école normale supérieure paris—saclay

LMF Laboratoire Méthodes Formelles

Synthesizing coalition strategies in parameterized concurrent games

Luc Lapointe
under Patricia Bouyer's guidance

MPRI M2 Internship
Gif-sur-Yvette, LMF
March - July 2023

General Context

Lots of modern systems used in everyday life happen to be distributed systems. Examples include cloud computing, blockchain technologies, servers with multiple clients, wireless sensor networks, bio-chemical systems, or fleets of drones. Verification of distributed systems is consequently a major topic in verification, and potential applications are numerous.

One additional difficulty is that some distributed systems may involve an arbitrary number of agents. To address this issue, the model-checking community designed not only distributed but also parameterized models, where the parameter is the number of agents.

Concurrent games model distributed systems by design. A previous work introduced concurrent games in which the number of agents is arbitrary, called parameterized concurrent games [1]. As the model is very recent, it is not a surprise that although some results on it have already been discovered with simple objectives [1, 2], most problems on parameterized concurrent games, even some apparently very simple ones, remain unsolved for now.

Problem Studied

My internship's goal was to find wether the cooperative reachability problem in parameterized concurrent games is decidable, and if so to discover the complexity class where it lies. Problems that have already been studied are reachability problem when one player faces all of the others [1] and safety problem when all players cooperate [2]. The cooperative reachability problem is at the crossroad of those two problems, and as such is a natural following in studying parameterized concurrent games.

As the problem is mostly finite, the consensus at the beginning of my internship was that it should be decidable. However, surprisingly, no algorithm was known yet, and *a fortiori* no complexity was known. Backing this intuition of decidability with a proven algorithm was thus an important preliminary step in studying the problem.

Proposed Contributions

The first step I climbed to solve this problem was to study a simpler version of the problem, where both the arena and expressivity of available plays are very restricted. Even this very restricted problem happened to be uneasy to solve.

After formally defining the problem (section 2), my first contribution is the design of an algorithm solving this restricted problem (section 3), thus establishing its decidability. Proving its correction, and in particular its completeness, was the main focus of my internship (section 4). This proof of completeness gives a naive upper bound on the complexity of the algorithm. I also established some lower bounds on its complexity for different versions of the problem (section 2.2).

Arguments Supporting Their Validity

The algorithm I designed seems like a natural algorithm to solve the problem. Despite being applied to very restricted cases, both in term of arena of the game and expressivity of what the agents can play, it can maybe be extended to more regular cases, through upgrading technical details rather than having pioneer ideas.

The correctness, and in particular the completeness of the algorithm, is a technical proof that, in idea, allows to truncate the infiniteness of the problem. This trick of making the problem finite to solve it seems like a core idea to solve the problem, as it was a very necessary step in different approaches I tried to solve it.

Also, bounds on the efficiency of my algorithm are currently quite loose, but we conjecture, after extensively studying different examples and prove schemes, that the actual complexity is the current lower bound.

Summary and Future Work

My contribution brings first pieces of knowledge about how to solve the parameterized cooperative reachability problem. It also lays the foundations for a deeper understanding of how cooperative parameterized concurrent games behave, as the algorithm I designed is both a natural first step for studying harder problems, and also features a key technique of truncating infinity.

Immediate future works for cooperative parameterized concurrent games include : extending the algorithm to less restricted cases of cooperative games, both in term of expressivity of plays and shape of the arena; and establishing tighter bounds on its complexity. A more long-term goal would be to extend the problem to objectives that are more general than only reachability, such as objectives described with temporal logic.

Other interesting future works tied to this problem are classifying problems on parameterized concurrent games where agents are not all on the same team; or implementing algorithms for concurrent games to some real-life system.

Abstract

Traditional concurrent games on graphs involve a fixed number of players, who take decisions simultaneously, determining the next state of the game. The aim of this internship is to study the problem of synthesizing a coalition strategy to achieve a reachability objective in a variant of concurrent games with an arbitrary number of agents. The problem is non-trivial since the agents do not know *a priori* how many they are when they start to play. We give different upper and lower bounds on the difficulty of synthesizing arbitrarily-large coalition winning strategy for reachability objectives depending on the form of the input, of the arena, and the expressivity of available plays.

Contents

1 Introduction

Context Lots of modern systems used in everyday life happen to be distributed systems. Examples include cloud computing, blockchain technologies, servers with multiple clients, wireless sensor networks, bio-chemical systems, or fleets of drones [5]. Synthesizing winning strategies for such systems means building controllers that will allow them to coordinate and adapt their behavior to stay safe or achieve some goal. Potential long-term applications are therefore numerous.

These systems are not only distributed, but they also involve an arbitrary number of agents. The model-checking community is interested both in verifying parameterized systems [8, 4] and synthesizing systems to achieve a given objective [12]. Our contribution is about synthesizing strategies to achieve a fixed goal in parameterized distributed systems.

Parameterized verification What we mean by parameterized verification is verification of systems where the number of agents can be arbitrarily large, and where most of the time each individual agent does not know how much agents are interacting within the system. Despite this lack of information, algorithms and protocols applied on such systems are designed to operate efficiently and correctly, whatever the number of agents involved.

In the last 15 years, parameterized verification algorithms were successfully applied to various case studies, such as data-consistency for cache coherence protocols in uniform memory access multiprocessors [7], and the core of simple reliable broadcast protocols in asynchronous systems [9].

Distributed synthesis In general, the problem of distributed synthesis asks whether strategies for individual agents to achieve a global objective can be computed, in a context where individuals have only a partial knowledge of the environment. In the context of parameterized verification, the knowledge agents do not have is the number of agents interacting with the system.

There are several possible formalizations for distributed synthesis, for instance via an architecture of processes with communication links between agents [12], or using coordination games [11, 10, 3]. The two settings are linked, and many (un)decidability results have been proven, depending on various parameters.

Concurrent games on graphs A *concurrent game on graphs* for n players is a turn-based game played inside a given graph called the *arena* in which the game is playing. An arena for n players have edges labelled with n-tuples of actions (or simply words of length n on the action alphabet). The game starts at a vertex v_0 of the arena. At each turn of the game, each of the n players selects an action, and the next vertex is determined by the combination of actions.

With this definition, it is implicit that players have identifiers. Thus, when player i chooses action a_i, next vertex is the one pointed at by an edge labelled with the word $a_1 \dots a_n$.

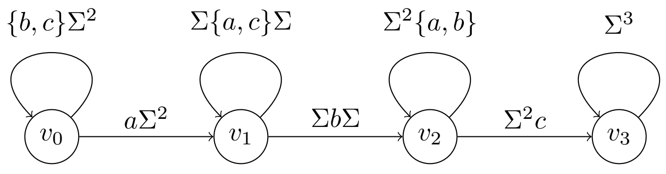


Figure 1: A very simple example arena with action alphabet $\Sigma = \{a, b, c\}$. Players reach state v_1 from state v_0 if the first player plays en a, whatever the other players chose to play. Similarly, they move from state v_1 (resp. v_2) to state v_2 (resp. v_3) if the second (resp. third) player plays b (resp. c).

Parameterized concurrent games on graphs A previous work introduced concurrent games in which the number of agents is arbitrary [1]. These games generalize concurrent games with a fixed number of agents, and can be seen as a succinct representation of infinitely many games, one for each fixed number of agents. This is done by replacing, on edges of the arena, words representing the choice of each of the agents by languages of finite yet unbounded words. Such a parameterized arena can represent infinitely many interaction situations, one for each possible number of agents. In parameterized concurrent games, the agents do not know at the start of the game the number of agents participating to the interaction. Each agent observes the action she plays and the vertices the play goes through. These pieces of information may refine the knowledge each agent has on the number of involved agents.

In [1], the problem considered is whether Agent 1 can ensure a reachability objective independently of the number of her opponents, and no matter how they play. The problem is non trivial since Agent 1 must win with a *uniform* strategy. It has been proven in this paper that when edges are labeled with regular languages, the problem is PSPACE-complete; and for positive instances one can effectively compute a winning strategy in polynomial space.

An other problem one might consider is whether the agents can collectively ensure a safety objective, without knowing how many they are. As in the previous problem, only a uniform strategy, this time of the coalition rather that of agent 1 only, can ensure such objective. It has been proven in [2] that synthesizing such a strategy is a PSPACE-hard problem that can be decided in exponential space.

Contributions The problem studied here is the cooperative reachability problem. It includes features of both of the problems described above, but yet the complexity class where it stands, and even its decidability, is until now unknown. In the following we define it precisely in section 2, where we also give the precise statements of our main results. We exhibit an algorithm to solve a restricted version of the problem in section 3. Section 4 establishes the correction of the algorithm in its restricted setting, by exhibiting a periodic pattern that allows to truncate infiniteness in the only unbounded parameter of the problem. This proof comes with an upper bound on the complexity of the algorithm.

2 Definitions

2.1 Defining the problem

A specific type of concurrent game We are interested in solving a type of concurrent game on graphs called *Parameterized cooperative reachability problem*. Let us look at what this long name means, by explaining meaningful words in it :

- parameterized : In this very problem, *parameterized* means that when you consider different runs on the same arena, the number of players involved in a run might change from a run to another. It is fixed at the beginning of a run, but at the start of the run it is not known by any player which is involved.
- cooperative : There are different players involved in a given run, but they win or lose all together. Thus, they must not play against each other, but help each other.
- reachability : The aim of the unique team of players is to reach a state of the graph.

To model the fact that a given arena must be playable for any number of players, one must slightly modify the concurrent game model explained above. In our model, arena edges are not labelled simply with words of length n (which would fit for runs with n players only), but with languages instead.

Definition 1 : Parameterized arena

A *parameterized arena* is a triple $\mathcal{A} = (V, v_0, \Sigma, \delta)$ where

- V is a finite set whose elements are called the *vertices* or *states* of the arena,
- $v_0 \in V$ is the *initial state* of the arena,
- Σ is a finite set called the *actions set* of the arena,
- δ is a function from $V \times V$ to 2^{Σ^*} called the *transitions labels* of the arena.

Furthermore we add the following conditions to the transitions labels :

$$\forall v, v', v'' \in V,\ \ v' \neq v'' \implies \delta(v, v') \cap \delta(v, v'') = \emptyset \qquad \text{(Deterministic arena)}$$

$$\forall v \in V,\ \ \bigcup_{v' \in V} \delta(v, v') = \Sigma^* \qquad \text{(Complete arena)}$$

From now on, the word *arena* always refer to a *parameterized* arena.

What is a cooperative strategy We consider that information available to a given player prior to the start of a run is : the whole parameterized arena, and her identifier. Thus, even though no player knows the number of players involved in a given run, if player i is involved, she knows that there are at least i players involved.

Definition 2 : History, cooperative strategy, plays, run

Let $\mathcal{A} = (V, v_0, \Sigma, \delta)$ be an arena.
A *history* $\mathcal{H} \in V^+$ of vertices of $\mathcal{A}$ is a finite sequence of vertices of $\mathcal{A}$.
A *strategy* for the i^{th} player is a function $\sigma_i : V^+ \to \Sigma$ that assigns to the history of vertices that she has visited so far an action she plays.
A *cooperative strategy* is a function $\sigma : V^+ \to \Sigma^\omega$. It can be seen as a profile of infinitely many strategies :

$$\sigma = \sigma_1; \sigma_2; \ldots$$

An infinite sequence of actions $p \in \Sigma^\omega$ is called a *general play* of a cooperative strategy. For a given history of vertices $\mathcal{H}$, the i^{th} letter of the general play $\sigma(\mathcal{H})$ denotes what the i^{th} player must play when seeing history $\mathcal{H}$ to follow the cooperative strategy σ.
Notation $p|_{\leq k}$ denotes the prefix of size k[a] of p. We call $p|_{\leq k}$ a *play* of a cooperative strategy when there are k players. Playing a general play when there are k players consists in playing its prefix of length k.
For a given cooperative strategy σ and a number of players k, we say that the *run* for k players of σ is the unique infinite sequence of plays $p_1 \ldots p_n \ldots$ that σ plays in $\mathcal{A}$. It is defined inductively together with its sequence of *visited vertices* $v_0 \ldots v_n \ldots$:

$$\begin{cases} & p_1 = \sigma(v_0)|_{\leq k} \\ \forall i \in [\![1, \infty[\![, & v_i \text{ is the unique vertex } v \text{ such that } p_i|_{\leq k} \in \delta(v_{i-1}, v_i) \\ \forall i \in [\![2, \infty[\![, & p_i = \sigma(v_0 \ldots v_{i-1})|_{\leq k} \end{cases}$$

An infinite sequence of plays $(p_i)_{i \in \mathbb{N}}$ is *coherent* with σ if there exists a number of players $k \in \mathbb{N}$ such that it is the run of σ for k players. A finite sequence of plays $p_1 \ldots p_n$ is *coherent* with σ if it is the prefix of a infinite sequence of plays coherent with σ.

[a] we start indexing letters from 1.

From now on, the word *strategy* always refers to a *cooperative* strategy, and the word *play*, when not explicitly a play for k players, always refers to a *general play*.

