be played from any

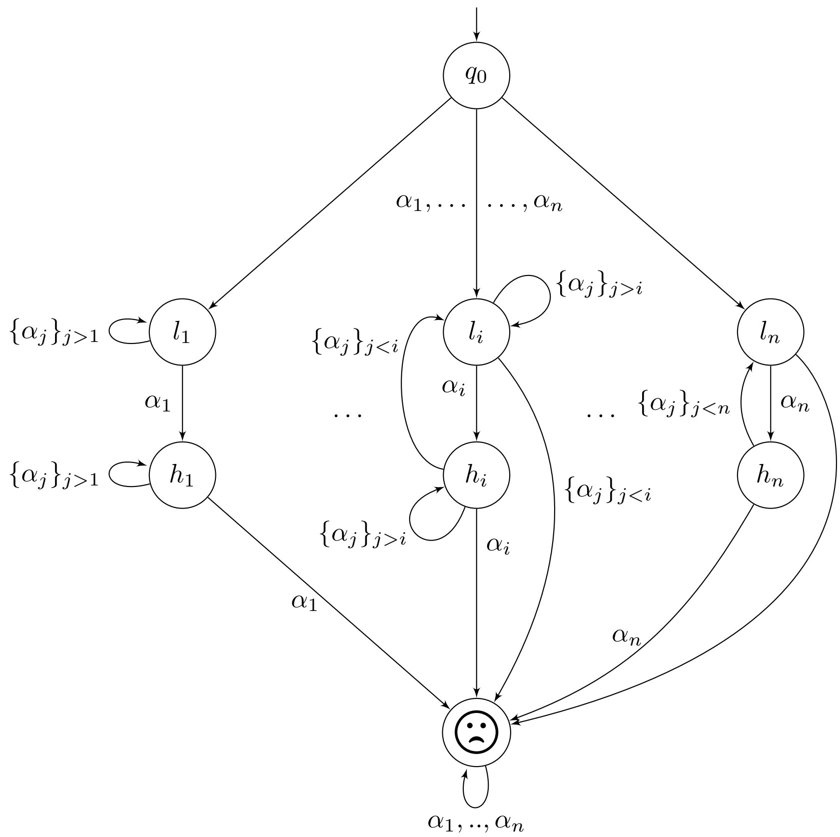


Figure 6 The counting gadget $\mathcal{A}_{\mathsf{count},n}$.

state of $\mathcal{A}_{\mathsf{count},n}$ but ☹, and only from f in $\mathcal{A}_{\mathsf{split}}$, leading to the global target state ☺.

Let $M = 2^{2^n+1} + n$. We deduce that the cut-off is $M - 1$ as follows:

- For M agents, a winning strategy for Agents is to first split n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, in order to fill each l_i with 1 token, and 2^{2^n+1} tokens to the q_0 of $\mathcal{A}_{\mathsf{split}}$. Then Agents splits evenly tokens between q_1, q_2 in $\mathcal{A}_{\mathsf{split}}$. In this way, Controller needs at least $2^n + 1$ steps to reach the final state of $\mathcal{A}_{\mathsf{split}}$ $(s2)$, but Controller reachs ☹ after these $2^n + 1$ steps in $\mathcal{A}_{\mathsf{count},n}$ $(c2)$.
- For $M - 1$ agents, Agents needs to use at least n tokens from the initial state to the q_0 of $\mathcal{A}_{\mathsf{count},n}$, else Controller can win easily. But then there are less than 2^{2^n+1} tokens in the q_0 of $\mathcal{A}_{\mathsf{split}}$. And thus by $(s1)$, Controller can reach f within 2^n steps, after which he still avoids ☹ in $\mathcal{A}_{\mathsf{count},n}$ $(c1)$. And then Controller sends all agents to ☺ using $*$.

Thus, the family $(\mathcal{A}_n)$ of NFA exhibits a doubly exponential cut-off. ◀