Definition of the general problem We are interested in finding how an agent with identifier i can synthesize a strategy with what she knows, such that if all agents locally synthesize their strategy, it creates a global strategy winning whatever is the number of agents involved in a run.

This leads to the general definition of the decision problem :

PARAMETERIZED COOPERATIVE REACHABILITY PROBLEM (PCRP)

INPUT: A parameterized arena $\mathcal{A} = (V, v_0, \Sigma, \delta)$, a target vertex $t \in V$.
OUTPUT: Is there a cooperative strategy σ such that for all numbers of players $k \in \mathbb{N}$, the run for k players of σ visits t ?

Note that if the answer is yes and the algorithm finding the answer is constructive, the input is such that any agent can locally run the algorithm to find the global strategy σ from which she can deduce her own strategy.

State of the art The parameterized cooperative *safety* problem is a PSPACE-hard problem that can be decided in exponential space [2]. To our knowledge, the decidability of the parameterized cooperative *reachability* problem, and *a fortiori* the complexity classes where it lies, is an open problem. The following work achieves the very first step towards a solution by studying a restricted version of the problem.

Definition 3 : Flat automaton, flat language, flat regular expression.

A *flat automaton* is a deterministic finite automaton whose induced graph contains no cycle except self-loops. A *flat language* is a language recognized by a flat automaton. A *flat regular expression* is a regular expression denoting a flat language.

Definition 4 : Tricolored flat arena

A *tricolored flat arena* $\mathcal{A} = (V, \Sigma, \delta)$ is a parameterized arena with target ☺ where transitions label languages are encoded with regular expressions, and :

$$V = \{☺, ☹, ◯\} \qquad \delta(☺, ☺) = \Sigma^* \qquad \delta(☹, ☹) = \Sigma^*$$
$$\delta(◯, ◯) = e_\circ \qquad \delta(◯, ☹) = e_\perp \qquad \delta(◯, ☺) = e_\top$$

and Σ^* is a disjoint union of flat languages $L_\circ, L_\perp$ and $L_\top$ of expressions $e_\circ, e_\perp, e_\top$. Initial state is ◯.

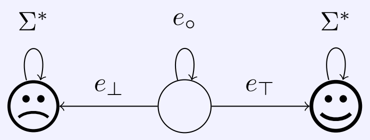


Figure 2: Illustration of a tricolored flat arena. $L_\circ \sqcup L_\perp \sqcup L_\top = \Sigma^*$.

In the following, we are interested in tricolored flat arenas, so the word *arena* alone will from now on always refer to a *tricolored flat arena*.

One way to represent the three languages $L_\top, L_\perp$ and $L_\circ$ is to build three different flat automata that recognize them. There is even a simple representation using a joint automaton, as below.

Lemma 5 : Joint automaton

For every triple of flat languages $L_\top, L_\perp$ and $L_\circ$ such that $L_\circ \sqcup L_\perp \sqcup L_\top = \Sigma^*$, there exists a structure $(\Sigma, Q, \delta, q_0, F_\top, F_\perp, F_\circ)$ with $F_\top \sqcup F_\perp \sqcup F_\circ = Q$ such that for every $\Delta \in \{\top, \perp, \circ\}$, $\mathcal{A}_\Delta = (\Sigma, Q, \delta, q_0, F_\Delta)$ is a deterministic flat automaton that recognizes L_Δ. We call this structure a *joint automaton*.

Proof sketch[a] Use cartesian product of the three original automata. □

[a] Proof sketches are shortened versions of complete proofs available in appendix.

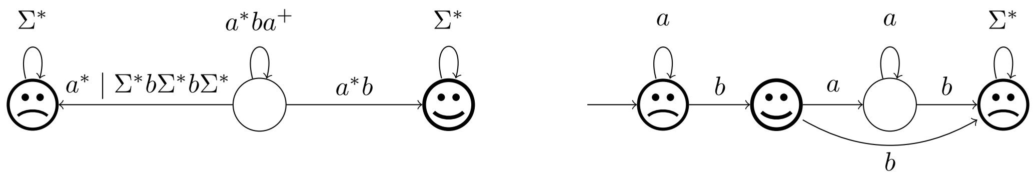


Figure 3: On the left, a specific tricolored flat arena. On the right, a joint automaton describing the outgoing languages at the ◯ state. In this graphical representations, states of the joint automaton depicted with ☺ (resp. ☹, ◯) are states labelled with $F_\top$ (resp. $F_\perp, F_\circ$) in the joint automaton.

Playing on a joint automaton A joint automaton A of the ◯ state in an arena $\mathcal{A}$ can be an efficient structure to check what is the outcome of a given play $p \in \Sigma^\omega$ from ◯ in $\mathcal{A}$. When considering a run where there are k players involved, reading p with A until its k^{th} letter leads to a state in $F_\top$ (resp. $F_\bot$, $F_\circ$) if and only if playing p with k players in $\mathcal{A}$ leads to the ☺ (resp. ☹, ◯) state of the arena.

Different kinds of inputs One considers two slightly different problem with this restricted version of Pcrp, where the difference lies in the way the three languages are represented.

The first one takes a triple of regular expression $(e_\top, e_\bot, e_\circ)$ whose languages are flat as an input.

Parameterized cooperative reachability problem in Tricolored Arena
with Flat Regular Expression

Input:	A triple of flat regular expressions $(e_\top, e_\bot, e_\circ)$ describing the outgoing languages at the ◯ state of a tricolored flat arena $\mathcal{A}$.
Output:	Is there a cooperative strategy σ such that for all numbers of players $k \in \mathbb{N}$, the run for k players of σ visits ☺ ?

The second is a slightly different version where it is a joint automaton of the ◯ state that is given as input, and not regular expressions.

Parameterized cooperative reachability problem in Tricolored Arena
with Joint Flat Automaton

Input:	A joint flat automaton A describing the outgoing languages at the ◯ state of a tricolored flat arena $\mathcal{A}$.
Output:	Is there a cooperative strategy σ such that for all numbers of players $k \in \mathbb{N}$, the run for k players of σ visits ☺ ?

Example 6 : Solving Pcrp on a specific example For the arena of figure 3, a winning strategy is that agent i plays b on the i^{th} turn, and a on any other turn. We explain how to find it in the following paragraphs.
It is important, when building a winning strategy, that it can never lose, whatever the number of players involved. This is the reason why player 1 must play b on first turn : because she doesn't know if there are any other players involved, and playing an a would be losing if she is the only one involved.
Knowing this, any other player must play an a on turn 1, because the only allowed b is played by player 1. So the first play of a winning strategy must be denoted by the expression

$$ba^\omega$$

whose i^{th} letter denotes what action player i plays on turn 1.
Once ba^ω is played, the game is over if there is only one player. Consequently, player 1 can play a rather than b without losing instantly. But if she does so, exactly one of the other players must play b ; because if no b or two b are played, the game is lost. If this one player is not player 2, then the game is lost when there are exactly two players, because in this case what is played is aa. Taking this constraint into account, one possible second non-losing play is

$$aba^\omega.$$

By repeating the reasoning, one eventually exhibits the (winning) strategy where player i plays b at i^{th} turn, and a at any other turn, represented as the sequence

$$ba^\omega; aba^\omega; a^2ba^\omega; \ldots$$

2.2 Theorem statement

The main theorem we prove is the following:

Theorem 7

Pcrp in tricolored arena with joint flat automaton is in Nexptime, and our algorithm solving the decision problem also synthesizes a winning strategy if it exists.

The algorithm which solves this problem is algorithm 1. From it we can build a sibling algorithm to prove corollary 8.

Corollary 8

Pcrp in Tricolored Arena with flat regular expression is in Nexptime, and our algorithm solving the decision problem also synthesizes a winning strategy if it exists.

We also discovered some hardness results on similar problems but whose inputs are slightly richer than those of the two problems described above.

Theorem 9

Pcrp in Tricolored Arena with flat regular expressions is NP-hard **when negation can be used** inside the regular expression it takes as an input.

Proof sketch Reduction from 3-SAT. Valuations on N variables are modelled by boolean words of size N. For each clause of the input formula, add to $L_\perp$ any word that evaluates this clause to *false*. The complementary of all of these additions is in $L_\top$. $\square$

Theorem 10

Pcrp in Tricolored Arena with regular expressions is PSpace-hard **when negation can be used** inside the regular expression it takes as an input.

Proof sketch Reduction from Non-Universality. Build an arena such that

- there exist a word in the complementary language of input regular expression e if and only if it is possible to win for a given arbitrary number of players;
- a win for one single number of players allows to win for any number of players.

$\square$

From now on, we fix an instance of our problem, that is, a triple of regular expressions $(L_\top, L_\perp, L_\circ)$ and a joint automaton A, both describing the same tricolored arena $\mathcal{A}$.

2.3 Technical definitions

Definitions in this section are mainly used to state and prove different intermediate lemmata.

Definition 11 : Winning play

Let $(p_i)_{i>0} \in (\Sigma^\omega)^\mathbb{N}$ be a sequence of plays. p_i is *winning for k players* if when k players are involved, the current state after playing p_i is ☺.

Representing the outcomes of a run during its execution For a given sequence of general plays coherent with a winning strategy σ, the set of players for which it has already won at a given time of the sequence is growing as more and more general plays are played. One can represent this growing set of achieved wins by a sequence of words in $\{0,1\}^\omega$ where the j^{th} letter of word i is 1 if and only if the game is won after play i when j players are involved. Definition 12 and example 13 expand on this idea.

Definition 12 : Wins word

Let $\mathbb{B} = \{0,1\}$ and σ a winning strategy in $\mathcal{A}$. Let $(p_i)_{i>0}$ a sequence of general plays in $\mathcal{A}$ coherent with σ. When k players are involved, the game might or might not be won after p_i is played.
We denote the unique set of numbers of players for which the game is won after p_i is played by an infinite word $w^i_\sigma \in \mathbb{B}^\omega$ where

The j^{th} letter of w^i_σ, noted $(w^i_\sigma)_j$, is 1
if and only if
after p_i is played, ☺ is in the history of visited vertices when there are j players involved.

w^i_σ is called *wins word of σ after the i^{th} play.*

This encoding is only defined for winning strategies. This is because it is winning strategies we will study, and restricting the definition to winning strategies allows not to define how to encode losing plays.

Example 13 : Wins word Consider the joint automaton depicted in figure 4. $p_2(n)$ (resp. $p_5(n)$) denotes the play going for the first (resp. second) column and self-looping n times on the first self-loop ; so $p_i(n)$ wins when there are $i+n$ players.
Consider a winning strategy σ starting with the following sequence of plays :

$$p_3; p_5(1); p_2(2); p_5(2); p_2(3); p_2(0); p_1$$

The sequence of wins word achieved by σ is of the form :

What is played	Wins word
p_3	$w_\sigma^1 = 00100000\dots$
$p_5(1)$	$w_\sigma^2 = 00100100\dots$
$p_2(2)$	$w_\sigma^3 = 00110100\dots$
$p_5(2)$	$w_\sigma^4 = 00110110\dots$
$p_2(3)$	$w_\sigma^5 = 00111110\dots$
$p_2(0)$	$w_\sigma^6 = 01111110\dots$
p_1	$w_\sigma^7 = 11111110\dots$

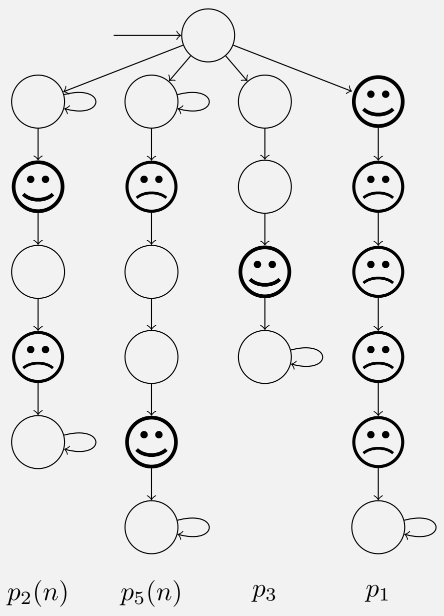


Figure 4: Example automaton. Action letters are omitted for clarity. This can always be done without loss of generality, and will be done from now on.

With such a representation, for a winning strategy σ, the path in automaton A that should be followed at turn i is a path where the j^{th} vertex is in $F_\perp$ only if the j^{th} letter of w_σ^{i-1} is a 1. Also, when following a winning strategy, the sequence of wins words tends to 1^ω when the number of turns played tends to infinity :

$$\lim_{i\to\infty} w_\sigma^i = 1^\omega$$

Definition 14 : Dependent wins

A play p depends on a win for n players if reading its prefix of length n reaches a state in $F_\perp$ in the joint automaton.

Example 15 : Dependent wins On example 13, p_1 depends on wins for $2, 3, 4, 5$ players. $p_2(n)$ depends on a win for $4+n$ players.

Definition 16 : Dependent plays

Let σ be a winning strategy, and $(p_i)_{i>0}$ be the sequence of plays coherent with σ. Let p be the k^{th} play of this sequence (so p is exactly p_k). The set of *direct dependencies of* p, denoted $dep_1(p)$, is the set of plays q where there exist n such that

- $q = p_l$ with $l < k$;
- p depends on a win for n players;
- and in a run where n players are involved, w_σ^l is the first wins word whose n^{th} bit is 1.

The set of all dependencies of p is

$$dep(p) = \bigcup_{i>0} dep_1^i(p)$$

where $dep_1^1(p) = dep_1(p)$ and for $i > 1$,

$$dep_1^i(p) = \bigcup_{q\in dep_1^{i-1}(p)} dep_1(q).$$

Example 17 : Dependent plays Consider the strategy described in example 13. We have the following equalities :

$$
\begin{array}{llll}
dep_1(p_3) = \emptyset & dep_1(p_5(1)) = \{p_3\} & dep_1(p_2(2)) = \{p_5(1)\} & dep_1(p_1) = \{p_2(0), p_3, p_2(2), p_2(3)\} \\
dep(p_3) = \emptyset & dep(p_5(1)) = \{p_3\} & dep(p_2(2)) = \{p_5(1), p_3\} & dep(p_1) = \{p_2(0), p_3, p_2(2), p_2(3), p_5(1), p_5(2)\}
\end{array}
$$

Definition 18 : Depth

In a deterministic automaton, the *depth of a state* q is the maximum number of different vertices of a path from the initial state to q. The *depth of the automaton* is the maximum of the depth of its states.

The automaton depicted in figure 4 has depth 7.

Definition 19 : Induced path and loopless path of a play, induced play of a path

Let G be the graph induced by A, and Σ the set of actions of the arena it denotes.
Let e be an ω-expression on Σ describing a play in $\mathcal{A}$ such that the only action with an ω-exponent is the last one of e[a]. The *path induced by* e is the sequence of vertices of G visited when playing actions of e', where e' is e but without the ω-exponent on the last action. The *loopless path induced by* e is the same but going through each vertex only once.
Let ρ be a finite path of vertices of G ending with a vertex admitting a self-loop. The *play induced by* ρ is the the sequence of actions that follows the path ρ until the self-loop is reached, and then self-loops.

[a] As the only cycles in A are self-loops, any ω-expression e describing a play in $\mathcal{A}$ admits an equivalent one whose only action with an ω-exponent is the last.

Example 20 The function associating a play to the induced play of its induced path is identity. The function matching a path to the induced path of its induced play is identity as well.

On figure 6 are depicted induced path and induced loopless path of $p_2(1)$. For any integer n, induced play of induced loopless path of $p_2(n)$ is $p_2(0)$.

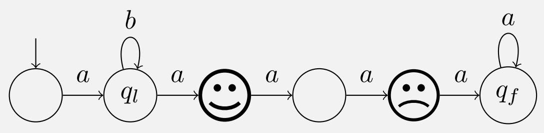


Figure 5: $p_2(n)$ of example 13 with letters and some state names.

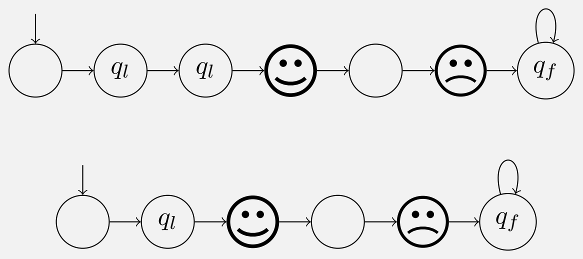


Figure 6: Induced path and induced loopless path of $p_2(1) = abaaaaa^\omega$.

3 The algorithms

The algorithm we use to solve PCRP WITH TRICOLORED JOINT FLAT AUTOMATON consists in playing so-called winning *safe paths* until it can not find any more, and returns *true* if and only if it was enough to win for all integers below a given bound f we will explicit later.

Safe paths are (a subclass of) paths that can not lose for any number of players at the time they are played.

Definition 21 : Safe path

Let S be an array of $\bot$ and $\top$ indexed from 1 to N, and si[a]$\in \{\top, \bot\}$. $\rho = q_1 \dots q_k$ is a *safe path* according to S and si if the two following properties hold.

1. $\forall i \leq k,\ q_i \in F_\bot \implies i \leq N \land S[i] = \top$
2. $q_k \in F_\bot \implies si = \top \land \forall i \in [\![k, N]\!], S[i] = \top$

[a] S stands for *safe*, and si stands for *safe infinity*.

Algorithm 1 Solving PCRP IN TRICOLORED ARENA WITH JOINT FLAT AUTOMATON

Input: A flat joint automaton $A = (\Sigma, Q, \delta, q_0, F_\top, F_\perp, F_\circ)$ describing a tricolored flat arena $\mathcal{A}$.
Output: Is there a cooperative winning strategy in $\mathcal{A}$?

```
1:  $h \leftarrow$ depth of $A$
2:  $S \leftarrow$ array of $\perp$ indexed from 1 to $f(h)$        ▷ $f$ is the function of theorem 23.
3:  $si \leftarrow \perp$                                            ▷ $si$ for *safe infinity*.
4:  $G \leftarrow$ induced labelled graph of $A$

5:  procedure UPDATE_S($\rho$)                                     ▷ $\rho = q_1 q_2 \ldots q_k$ is a safe path.
6:      for each vertex $q_i$ in $\rho$ that is in $F_\top$ do
7:          $S[i] \leftarrow \top$
8:      if $q_k \in F_\top$ then      ▷ The final self-loop solves all high numbers of players in one hit.
9:          $si \leftarrow \top$
10:         for $k \le i \le |S|$ do
11:             $S[i] \leftarrow \top$

12: while $true$ do
13:     Find a safe path $\rho$ in $G$ of length less than $f(h) + h$ such that UPDATE_S($\rho$) sets at least one $\perp$ of $S$
    to $\top$.
14:     UPDATE_S($\rho$)
15:     if $\forall 1 \le i \le |S|,\ S[i] = \top$ then
16:         return $true$
17:     if neither $S$ nor $si$ changed inside this while loop iteration then
18:         return $false$
```

Example 22 : Execution on the joint automaton of figure 3 At the beginning of the game, as the run has not won for any number of players yet, safe plays are exactly the ones never going through a ☹ state. Thus, the only safe play available is ba^ω.
It happens that this play wins for 1 player. Consequently, at second turn, another safe play available is aba^ω.
At each new turn, there is exactly one new safe path that sets at least one 0 to 1 in the array. The sequence of those paths is the one described in example 6.

Algorithm 1 solves the decision problem without synthesizing a winning strategy, but it can quite easily be adapted to do so, by remembering in order the paths ρ such that UPDATE_S(ρ) add $\top$s in S.

Corollary algorithm to solve PCRP IN TRICOLORED ARENA WITH FLAT REGULAR EXPRESSIONS
From algorithm 1 we can design another algorithm that takes the triple of expressions $(e_\top, e_\perp, e_\circ)$ as an input rather than the deterministic joint automaton.

It starts with building three nondeterministic automata $A_\top, A_\perp, A_\circ$, and then does roughly exactly the same things, by guessing plays rather than paths. As we have three nondeterministic automata rather than one deterministic joint automaton, testing if the play we guessed is safe or not, and testing the numbers of players for which it is winning, is only quadratically slower, because it can be achieved by testing each prefix.

The depth of a nondeterministic automaton is the same as the depth of the deterministic one built with the powerset construction. Thus, for a given regular expression, there exists a (non)deterministic automaton that recognizes its language, and whose depth is linear in the size of the input expression. Consequently, if algorithm 1 is in P (resp. EXPTIME), this sibling algorithm also is.

3.1 Correctness

This algorithm is correct because there exists $f : \mathbb{N} \to \mathbb{N}$ such that theorem 23 is true. Please note that f is hard-coded in the theorem statement. Section 4 is devoted to proving this theorem with an exponential such f.

Theorem 23

Let $\mathcal{A}$ be a tricolored flat arena, and A be a joint automaton describing the outgoing edges of the ◯ state in $\mathcal{A}$. Assume A has depth h.

There exists a cooperative strategy σ winning whatever the number of players involved if and only if there exists a cooperative strategy σ' such that :

1. σ' wins for any number of player $n \leq f(h)$;
2. no play of σ' winning for n players with $n \leq f(h)$ requires that the game has been won by a previous play for m players with $m > f(h)$.

Reverse ($\Leftarrow$) implication of theorem 23 ensures correctness, meaning that algorithm 1 is right when it answers *true*. Direct ($\Rightarrow$) implication ensures completeness, meaning that it is right when it answers *false*.

3.2 Complexity upper bound

There are two sources of high complexity in this algorithm :

1. The amount of iterations in the while loop line 12. It is bounded by $f(h)+2$ due to the two stop conditions that are tested.
2. Finding a convenient path on line 13. It can be done non-deterministically in time linear in $f(h)+h$.

Thus, assuming theorem 23 is true for some $f(h) \geq h$, Pcrp in Tricolored Arena with Joint Flat Automaton is in NTIME(f^2). Section 4, which is devoted to proving the theorem, exhibits an exponential such f.

4 Winning strategies can have bounded dependencies

The goal of this section is to prove that there exist a function $f : \mathbb{N} \to \mathbb{N}$ such that theorem 23 is true. In the rest of this section, A will be the flat automaton of a tricolored arena $\mathcal{A}$, and h the depth of A.

4.1 Correctness

The reverse implication of the theorem is the easiest. It is what will prove the algorithm to be correct. It only requires that

$$f(h) \geq h + 1.$$

The key idea to prove that a lot of consecutive wins is enough to win for arbitrarily large numbers of player is that a winning strategy makes use of so-called *pioneer plays*.

Definition 24 : Pioneer play

Let σ be a winning strategy, p a play of σ, and $n \in \mathbb{N}$. p is said to be a *pioneer play for n* if it is the play triggering the win when there are n players involved, and it depends on no wins for m players with $m > n$.

Please note that the notion of pioneer play depends on the strategy, and not only on the automaton.

Example 25 : Pioneer plays In the strategy described in example 13, pioneer plays are $p_3, p_5(1), p_5(2)$. In the alternative strategy

$$p_3; p_5(1); p_5(4); p_2(2); p_5(2); \ldots$$

whose wins word are described on the right, pioneer plays are $p_3, p_5(1), p_5(4)$. In this alternative strategy, $p_5(2)$ is **not** a pioneer play, because it wins for 7 players, but $p_5(4)$ was played prior to it and won for 9 players, which is larger than 7.

p_3	$001000000\ldots$
$p_5(1)$	$001001000\ldots$
$p_5(4)$	$001001001\ldots$
$p_2(2)$	$001101001\ldots$
$p_5(2)$	$001101101\ldots$
$\vdots$	$\vdots$

The notion of pioneer plays allows a smooth proof of correctness of algorithm 1.

Proof sketch of reverse implication of theorem 23 Winning for $h+1$ players can only be achieved by a pioneer play, and this pioneer play has a self-loop in its induced path. Progressively increase the number of self-loopings on it to win for any number of players. Those plays are not losing because of the consecutive wins achieved so far. ☐

4.2 Completeness

The other implication is the hard part. It is what will prove the algorithm to be complete.

An arbitrary winning strategy might be silly, and win for a number less than or equal to $f(h)$ using dependencies in a previous win against a number of players larger than $f(h)$.

Example 26 Consider the automaton depicted in figure 7. As in example 13, $p_i(n)$ denotes the play with n self-loopings on the first self-loop, and is winning for $i + n$ players.

To win with number of players 4, there exists a family of silly strategies (σ_n) where the play winning for 4 players depends on *each* previous play, with number of previous plays arbitrarily high, and the following chain of dependencies (where $i \to j$ denotes a win for j players, depending[a] on a win for i players).

$$\begin{tikzcd} 1,2 \arrow[r] & 5 \arrow[r] & 8 \arrow[r] & \cdots \arrow[r] & 3n+2 \arrow[r] & 3n+5 \arrow[d] \\ & 4 & 7 \arrow[l] & \cdots \arrow[l] & 3n+1 \arrow[l] & 3n+4 \arrow[l] \end{tikzcd}$$

A strategy σ_n achieving such dependency is:

```
1: Play $p_1$
2: for $j$ from 0 to $n$ do
3:     Play $p_5(3j)$
4: Play $p_3(3n+1)$
5: for $j$ from $n$ to 1 do
6:     Play $p_2(3j-1)$
```

However, despite this, there exists one simple strategy σ to win for 4 players, which is to play p_1, then $p_5(0)$ and $p_3(1)$.

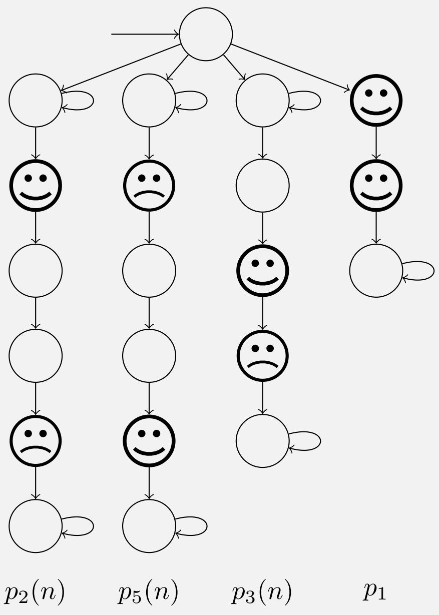


Figure 7: Example automaton. Action letters are omitted for clarity.

[a] Please note that in this paragraph, the first notion of dependencies is a dependency on previous *plays* (as in definition 16), and then it is the notion of dependencies on previous *wins* that is used (as in definition 14).

The aim of this section is to prove that an arbitrary strategy (including silly ones) can actually be modified in order not to depend on wins against a number of players larger than $f(h)$.

Subsections of this proof are the following. Their content is described in the table below.

Subsection	Contents
4.2.1	Proves that winning enough consecutive wins at a given position is a sufficient condition not to depend on farther wins.
4.2.2	Gives some properties of long paths in an automata, and gives more hypothesis for the hard case of the proof by getting rid of easy cases.
4.2.3	Explains how to, from a winning play, deduce some other winning plays.
4.2.4	Describes and proves the correctness of *pioneer play graphs*, a technical construction that will provide a proof in the next subsection.
4.2.5	Makes use of a pioneer play graph to build a periodic wins pattern in the strategy. This allows to think modulo the period, and thus truncates infiniteness in the number of players.
4.2.6	Fills the missing wins in the periodic pattern built in the previous section.

4.2.1 Backbone lemmata

The aim of subsection 4.2.1 is to establish that winning for a large enough number of consecutive players with no dependencies larger than $f(h)$ is a sufficient condition to prove the reverse implication.

A first step is to ensure that whenever a winning strategy wins for a large enough number of consecutive players, then one can modify the strategy in order to win for arbitrarily large numbers of players without depending on wins against even larger numbers of players. This is what ensures lemma 28, but we suggest first to take a look at example 27 to understand why this lemma is useful.

Example 27 : Silly winning strategy Consider the automaton on figure 8. $p(n)$ (resp. $p'(n)$) denotes the play going for the first (resp. second) row with n self-loopings on the first state; thus, both $p(n)$ and $p'(n)$ win for n players. One winning strategy σ could be :

```
1: Play $p(0)$
2: $i \leftarrow 0$
3: while $\top$ do
4:     Play $p(2^i)$
5:     for $j$ from $2^i - 1$ to $2^{i-1} + 1$ do
6:         Play $p'(j)$
7:     $i \leftarrow i + 1$
```

By following such a strategy, only plays that are powers of 2 do not depend on wins for larger numbers of players. However, there exists one winning strategy τ that never relies on wins for a larger number of players : playing $p(n)$ for n from 0 to infinity.

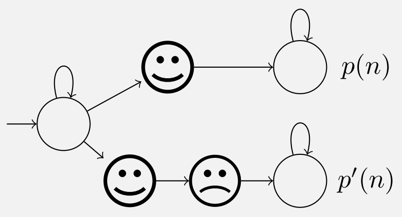


Figure 8: An example automaton.

τ needs no consecutive wins to start generating wins for an arbitrarily large number of players without dependencies in larger number of players. Lemma 28 aims at proving that for any winning strategy, one can replace strategies depending on wins for a larger number of players by strategies that do not, as τ in this example.

Lemma 28 ensures that winning for large enough consecutive numbers of players allows to modify a winning strategy in order not to depend on wins for larger numbers of players in order to win for increasingly larger numbers of players.

Lemma 28 : Winning window

Let σ be a winning strategy, and $(p_i)_{i>0}$ be a sequence of plays coherent with σ. Assume that there exist integers N and j such that when the number of players is between $N + 1$ and $N + 3h$, the vertex ☺ is among the first j visited vertices[a].
Then there exists one single path (containing a self-loop before a state in $F_\top$) that is sufficient to win for all numbers of players larger than $N + 3h$ from turn $j + 1$.

[a] In other words, the game is won before play $j + 1$ when there are between $N + 1$ and $N + 3h$ players.

Proof sketch

Case 1 : there exist a play σ plays whose final self-loop is not in $F_\circ$. This means that it plays at some previous point a play p whose final self-loop is in $F_\top$. After the window is complete, copy the beginning of p. When it enters the window, fork it and go straight to the final self-loop. Protection of the window allows not to care of losing states in the path.

Case 2 : plays of σ always end with a final self-loop in $F_\circ$. To win before this window : copy plays of σ winning before the window, but when they enter the window, fork them and use the protection given by the window to reach a terminal neutral self-loop.

To win after this window : use any pioneer play.

□

From now on we write $W(h) := 3h$ (W for *Window*).

Lemma 29 allows to make winning for small numbers of players not being dependent on winning for large numbers of players.

Lemma 29 : No distant dependencies

Let $N \in \mathbb{N}$, and σ be a winning strategy. Assume that when the number of players is between $N+1$ and $N+h$, the vertex ☺ is among the first i visited vertices[a]
Then there exists a winning strategy σ' where plays winning for a number of players $n \in [\![1, N]\!]$ after the i^{th} one do not depend on any win for a number of players larger than $N+h$.

[a] In other words, the game is won before play $i+1$ when there are between $N+1$ and $N+h$ players.

Proof sketch Copy plays of σ in order but truncate their ends thanks to the consecutive achieved wins. □

Lemmata 28 and 29 combined makes winning for $W(h)$ consecutive number of players with no dependencies larger than $f(h)$ a sufficient condition to prove the first implication of theorem 23. Proving that a winning game always admits a strategy satisfying such sufficient condition is the aim of the rest of section 4.2.

4.2.2 Structure of long paths

The aim of this section is to identify what is the structure of plays in an automaton winning for a very large number of players. Knowing this structure, we will be able to focus on hard cases of the proof of theorem 23 for the rest of section 4.2.

Lemma 30 allows us to solve the easy cases of section 4.2 where a winning strategy choses a pioneer play p that wins for infinitely many numbers of players, and gives stronger hypothesis in the other cases.

Lemma 30 : Winning paths final state

Let σ be a winning strategy, i an integer, and ρ the path of the i^{th} play of σ. Let q_{final} be the last state ρ is visiting. One of the following holds :

1. q_{final} is in $F_\circ$, or
2. There exists $N \in \mathbb{N}$ such that for all numbers of players $M > N$, when there are M players involved, either σ wins before playing ρ or by playing ρ.

Proof To win for any number of players with a finite number of plays, as described in assumption 2, a strategy must go for a play ending in a $F_\top$ state. Thus, 2 holds as long as either ρ ends with a state in $F_\top$, or a previous play ends in a state in $F_\top$. As σ is winning, this must be the case if ρ ends with a state in $F_\perp$. If ρ ends with a state neither in $F_\top$ nor in $F_\perp$, then 1 holds. □

Next lemma ensures that when there are enough vertices on a path ρ followed by a winning strategy, either there is a self-looping vertex in $F_\circ$ on this path, or ρ either needs or generate $W(h)$ consecutive wins.

Lemma 31 : Long paths loops

Let σ a winning strategy, and ρ the path of a play such that its $hW(h)^{th}$ state q is not its final state. One of the following holds :

1. There is on ρ a self-looping state in $F_\circ$, or
2. ρ needs $W(h)$ consecutive wins prior to q made by previous plays, or
3. ρ generates $W(h)$ consecutive wins prior to q.

In particular, in case 2, as σ is winning, it means that $W(h)$ consecutive wins have been previously generated.

Proof Only self-looping states can appear multiple times on ρ. If the $hW(h)^{th}$ state q is not final, then one self-looping state q_{loop} repeats at least $W(h)$ time before q. If $q_{\text{loop}} \in F_\circ$, 1 holds. If it is in $F_\perp$, as σ is winning, ρ requires $W(h)$ consecutive wins prior to being played, so 2 holds. If it is in $F_\top$, then 3 holds. □

From now on we write $L(h) := hW(h)$ (L for *Long*).

Figure 9 illustrates what the following paragraph explains.

Lemma 31 allows us to assume that if a play p wins for $n \geq L(h)$ players, and $W(h)$ consecutive wins is not fulfilled after playing p, then there is a self-looping state in $F_\circ$ on the path induced by p. Name q_{loop} the last self-loop in $F_\circ$ before q_n the n^{th} state of the path. By using lemma 31 again with the path starting at q_{loop}, there are at most $L(h)$ states between q_{loop} and q_n, because there are no other self-looping state in $F_\circ$ between q_{loop} and q_n, and it is an hypothesis that playing p does not generate $W(h)$ consecutive wins. It is an important hypothesis to enforce next lemmata. Also, lemma 30 gives that the final state of all such paths are neutral.

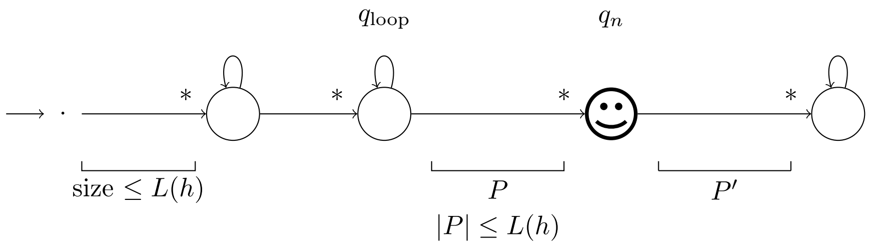


Figure 9: Some plays winning to more than $L(h)$ players follow paths of this form.

From now on we add a formal definition of strategies such that all paths of interest are of the form described in figure 9.

Definition 32 : Non-trivial strategy

Let σ be a winning strategy. It is a *non-trivial strategy* if all plays winning for n players with $L(h) \leq n \leq f(h)$ have each of the following properties :

1. they have a self-loop in $F_\circ$ visited before their $L(h)^{th}$ state, and
2. they end with a state in $F_\circ$, and
3. between the last state in $F_\top$ and the last self-loop in $F_\circ$ before it, there are at most $L(h)$ (non-necessarily different) visited states.

Until the very end of section 4.2, we will only consider non-trivial strategies, because a strategy that is not non-trivial will easily ensure completeness, as we will see in the proof of theorem 23 at the end of section 4.2.

4.2.3 Duplicating winning plays

The aim of this section is to explain how, from some given winning plays played by a non-trivial winning strategy, deduce some other winning plays.

For the rest of section 4.2.3, fix τ any non-trivial winning strategy playing infinitely many pioneer plays. This seems a quite restrictive hypothesis, but it happens to be a finer definition of the hard case of the proof. We will see in the proof of the first implication of theorem 23, in subsection 4.2.6, that if any of those hypothesis doesn't hold, proof is much shorter.

Next ideas are a bit more tricky to understand. The main plan is the following. As τ is winning, it will eventually win for $W(h)$ consecutive number of players. However, dependencies of such consecutive wins might include a number of players larger than $f(h)$. In such case, the loopless path induced by plays of τ will allow to build a new strategy achieving $W(h)$ consecutive wins without dependencies in number of players greater than $f(h)$.

When following τ, consider p_n any pioneer play winning for n with $n \geq L(h)$. Let ρ_n be the induced path of p_n. Call q_n the state in $F_\top$ that is the n^{th} state of p_n, and q^n_{loop} the state of ρ before q_n, in $F_\circ$ and with a self-loop on it, that is the closest to q_n. As described by lemma 31, p_n is of the form described in figure 9, its n^{th} winning state being the one between P and P'.

The trick is that one can generate new different plays by playing along ρ_n, but changing the amount of self-loopings on q_{loop}, in order to shift the number of players for which the play is winning. However, by doing so, dependencies in the P and P' part will also shift. Half of the problem is already solved because as p_n is a pioneer play, it has no state in $F_\perp$ in its P' part by design. Thus, if the amount of self-loopings on q_{loop} changes, the dependencies we will actually have to care about are only the ones in the P part. This is our main concern until the end of section 4.2.

To describe what are the dependencies needed to play a path of the form described in figure 9, one can use boolean words of size (at most) $L(h)$. Reading from right to left, the i^{th} letter is a 1 if and only if the i^{th} state before q_n is in $F_\perp$. This implies that if the i^{th} state before q_n is the j^{th} state played, the game must have been previously won for j players. Figure 10 illustrates this idea with a word of length much shorter than $L(h)$.

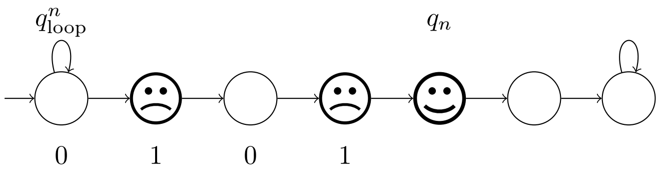


Figure 10: Consider the play consisting in visiting exactly once each vertex on the illustrated graph. A sequence of bit of length 4 describing the dependencies of q_n is 0101.

More precisely, we will pay attention to dependencies *between* q^n_{loop} and q_n, as they are the only ones that will shift. Knowing this, we can build the following proof scheme :

- Consider each pioneer play for n where $3L(h) \leq n \leq f(h)$. This bound of $3L(h)$ rather than $L(h)$ allows not to consider limit cases in the transition from "not a long path" to "long path";
- Keep track of their dependencies, in the form of a word in $\{0,1\}^{L(h)}$;
- If two different plays p_i and p_j have the same dependencies, repeating the sequence of plays $p_i \dots p_j$ can achieve pioneer plays for arbitrarily high n.

It is the aim of subsection 4.2.5 to describe precisely this proof scheme. It relies on a so-called *pioneer play graph*, whose definition and correctness are given in subsection 4.2.4.

4.2.4 Pioneer play graph

The aim of this section is to model the sequence of pioneer plays described in subsection 4.2.3 with a path in a "De Bruijn-like" graph [6], where vertices are words of $\{0,1\}^{L(h)}$, and edges are pioneer plays. Informally, the meaning of an edge from v to w is "There exist a pioneer play that can be played from a v pattern, and from which a w pattern can be reached farther". From now on, we define

$$L'(h) := 3L(h).$$

The notion of pioneer resources described below is a **different** notion from what we named dependencies in subsection 4.2.3. The pioneer resources of a pioneer play describes some sufficient condition for it to be played, where its dependencies described a necessary condition for it to be played.

Definition 33 : Pioneer resources

Let σ be a non-trivial winning strategy, and PP_∞ be the set of pioneer plays for n played by σ, where $n \geq L'(h)$. Let $p \in PP_\infty$, and w the wins word right before playing p. The *pioneer resources* of p is the factor of w of size $L(h)$ whose last letter is the $(m-1)^{th}$ letter of w, where m is the maximum number of players for which p is winning.

Why considering pioneer resources and not just dependencies of pioneer plays ? Sometimes, between two consecutive pioneer plays p and p', some wins that are not dependencies of p' might be achieved by non-pioneer plays. Considering that the dependencies of p are enough to fulfill the dependencies of p' would be true, but would forget all of those intermediate wins achieved by non-pioneer plays. Considering that dependencies of p can fulfill resources of p' is a way to capture all wins achieved in between by non-pioneer plays, even though those wins might not be dependencies of p'.

Example 34 : Pioneer resources To illustrate the notion of pioneer resources, one ignores for the sake of clarity the two conditions of using words of size $L(h)$, and of considering pioneer plays **for $\mathbf{n \geq L'(h)}$**. Consider again example 13. $p_5(1)$ and $p_5(2)$ are pioneer plays. Their pioneer resources are 00100 and 001101. Should $p_5(3)$ be played right after p_1, its pioneer resources would be 1111111.

We are now armed to define vertices of our pioneer play graph. Next step is to know how to define edges.

The following definition describes triples (r_1, n, r_2). Informally, they mean that going from resources r_1, there exist a pioneer play that can achieve a win for a number of players increased by n (in other words, a shift of n in the wins word) and still reach pioneer resources r_2.

Definition 35 : Pioneer transition

Let σ be a non-trivial winning strategy, and p a pioneer play for n played by σ, where $n \geq L'(h)$[a]. Assume that p is the i^{th} pioneer play played by σ, and consider q the $(i+1)^{th}$ pioneer play it plays, which is a pioneer play for $n + n_q$.
Let r_p (resp. r_q) the pioneer resources of p (resp. q). The *pioneer transition* $PT(p)$ for p is the transition $(r_p, n_q, r_q) \in \{0,1\}^{L(h)} \times \mathbb{N} \times \{0,1\}^{L(h)}$.

[a] Pioneer transitions are undefined when $n < L'(h)$.

The ideas behind this definition is that if a pioneer play can be played under some resource condition, it can be played again under the exact same resource conditions.

Example 36 : Pioneer transition Consider example 13 again, and assume $p_5(3)$ is played right after p_1. As in example 34, forget conditions on size of words and of being pioneer for a large number of players. Then $PT(p_5(1)) = (00100, 3, 001101)$ and $PT(p_5(2)) = (001101, 1, 1111111)$.

Lemma 37 ensures that the definition of pioneer transition is of use. Figure 11 illustrates it.

Lemma 37 : Pioneer transitions to pioneer plays

Let σ be a non-trivial winning strategy, and $e = (y, k, z) = PT(p)$ an edge of the pioneer play graph of σ. Assume the i^{th} wins word of σ is of the form

$$wy0^\omega$$

where $wy \in \{0,1\}^{\geq L'(h)}$, and $y = y_1 y_2$ with $|y_1| = k$. Then one can insert right after the i^{th} play of σ a pioneer play followed by some other plays resulting in a wins word, larger than or equal, according to pointwise order, to

$$wy_1z0^\omega$$

and keep the property that this new strategy is winning.

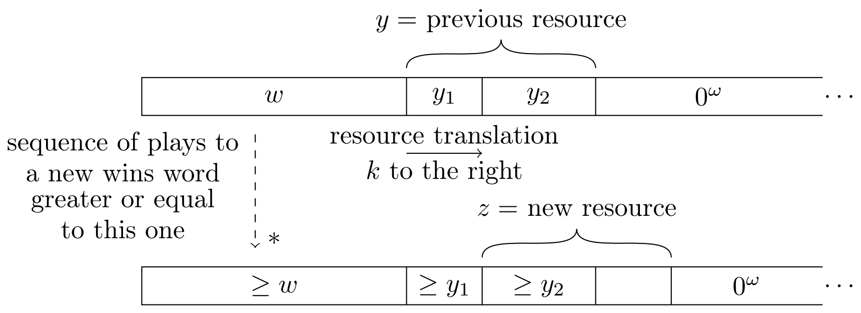


Figure 11: Meaning of lemma 37.

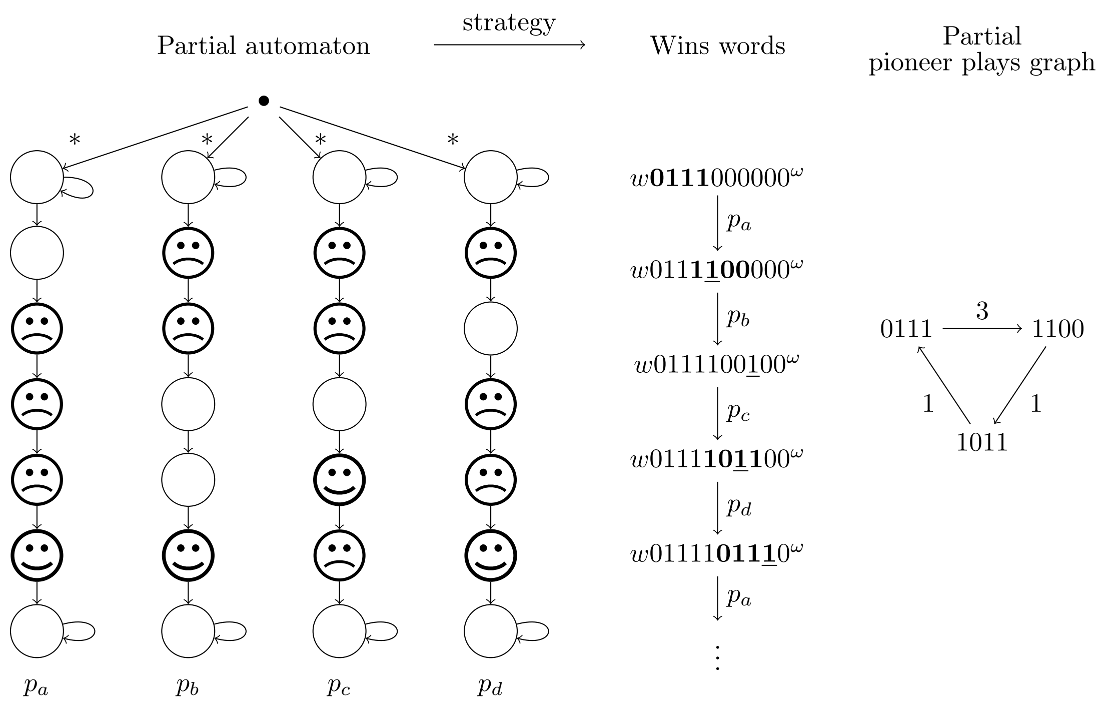
Proof sketch If $e = (y, k, z) = PT(p)$ is in the pioneer play graph of σ, it means that when p is played by σ, it has resources y and can reach with a sequence of plays farther resources z. Copy this sequence of plays but with a number of self-looping on their self loop modified, so that indices of resource of p when p is played are shifted to indices of resource y in the word $wy0^\omega$. $\square$

Definition 38 : Pioneer plays graph

Let σ be a winning strategy, and PP_∞ the set of pioneer plays of σ for n where $n \geq L'(h)$. The *pioneer plays graph* of σ is a graph (V, E) where :

- $V = \{0,1\}^{L(h)}$
- $E = \bigcup_{p \in PP_\infty} PT(p)$ where $PT(p)$ is the pioneer transition for p.

Part of the construction of a pioneer plays graph is depicted in figure 12.



Partial automaton
strategy
Wins words
Partial pioneer plays graph
*
p_a
p_b
p_c
p_d
$w\mathbf{0111}000000^\omega$
$w0111\mathbf{\underline{1}100}000^\omega$
$w0111100\underline{1}00^\omega$
$w0111\mathbf{1}\mathbf{01}\underline{1}\mathbf{1}00^\omega$
$w01111\mathbf{0111}\underline{1}0^\omega$
0111
1100
1011
3
1
1

Figure 12: How a pioneer plays graph is built and used. Words described in the wins graph are shorter than $L(h)$ but sufficient for clarity. On the left is part of an automaton. In the middle is a sequence of plays in this automaton, progressively increasing the wins word. On the right is part of the pioneer plays graph one can deduce from this sequence of plays.
Underlined bits are the ones added by previous play. Bold bits are the ones that will be considered for the pioneer plays graph. In the sequence of wins word, they are located in wins word right before a pioneer play. Inside a wins word, they are located right before the next (underlined) bit added by the next pioneer play.

4.2.5 Generating periodic pioneer wins

See in figure 12 that the sequence of plays $p_a; p_b; p_c; p_d$ can be repeated to generate periodic wins. Lemma 39 ensures that one can always find such sequence.

Lemma 39 : Generating periodic wins

Let σ be a non-trivial winning strategy such that if a pioneer play is winning for $n < f(h)$, the next pioneer play winning for m is such that $n + L(h) \geq m$.
Assume that from all vertex v **in the pioneer plays graph**, there is a path of length at most k visiting some vertex v' twice.
Then one can insert plays in σ to generate wins for numbers of players periodically increasing to infinity through pioneer plays, with a period of less than $kL(h)$.

Note that in any pioneer play graph $G = (V, E)$, any path of length at least $|V| + 1$ meets some vertex twice.

Proof Consider r the pioneer resource of p the first pioneer play for n where $n \geq L'(h) = 3L(h)$. r is a vertex of the pioneer plays graph of σ. Let ρ be a path in the pioneer plays graph of length at most k with a cycle in it. It exists by hypothesis.
After p, apply successively lemma 37 to all edges in the path, to deduce new plays to add to the strategy. This process can be repeated an arbitrarily long time, as there is a cycle in ρ.
As pioneer wins are distant of at most $L(h)$ number of players by hypothesis, and there is a cycle in ρ, the period is at most $kL(h)$. □

From now on, we denote by $C(h)$ the minimum k such that from all vertex v in the pioneer play graph of an arbitrary winning strategy, there is a path of length at most k meeting some vertex twice. As the pioneer plays graph has $2^{L(h)}$ vertices, we know that $C(h) \leq 2^{L(h)} + 1$. Lemma 39 is true for $k := C(h)$ by design.

4.2.6 Generating enough consecutive wins with no larger dependencies

A strategy achieving pioneer wins with a period of P allows to think about wins for large numbers of players modulo P.

Lemma 40 : From periodic wins to consecutive wins

Let σ be a non-trivial winning strategy, verifying the same hypothesis as in lemma 39. Let P be the period of the periodic wins given by this lemma.
One can build a new strategy τ by inserting plays such that τ wins for $W(h)$ consecutive number of players without any such play depending on a win for a number of players larger than

$$3L(h) + 2P^2 + W(h).$$

Proof sketch Think modulo P. Add enough periodic patterns. Then use $q_1, \dots, q_K$ plays of σ winnings for different values modulo P but shifted to fill the holes in the added periodic patterns.

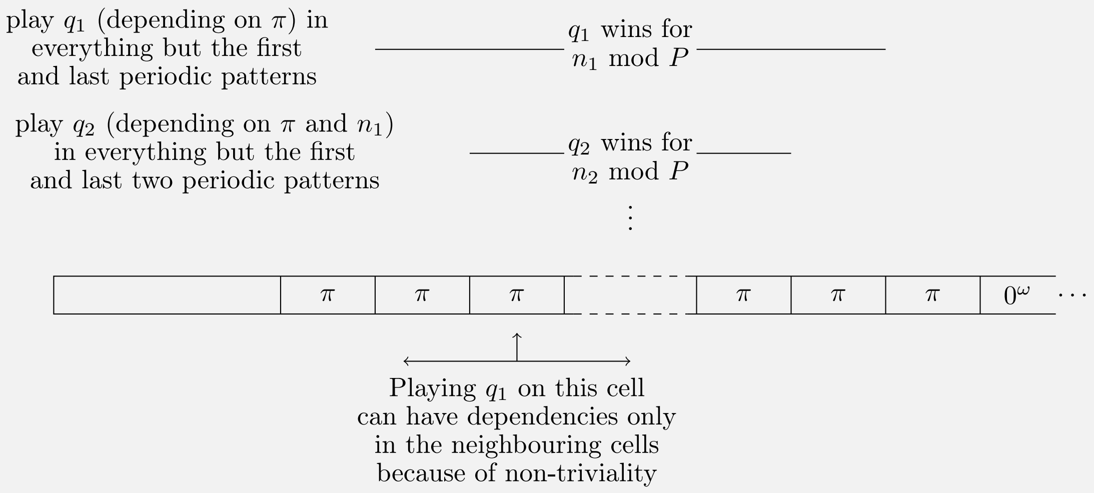

play q_1 (depending on π) in everything but the first and last periodic patterns
q_1 wins for n_1 mod P
play q_2 (depending on π and n_1) in everything but the first and last two periodic patterns
q_2 wins for n_2 mod P
π
0^ω
Playing q_1 on this cell can have dependencies only in the neighbouring cells because of non-triviality

Figure 13: How to fill missing wins in periodic pattern. □

Now we are armed to prove the hard implication of our main theorem.

Restate of theorem 23

Let $\mathcal{A}$ be a tricolored flat arena, and A be a joint automaton describing the outgoing edges of the ◯ state in $\mathcal{A}$. Assume A has depth h.
There exists a cooperative strategy σ winning whatever the number of players involved if and only if there exists a cooperative strategy σ' such that :

1. σ' wins for any number $n \leq f(h)$ of player;
2. no play of σ' winning for n players with $n \leq f(h)$ requires that the game has been won by a previous play for m players with $m > f(h)$.

Proof sketch of first implication Let σ be a winning strategy. If it is non-trivial, by lemma 40, we're done by simply applying it, as long as

$$f(h) \geq 3L(h) + 2C(h)^2 + W(h).$$

If not, any of the hypothesis of σ that can make it not non-trivial allows to easily prove the theorem. □

This concludes that theorem 23 is true with

$$f(h) = 3L(h) + 2C(h)^2 + W(h).$$

where $L(h) = hW(h) = 3h^2$, and $C(h) \leq 2^{L(h)} + 1$. This simple bound on $C(h)$ is enough to show that PCRP IN TRICOLORED ARENA WITH JOINT FLAT AUTOMATON is in NEXPTIME.

References

[1] Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. “Concurrent Parameterized Games”. In: *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*. Ed. by Arkadev Chattopadhyay and Paul Gastin. Vol. 150. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 31:1–31:15. ISBN: 978-3-95977-131-3. DOI: 10.4230/LIPIcs.FSTTCS.2019.31. URL: https://drops.dagstuhl.de/opus/volltexte/2019/11593.

[2] Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. “Synthesizing Safe Coalition Strategies”. In: *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*. Ed. by Nitin Saxena and Sunil Simon. Vol. 182. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 39:1–39:17. ISBN: 978-3-95977-174-0. DOI: 10.4230/LIPIcs.FSTTCS.2020.39. URL: https://drops.dagstuhl.de/opus/volltexte/2020/13280.

[3] Dietmar Berwanger, Lukasz Kaiser, and Bernd Puchala. “A Perfect-Information Construction for Coordination in Games”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*. Ed. by Supratik Chakraborty and Amit Kumar. Vol. 13. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 387–398. ISBN: 978-3-939897-34-7. DOI: 10.4230/LIPIcs.FSTTCS.2011.387. URL: http://drops.dagstuhl.de/opus/volltexte/2011/3335.

[4] Roderick Bloem et al. “Decidability in parameterized verification”. In: *ACM SIGACT News* 47.2 (2016), pp. 53–64.

[5] Tristan Charrier et al. “Reachability and Coverage Planning for Connected Agents”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 144–150. DOI: 10.24963/ijcai.2019/21. URL: https://doi.org/10.24963/ijcai.2019/21.

[6] Nicolaas Govert De Bruijn. “A combinatorial problem”. In: *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam* 49.7 (1946), pp. 758–764.

[7] Giorgio Delzanno. “Constraint-based verification of parameterized cache coherence protocols”. In: *Formal Methods in System Design* 23 (2003), pp. 257–301.

[8] Javier Esparza. “Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk)”. In: *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Ed. by Ernst W. Mayr and Natacha Portier. Vol. 25. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 1–10. ISBN: 978-3-939897-65-1. DOI: 10.4230/LIPIcs.STACS.2014.1. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4498.

[9] Igor Konnov, Helmut Veith, and Josef Widder. “What You Always Wanted to Know About Model Checking of Fault-Tolerant Distributed Algorithms”. In: *Perspectives of System Informatics*. Ed. by Manuel Mazzara and Andrei Voronkov. Cham: Springer International Publishing, 2016, pp. 6–21. ISBN: 978-3-319-41579-6.

[10] Swarup Mohalik and Igor Walukiewicz. “Distributed Games”. In: *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*. Ed. by Paritosh K. Pandya and Jaikumar Radhakrishnan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 338–351. ISBN: 978-3-540-24597-1.

[11] Gary L. Peterson and John H. Reif. “Multiple-person alternation”. In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 348–363. DOI: 10.1109/SFCS.1979.25.

[12] A. Pneuli and R. Rosner. “Distributed reactive systems are hard to synthesize”. In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 746–757 vol.2. DOI: 10.1109/FSCS.1990.89597.

Appendix

Algorithm to solve Pcrp in Tricolored Flat Arena

Definition 41 : Safe play

Let S be an array of $\bot$ and $\top$ indexed from 1 to N, and si[a]$\in \{\top, \bot\}$. $p = a_1 \dots a_k$ is a *safe play* according to S and si if the two following properties hold.

1. $\forall i \leq k,\ p|_{\leq i} \in L_\bot \implies i \leq N \wedge S[i] = \top$
2. $p|_{\leq k} \in L_\bot \implies si = \top \wedge \forall i \in [\![k, N]\!], S[i] = \top$

[a] S stands for *safe*, and si stands for *safe infinity*.

Algorithm 2 Solving Pcrp in Tricolored Flat Arena

Input: A triple of regular expressions $(e_\top, e_\bot, e_\circ)$ describing a tricolored flat arena $\mathcal{A}$. Their languages are $(L_\top, L_\bot, L_\circ)$.
Output: Is there a cooperative winning strategy in $\mathcal{A}$?

```
1: Build $A_\top, A_\bot, A_\circ$ nondeterministic automata recognizing $L_\top, L_\bot, L_\circ$.
2: $h \leftarrow$ the maximum height of $A_\top, A_\bot$ and $A_\circ$.
3: $S \leftarrow$ array of $\bot$ indexed from 1 to $f(h)$          ▷ $f$ is the function of theorem 23.
4: $si \leftarrow \bot$                                              ▷ $si$ for safe infinity.

5: procedure Update_S($p$)                                            ▷ $p = a_1a_2 \dots a_k^\omega$ is a safe play.
6:     for $i$ from 1 to $k$ do
7:         Guess one language between $L_\top, L_\bot$ or $L_\circ$
8:         Check that $p|_{\leq i}$ is in it by nondeterministically using the relevant automaton
9:         if $p|_{\leq i} \in L_\top$ then
10:            $S[i] \leftarrow \top$
11:    if $p|_{\leq k} \in L_\top$ then
12:        $si \leftarrow \top$
13:        for $k \leq i \leq |S|$ do
14:            $S[i] \leftarrow \top$

15: while true do
16:     Find a safe play $p$ of length less than $f(h) + h$ such that Update_S($p$) sets at least one $\bot$ of $S$ to $\top$.
17:     Update_S($p$)
18:     if $\forall 1 \leq i \leq |S|,\ S[i] = \top$ then
19:         return true
20:     if neither $S$ nor $si$ changed inside this while loop iteration then
21:         return false
```

Algorithm 2 is correct because algorithm 1 is.

Detailed proofs

Restate of lemma 5 : Joint automaton

For every triple of flat languages $L_\top, L_\bot$ and $L_\circ$ such that $L_\circ \sqcup L_\bot \sqcup L_\top = \Sigma^*$, there exists a structure $(\Sigma, Q, \delta, q_0, F_\top, F_\bot, F_\circ)$ with $F_\top \sqcup F_\bot \sqcup F_\circ = Q$ such that for every $\Delta \in \{\top, \bot, \circ\}$, $\mathcal{A}_\Delta = (\Sigma, Q, \delta, q_0, F_\Delta)$ is a deterministic flat automaton that recognizes L_Δ. We call this structure a *joint automaton*.

Proof Let $\mathcal{A}_\top, \mathcal{A}_\bot$ and $A_\circ$ three flat automaton recognizing languages $L_\top, L_\bot$ and $L_\circ$. Consider $\mathcal{A} = (\Sigma, Q, \delta, q_0)$ the product automaton (without final state) of $\mathcal{A}_\top, \mathcal{A}_\bot$ and $A_\circ$; so Q is made of triple of states of $\mathcal{A}_\top, \mathcal{A}_\bot$ and $A_\circ$. As $L_\top, L_\bot$ and $L_\circ$ are partitioning Σ^*, each state in Q has exactly one

component of its triple of states that is final in its original automaton. Label states of $\mathcal{A}$ accordingly with $F_\top, F_\bot$ and $F_\circ$ to get an automaton as described in the lemma statement.
Furthermore, as $\mathcal{A}$ is a product of flat automata, it also is one. □

Restate of theorem 9

PCRP IN TRICOLORED ARENA with flat regular expressions is NP-hard **when negation can be used** inside the regular expression it takes as an input.

Proof Reduction from a variant of 3-SAT where all clauses have exactly 3 literals, and no clause contains two literals corresponding to the same variable. Let

$$\varphi = \bigwedge_{i=1}^{n} (l_{i_1} \vee l_{i_2} \vee l_{i_3})$$

an input of this variant of 3-SAT, where variables are v_i's indexed from 1 to a given $N \in \mathbb{N}$. We consider w.l.o.g. that $i_1 < i_2 < i_3$. One consider the tricolored arena whose inputs are the following actions set and regular expressions :

$$\Sigma = \mathbb{B} := \{0, 1\} \qquad e_\circ = \emptyset$$

$$e_\bot = \bigvee_{i=1}^{n} \mathbb{B}^{\alpha_{i_4}} \bar{b}_{i_1} \mathbb{B}^{\alpha_{i_5}} \bar{b}_{i_2} \mathbb{B}^{\alpha_{i_6}} \bar{b}_{i_3} \mathbb{B}^{\alpha_{i_7}} \qquad e_\top = \neg L_2$$

where :

- In the word
$$w_i = \mathbb{B}^{\alpha_{i_4}} \bar{b}_{i_1} \mathbb{B}^{\alpha_{i_5}} \bar{b}_{i_2} \mathbb{B}^{\alpha_{i_6}} \bar{b}_{i_3} \mathbb{B}^{\alpha_{i_7}},$$
α's are such that boolean $\bar{b}_j$ is the j^{th} letter of w_i, and
$$\forall i \leq n, \alpha_{i_4} + \alpha_{i_5} + \alpha_{i_6} + \alpha_{i_7} + 3 = N;$$
- b_j is the boolean 0 if and only if l_j is a negative literal;
- The $\bar{\cdot}$ function matches 0 to 1 and 1 to 0.

The idea is that words of length N will encode truth functions.

3-SAT returns *true* on $\varphi \implies$ PCRP returns *true* on the input described above There exist a truth function τ such that $\tau(\varphi) = \top$. Consider the strategy consisting in one single play

$$p = \tau(v_1 \dots v_N)0^\omega$$

Then as τ values any clause to $\top$, p is not in the language of $e_\bot$, so it is in the language of $e_\top$.

3-SAT returns *true* on $\varphi \impliedby$ PCRP returns *true* on the input described above Let p the first play of an existing winning strategy. Consider strategy τ such that $\tau(v_i) = p_i$. As p is a play of a winning strategy, it is not in the language of $e_\bot$. Thus τ evaluates no clause to $\bot$.

As this reduction is clearly polynomial (even linear), this concludes the proof of theorem 9. □

Restate of theorem 10

PCRP IN TRICOLORED ARENA with regular expressions is PSPACE-hard **when negation can be used** inside the regular expression it takes as an input.

Proof Consider the well-known PSPACE-complete UNIVERSALITY problem :

INPUT: A regular expression e on alphabet Σ_e
OUTPUT: Are all words in the language of e ?

The output of UNIVERSALITY is the same as : "Is the complementary of the language of e empty ?". As PSPACE = CO-PSPACE, we design a reduction from NON-UNIVERSALITY the complementary problem of UNIVERSALITY, whose output is : "Is the complementary of the language of e non-empty ?".

The main idea is to build an equivalence between "The complementary language of e is not empty" and "There exist a strategy winning for at least one number of players". We achieve this first equivalence with the arena described by the following actions et and regular expressions :

$$\Sigma = \Sigma_e \qquad e_\circ = e$$
$$e_\bot = \emptyset \qquad e_\top = \neg e$$

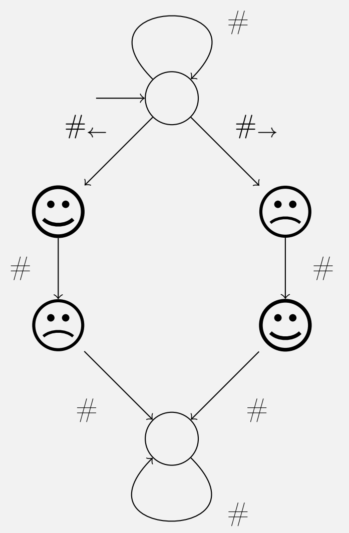


Figure 14: A joint automaton that can win for any number of players from one single win.

From there, we add a way to win for each player that can be used only when there is at least one win. This can for example be achieved by automaton described in figure 14. From a win for n players, this automaton can achieve a win for $n-1$ players with its left branch and a win for $n+1$ players with its right branch.

In term of inputs of PCRP, it can be performed by the followings :

$$\Sigma = \Sigma_e \sqcup \{\#, \#_\leftarrow, \#_\rightarrow\} \qquad e_\circ = e \vee \#^* \vee \#^*(\#_\leftarrow \vee \#_\rightarrow)\#^+$$
$$e_\bot = \#^*(\#_\leftarrow\# \vee \#_\rightarrow) \vee L_{mix} \qquad e_\top = \neg e \vee \#^*(\#_\leftarrow \vee \#_\rightarrow\#)$$

where :

- $\#, \#_\leftarrow$ and $\# \rightarrow$ are fresh letters;
- L_{mix} describes the language of words with at least one letter of Σ_e and one of $\{\#, \#_\leftarrow, \#_\rightarrow\}$:
$$L_{mix} = \Sigma^*\Sigma_e\Sigma^*\{\#, \#_\leftarrow, \# \rightarrow\}\Sigma^* \vee \Sigma^*\{\#, \#_\leftarrow, \#_\rightarrow\}\Sigma^*\Sigma_e\Sigma^*$$

NON-UNIVERSALITY returns *true* on $e \implies$ PCRP returns *true* on the described input
There exist a word w in the complementary language of e. Consider as a first play any infinite word of Σ_e^ω with prefix w. This play is winning for $|w|$ players, and safe for any other number of player.
From there, play words of the form
$$\#^*\#_\leftarrow\#^\omega$$
whose first prefix of # ranges from $|w|-2$ to 0. Those words are safe, and winning for $|w|-1$ to 1 players. After they are played, play words of the form
$$\#^*\#_\rightarrow\#^\omega$$
whose first prefix of # ranges from $|w|-1$ to ∞. Those words are safe, and winning for $|w|+1$ to ∞ players.

NON-UNIVERSALITY returns *true* on $e \impliedby$ PCRP returns *true* on the described input
If there is a winning strategy, its first winning can not have any prefix in $L_\bot$. The only possible such plays, when winning for n players, are the ones admitting a prefix of length n in the complementary language of e.

□

A winning strategy either wins for infinitely many players with one single play, or plays infinitely many pioneer plays, as lemma 42 points out.

Lemma 42 : Number of pioneer plays

Let σ be a winning strategy. One of the two following properties hold :

- One play of σ ends with a self-looping play in $F_\top$;
- σ plays infinitely many pioneer plays.

Proof As σ is a winning strategy, it wins for any number of players. If none of its plays ends with a state in $F_\top$, all of its plays win for finitely many different numbers of players. Thus, at any given time of a run, there exists a maximum number of players M such that σ has already won for this number of players. The next of its plays winning for a number of players larger than M will be a pioneer play. □

Restate of lemma 28 : Winning window

Let σ be a winning strategy, and $(p_i)_{i>0}$ be a sequence of plays coherent with σ. Assume that there exist integers N and j such that when the number of players is between $N+1$ and $N+3h$, the vertex ☺ is among the first j visited vertices[a].
Then there exists one single path (containing a self-loop before a state in $F_\top$) that is sufficient to win for all numbers of players larger than $N+3h$ from turn $j+1$.

[a] In other words, the game is won before play $j+1$ when there are between $N+1$ and $N+3h$ players.

Proof As σ is winning, one can go for a case disjunction according to lemma 42 :

Case 1 At some point, σ follows a path ρ whose last state q_{final}, on which there is a self-loop, is in $F_\top$. In this case, ρ satisfies the hypothesis we're looking for, because when the game is won for $[\![N+1, N+3h]\!]$ players, ρ can be used to win for any large number of players the following way:

1. Follow ρ as when it's played by σ until the $N+1$ state of the path;
2. Then stop following intermediate self-loopings of ρ, and instead go straight to q_{final}.

Part 1 of the path is not losing, as it is played by a winning strategy. Part 2 neither, because from any state of the path, q_{final} can be reached in less than h transitions, and the game is already won when there are $n \in [\![N+1, N+h+1]\!]$ players.

Case 2 σ plays infinitely many pioneer plays. In this case, consider the path ρ induced by the first pioneer play p winning for a number of players $n > N+3h$. ρ satisfies the following :

- The n^{th} state on this path, called $q_\top$, is in $F_\top$;
- There is a state q_l between the $N+1^{th}$ and $N+h+1^{th}$ state of ρ admitting a self-loop, because of the pigeonhole principle. $q_\top$ is distant of at most h states from q_l;
- There is a state q_{final} that can be reached from $q_\top$ in at most h states;
- There are no state in $F_\perp$ between $q_\top$ and q_{final}, because p is a pioneer play for n. (†)

ρ can be used to win for any large number of players in the following way:

1. Follow ρ (as done by σ) until reaching q_l;
2. Self-loop on q_l an amount of time such that right after self-looping, by going straight to $q_\top$, it is the $N+3h+1$ state met on the path;
3. Then end the path by simply going from $q_\top$ to q_{final}.

This path is safe. Part 1 because it's played by a winning strategy, part 2 because of the previous consecutive wins, and part 3 because of (†) hypothesis. Also, this path is winning for $N+3h+1$ players. One can then repeatingly using this path by progressively incrementing the number of times it is looping on q_l to progressively win for larger number of players.

□

Restate of lemma 29 : No distant dependencies

Let $N \in \mathbb{N}$, and σ be a winning strategy. Assume that when the number of players is between $N+1$ and $N+h$, the vertex ☺ is among the first i visited vertices[a]
Then there exists a winning strategy σ' where plays winning for a number of players $n \in [\![1, N]\!]$ after the i^{th} one do not depend on any win for a number of players larger than $N+h$.

[a] In other words, the game is won before play $i+1$ when there are between $N+1$ and $N+h$ players.

Proof Consider p_i the first play after which the game is won when there are between $N+1$ and $N+h$ players. From then, follow as long as it is possible this procedure :

1. Consider the smallest $j > i$ such that the j^{th} play p_j wins for some $n \leq N$ number of players.
2. Build p'_j from p_j the following way :
   (a) Copy p_j until its N^{th} state;
   (b) Then go straight to the last self-loop of p_j.
3. Play p'_j right before p_j.

The only case where self-looping to infinity is not safe is if this self-loop is in $F_\perp$. If so, as it is the same self-loop as the *safe* play p_j, it means that a play ending with a self-looping state in $F_\top$ has been met prior to p_j. Let $p_\top$ be such a play. Apply to $p_\top$ transformation of step 2 described to get $p'_\top$, and insert it before p'_j. $p'_\top$ is winning for all numbers of players larger than $N+h$, and thus makes self-looping on the final state of p'_j safe.
Part 2a of p'_j is safe if played right before p_j, because p_j is safe. Part 2b is also safe until the self-loop, because any state can reach a self-loop at most h states away, and the game is won if there are between $N+1$ and $N+h$ players. Finally, self-looping to infinity can be made safe. Hence p'_j wins for n ans satisfies the conditions of the statement. $\square$

Restate of lemma 37 : Pioneer transitions to pioneer plays

Let σ be a non-trivial winning strategy, and $e = (y, k, z) = PT(p)$ an edge of the pioneer play graph of σ. Assume the i^{th} wins word of σ is of the form

$$wy0^\omega$$

where $wy \in \{0,1\}^{\geq L'(h)}$, and $y = y_1y_2$ with $|y_1| = k$. Then one can insert right after the i^{th} play of σ a pioneer play followed by some other plays resulting in a wins word, larger than or equal, according to pointwise order, to

$$wy_1z0^\omega$$

and keep the property that this new strategy is winning.

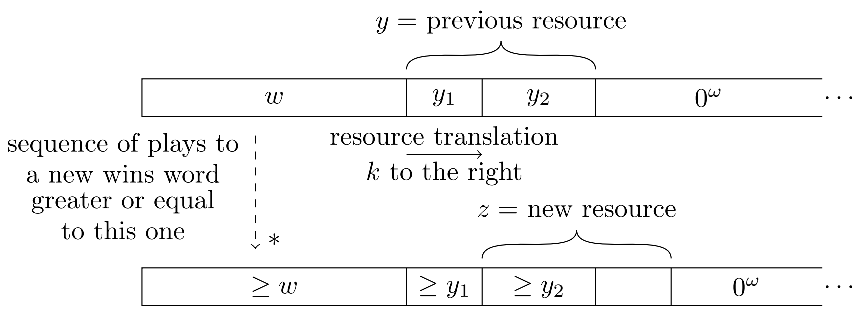


Figure 15: Meaning of lemma 37.

Proof Throughout this proof, references to figure 16 will appear in gray to help understanding ideas and concepts.
We prove this lemma by induction on the structure of $PT(p)$.

1. Assume (y, k, z) is of the form (r_p, n_q, r_q), where p is winning for $n \geq L'(h)$ and q is winning for $n + n_q$. For example, p_a is winning for n, and p_b is winning for $n+3$.

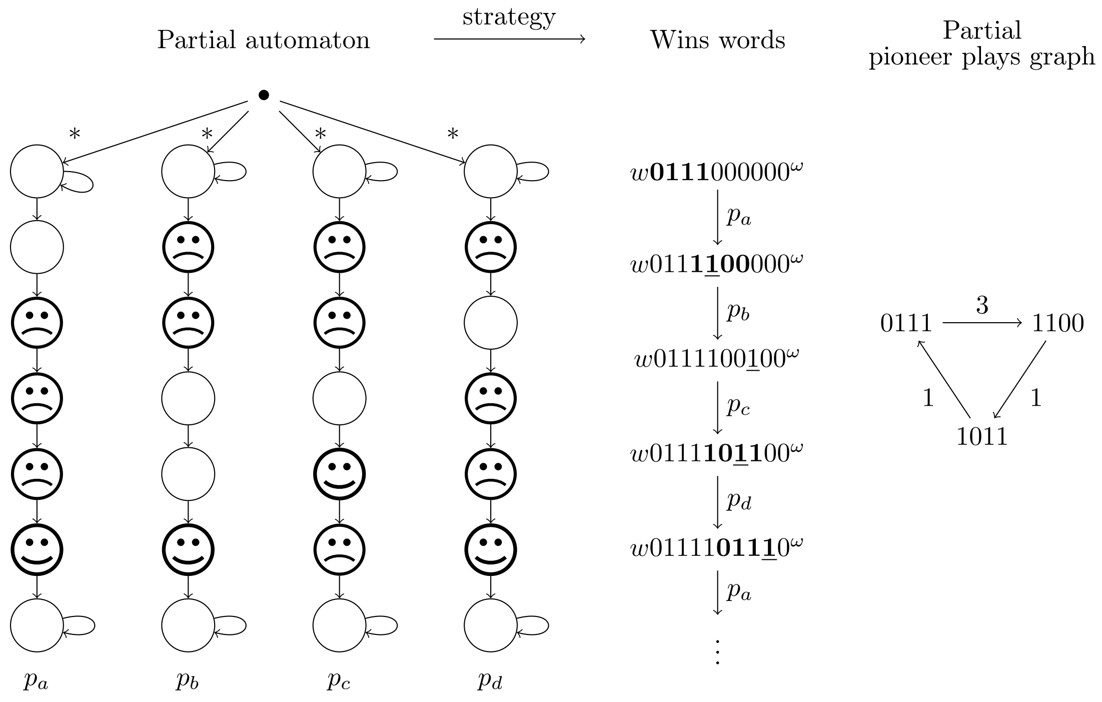

Partial automaton
strategy
Wins words
Partial pioneer plays graph
*
*
*
*
$w\mathbf{0111}000000^\omega$
p_a
$w0111\underline{1}\mathbf{100}000^\omega$
p_b
$w0111100\underline{1}00^\omega$
p_c
$w01111\mathbf{10}\underline{\mathbf{1}}\mathbf{1}00^\omega$
p_d
$w01111\mathbf{0111}\underline{1}0^\omega$
p_a
0111
3
1100
1
1
1011
p_a
p_b
p_c
p_d

Figure 16: How a pioneer plays graph is built and used. Words described in the wins graph are shorter than $L(h)$ for clarity. On the left is part of an automaton. In the middle is a sequence of plays in this automaton, progressively increasing the wins word. On the right is part of the pioneer plays graph one can deduce from this sequence of plays.
Underlined bits are the ones added by previous play. Bold bits are the ones that will be considered for the pioneer plays graph. In the sequence of wins word, they are located in wins word right before a pioneer play. Inside a wins word, they are located right before the next (underlined) bit added by the next pioneer play.

Their induced path is long (greater than $L'(h)$). The main idea is to use the safe self-loop in it given by the fact that σ is non-trivial. Adding more self-loopings to it will increase the number of players those paths are winning for. However, doing so will also shift the number of wins those plays depend on. If p_a is winning for n, it depends on wins for $n-1, n-2$ and $n-3$ players. Adding 3 more self-loopings makes it winning for $n+3$, but then it depends on wins to $n+2, n+1$ and n players.

One can not just simply play q right after p, because there might be some needed non-pioneer plays in between. For p_a and p_b there are none, but between p_b and p_d, there is p_c that must be played in between.

Consider the sequence of plays $p, p_1, \ldots, p_k$ played between p and q and of length at least $L'(h)$, where p_k is the play right before q. If $p = p_b$, then this sequence is p_b, p_c. This sequence of plays, starting with pioneer resources r_p whose last bit was at position $n-1$, resulted in achieving pioneer resources r_q whose last bit was at position $n-1+n_q$.

Among the plays $p, p_1, \ldots, p_k$, first consider p. It is played by a non-trivial strategy, so it admits a safe self-loop before its n^{th} state q_n. Let q_{loop} the last safe self-looping state before its last state in $F_\top$. We want to deduce from p a play winning for $|wy| + 1$. Let $\delta := |wy| + 1 - n$. δ, which can be either positive or negative, is the number of self-looping we will add (or remove) on q_{loop}.

Build a play p' by copying p but it self-loops on q_{loop} δ times more (or $|\delta|$ less, in case δ is negative), so that p' wins for $n+\delta$ players. Due to the last hypothesis in the definition of a non-trivial winning strategy, the last visit of q_{loop} is at most $L(h)$ states before the visit of q_n.

If p' depends on a win to α players, then either

The α^{th} state of p' appears before q_{loop}. Such dependencies are hidden in the $\rightarrow^*$ arrows in figure 12. In this case, this dependency is not shifted by self-looping more on q_{loop}, so this dependency is also a dependency of p. As σ is winning, one of the plays before p is winning to α players. This play is also before p when p' is inserted after p, so it solves the dependency.

The α^{th} state of p' appears after q_{loop}. Those are the visible dependencies in figure 12. In this case, $(n+\delta)-\alpha \leq L(h)$, because of non-triviality of σ.

If p' depends on a win to α players, then p depends on a win to $\alpha-\delta$ players. This is due to the fact that the distance between the win and the dependency is the same in p and in p', because no self-loopings were added in between :

$$(n+\delta)-\alpha = n-(\alpha-\delta)$$

From this, we deduce that the $n-(\alpha-\delta)$ bit in r_p, counting from the right, is a 1, because σ is winning. This bit exists because $(n+\delta)-\alpha \leq L(h)$.

Thus, as $r_p = y$, the $(n+\delta)-\alpha$ bit in y counting from the right is also a 1. This bit happens to be the α^{th} bit of wy, because its index is the index of the winning state, $n+\delta$, minus the distance between the winning state and the dependency, $(n+\delta)-\alpha$. As the α^{th} bit of the word is a 1, the dependency is solved.

This concludes that p' can be inserted after p in σ if at some point the wins words ends with y. But we are not done yet; it remains to prove that shifted versions of $p_1, \ldots, p_k$ can also be inserted.

What we prove is that when wins words is $wy0^\omega$ and $p_0, \ldots, p_k$ have been played (with $p_0 = p$), one can insert $p'_0, \ldots, p'_k$ where the primed versions of played have increased the number of self-loopings on their last self-loop before a state of $q_\top$ by δ. We prove this by induction on i the index of the plays. Case $i = 0$ has been treated above.

Let $i \leq k$. Let p'_i the play p_i with δ more self-loopings on q_{loop} its last safe self-loop before its last winning state.

- Dependencies of p'_i before q_{loop} are solved for the same reasons as explained in the initialization.
- Dependencies of p'_i after q_{loop} might be at positions that are not winning in y. If so, as p_i is not losing, it means that a play between p and p_i has been played to add a 1 bit at the relevant position. This play can be shifted by induction hypothesis, so this case is also solved.

□

Restate of lemma 40 : From periodic wins to consecutive wins

Let σ be a non-trivial winning strategy, verifying the same hypothesis as in lemma 39. Let P be the period of the periodic wins given by this lemma.
One can build a new strategy τ by inserting plays such that τ wins for $W(h)$ consecutive number of players without any such play depending on a win for a number of players larger than

$$3L(h) + 2P^2 + W(h).$$

Proof Let π be the periodic pattern achieving periodic wins of period P. By definition, $|\pi| = P$.
First, if $P \leq L(h)$, consider instead of P its smallest multiple greater than $L(h)$. We assume next that $P \geq L(h)$.
Build σ' by adding to σ through lemma 39 periodic wins by repeating the periodic pattern $2P + \lceil W(h) / P \rceil$ times. Let n_0 be the number of players where the periodic wins start. This construction is illustrated in figure 17.

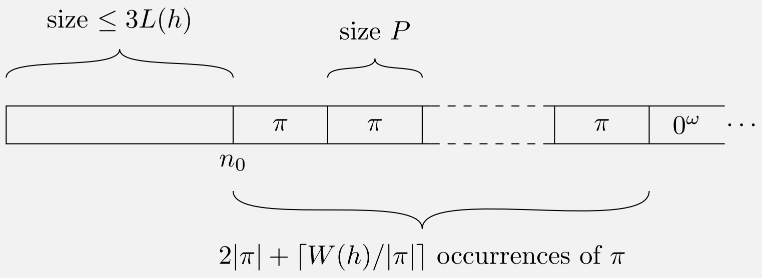


Figure 17: Generate through lemma 39 wins of this form. π is the factor of wins generated. In figure 12, π is 10111.

For $i \in [\![1, P-1]\!]$, consider W_i the sets of plays p in σ' where there exist $z > 0$ such that p is winning for a number of players $n_0 + L(h) + i + zP$. The $L(h)$ addition is here to get rid of limit cases where the transition from short paths to long paths happen.
For each of those sets, pick the first play in this set that σ' plays. Order them by time of play to get a sequence of plays $q_1, \dots, q_K$ (where $K \leq P-1$). In this sequence, denote by n_i the index modulo P for which q_i is winning.
For $i \leq K$, q_i can only have dependencies modulo[a] P that appears either in f, or that are solved by q_j for $j < i$. Thus, next plays added to σ the following way have all of their dependencies solved. This construction is illustrated by figure 18.

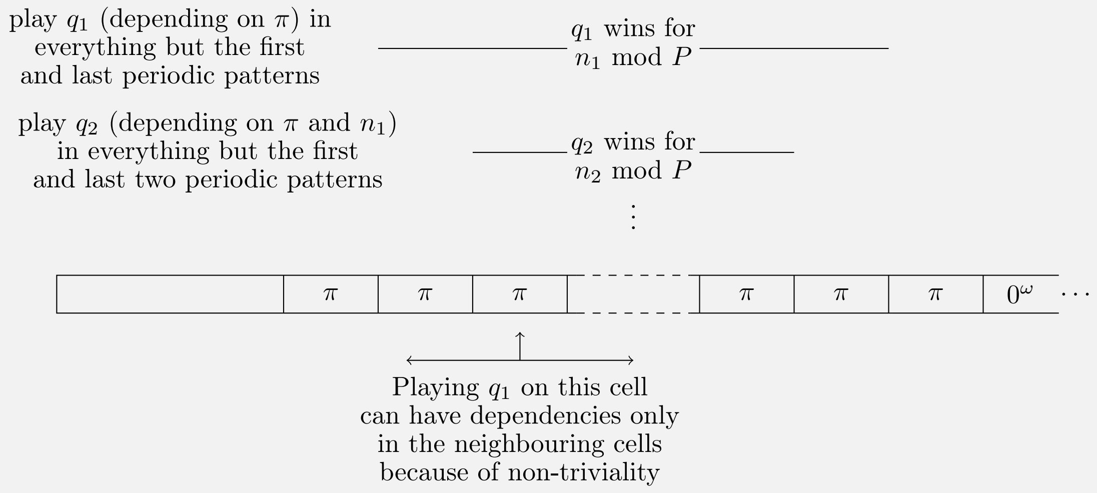

play q_1 (depending on π) in everything but the first and last periodic patterns
q_1 wins for n_1 mod P
play q_2 (depending on π and n_1) in everything but the first and last two periodic patterns
q_2 wins for n_2 mod P
π
0^ω
Playing q_1 on this cell can have dependencies only in the neighbouring cells because of non-triviality

Figure 18: How to fill missing wins in periodic pattern.

This construction propagates the limit case where we are not sure that dependencies are solved. After repeating the first for loop K times, the patterns π that are fulled are the ones that are not the K first from the left nor from the right. As $2P + \lceil W(h)/P \rceil$ consecutive patterns π have been built and $K \leq P$, there are at least $\lceil W(h)/P \rceil$ consecutive full patterns pi when this constructions is over.
After all of these copies are played, there are $W(h)$ consecutive wins in the wins word. □

[a] counting from n_0

Restate of theorem 23

Let $\mathcal{A}$ be a tricolored flat arena, and A be a joint automaton describing the outgoing edges of the ◯ state in $\mathcal{A}$. Assume A has depth h.
There exists a cooperative strategy σ winning whatever the number of players involved if and only if there exists a cooperative strategy σ' such that :

1. σ' wins for any number of player $n \leq f(h)$;
2. no play of σ' winning for n players with $n \leq f(h)$ requires that the game has been won by a previous play for m players with $m > f(h)$.

Proof of reverse implication Let σ' be a convenient strategy. The first pioneer play p winning for $h+1$ players satisfies the following :

- There is a state q_l on the first $h+1$ states of this path admitting a self-loop, because $f(h) \geq h+1$ and pigeonhole principle;
- There is a state $q_\top$ in $F_\top$ on this path after q_l, because it wins when there are $h+1$ players involved;
- There is no state in $F_\perp$ on this path after $q_\top$, because it is a pioneer play.

Thus, as the only dependencies of this play are met before $q_\top$ and there is a loop before $q_\top$, one can build σ the following way :

1. Copy σ' until it has won for any number of player up to $f(h)$;
2. Then forget σ', and instead copy p but progressively increase the number of self-looping on q_l to win against $f(h)+1$ players, then $f(h)+2$, and so on.

$\square$

Proof of first implication Let σ be a winning strategy. The aim is to build from σ a strategy winning for $W(h)$ number of players, such that any of the plays achieving this is losing to a number of players more than $f(h)$ if played first. Lemmata 28 and 29 will conclude from there.

If σ matches the hypothesis of lemma 40, we're done by simply applying it, as long as

$$f(h) \geq 3L(h) + C(h)^2 + W(h).$$

If not, then it depends on what assumption breaks :

1. There exists a play winning for number of players between $L(h)$ and $f(h)$ such that one of these holds:
   (a) It has no self-loop in $F_\circ$ visited before its $L(h)^{th}$ state. Then lemma 31 concludes.
   (b) It ends with a state not in $F_\circ$. Assume the final state is the i^{th} one. If it is in $F_\top$, it wins for all numbers of players larger than i in one single play, so in particular it wins for $W(h)$ consecutive numbers of players. If it is in $F_\perp$, then as σ is winning, one of the previous plays has already won for all players larger than i, so again in particular for $W(h)$ consecutive number of players.
   (c) There are more than $L(h)$ visited states between the last self-loop in $F_\circ$ and the final state. Then applying lemma 31 to the path starting right after the last self-loop in $F_\circ$ concludes.
2. There exist a pioneer play winning for $n < f(h)$ such that the next pioneer play p is winning for m with $m > n + L(h)$. Then as σ is winning, p has no state in $F_\perp$ after its n^{th} state. If 1c holds, we already proved we are done. If not, we know that the last self-loop in $F_\circ$ in p is at most $L(h)$ states before m. Combined, it means that there are no state in $F_\perp$ between the last self-loop in $F_\circ$ of p and its following state in $F_\top$. Thus, one can repeat it $W(h)$ times by each time increasing the number of self-loopings on the safe loop by one to conclude.

$\square$