A Superpolynomial Lower Bound for Strategy Iteration based on Snare Memorization

Oliver Friedmann

February 13, 2013

Abstract

This paper presents a superpolynomial lower bound for the recently proposed snare memorization non-oblivious strategy iteration algorithm due to Fearnley. Snare memorization is a method to train strategy iteration techniques to remember certain profitable substrategies and reapply them again. We show that there is not much hope to find a polynomial-time algorithm for solving parity games by applying such non-oblivious techniques.

1 Introduction

In this paper, we present a superpolynomial lower bound for the strategy improvement algorithm for solving parity games when parameterized with Fearnley's recent non-oblivious rule [6].

Parity games are played on a directed graph that is partitioned into two node sets associated with the two players; the nodes are labeled with natural numbers, called priorities. A play in a parity game is an infinite sequence of nodes whose winner is determined by the parity of the highest priority that occurs infinitely often, giving parity games their name.

Parity games occur in several fields of theoretical computer science, e.g. as solution to the problem of complementation of tree automata [1, 4] or as algorithmic backend to the model checking problem of the modal μ-calculus [5, 18].

Solving parity games is one of the few combinatorial problems that belong to the complexity class $\mathtt{NP} \cap \mathtt{coNP}$ and that are not (yet) known to belong to $\mathtt{P}$ [5, 3]. It has also been shown that solving parity games belongs to $\mathtt{UP} \cap \mathtt{coUP}$ [11].

There are many algorithms that solve parity games, such as the recursive decomposing algorithm due to Zielonka [20] and its recent improvement by Jurdziński, Paterson and Zwick [13], the small progress measures algorithm due to Jurdziński [12] with its recent improvement by Schewe [15], the model-checking algorithm due to Stevens and Stirling [17] and finally the two strategy improvement algorithms by Vöge and Jurdziński [19] and Schewe [16].

The *strategy improvement*, *strategy iteration* or *policy iteration* technique was introduced by Howard [10] for solving problems on Markov decision processes and has been adapted by several other authors for solving nonterminating stochastic games [9], simple stochastic games [3], discounted and mean payoff games [14, 21] as well as parity games [19].

Strategy iteration is an algorithmic scheme that is parameterized by an *improvement rule* that defines how to select a successor strategy in the iteration process. Several rules, including randomized ones, have been proposed. The most popular deterministic improvement rule is the original locally improving rule that selects the best local improvement in every point [19]. However, we have shown that there is an exponential lower bound for this rule [7, 8].

The exponential lower bound is based on the implementation of a binary counter in the context of a parity game. Cyclic structures represent the bits of the counter with certain strategy settings corresponding to set bits and unset bits. In fact, there is only a polynomial number of such settings for each bit occurring in a run of the strategy iteration with some of these settings reoccurring exponentially often.

Fearnley [6] therefore proposed so-called *non-oblivious* strategy iteration that is aware of reoccurring settings by memorization. He introduced the notion of *snares* and showed that the cyclic structures of our exponential lower bound belong to this subgame type. The paper presents an algorithmic scheme that memorizes occurring snares and tries to reapply learned snare-solving strategies whenever it is more profitable than proceeding with the standard improvement rule. As it turns out, this non-oblivious rule is able to solve the original lower bound games in polynomial time.

The main contribution of this paper, however, is the proof that the non-oblivious strategy iteration technique does not solve parity games in general in polynomial time. We present a family of games on which the non-oblivious rule requires superpolynomially many iterations. Essentially, we propose a binary counter again with cycles representing the bits, but here, every cycle looks a bit differently in each iteration, yielding an exponential number of different occurring snares. Non-oblivious strategy iteration is not able to solve the games in polynomially many iterations as no learned snare can be reused in the run.

The rest of the paper is organized as follows. Section 2 defines the basic notions of parity games and notation that is employed throughout the paper. Section 3 recaps the strategy improvement algorithm by Vöge and Jurdziński. In Section 4, we define and motivate snares and their presence as cyclic structures in strategy iteration and show why they are difficult to solve. Section 5 defines the memorization scheme proposed by Fearnley. In Section 6, we present a family of games on which the snare memorization rule requires a superpolynomial number of iterations and prove our claim in Section 7 correct.

2 Parity Games

A *parity game* is a tuple $G = (V, V_0, V_1, E, \Omega)$ where (V, E) forms a directed graph whose node set is partitioned into $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$, and $\Omega : V \to \mathbb{N}$ is the *priority function* that assigns to each node a natural number called the *priority* of the node. We assume the graph to be total, i.e. for every $v \in V$ there is a $w \in V$ s.t. $(v, w) \in E$.

For two priority assignment functions Ω and Ω', we say that they are *equivalent* iff $\Omega(v) \mod 2 = \Omega'(v) \mod 2$ for every $v \in V$ and $\Omega(v) \leq \Omega(w)$ iff $\Omega'(v) \leq \Omega'(w)$ for every $v, w \in V$. We say that two parity games are *equivalent* iff they have the same graph and equivalent priority assignment functions.

W.l.o.g. we assume Ω to be injective, i.e. there are no two different nodes with the same priority. However, we will reuse priorities in our concrete lower bound constructions whenever a distinction between certain priorities does not have any effect on the analysis of strategy iteration. Given the lower bound games, the formal proof then applies to *any* equivalent game with injective priority assignment function.

In the following we will restrict ourselves to finite parity games.

We use infix notation vEw instead of $(v, w) \in E$ and define the set of all *successors* of v as $vE := \{w \mid vEw\}$. The size $|G|$ of a parity game $G = (V,\ V_0,\ V_1,\ E,\ \Omega)$ is defined to be the cardinality of E, i.e. $|G| := |E|$; since we assume parity games to be total w.r.t. E, this is a reasonable way to measure the size.

The game is played between two players called 0 and 1: starting in a node $v_0 \in V$, they construct an infinite path through the graph as follows. If the construction so far has yielded a finite sequence $v_0 \dots v_n$ and $v_n \in V_i$ then player i selects a $w \in v_nE$ and the play continues with $v_0 \dots v_n w$.

Every play has a unique winner given by the *parity* of the greatest priority that occurs infinitely often. The winner of the play $v_0 v_1 v_2 \dots$ is player 0 iff the largest priority occuring infinitely often is even, otherwise player 1 wins the play. That is, player 0 tries to make an even priority occur infinitely often without any greater odd priorities occurring infinitely often, player 1 attempts the converse.

We depict parity games as directed graphs where nodes owned by player 0 are drawn as circles and nodes owned by player 1 are drawn as rectangles; all nodes are labeled with their respective priority, and – if needed – with their name. We also use diamond-shaped nodes whenever we want to emphasize that it does not matter which player owns the node, i.e. if the out-degree is one.

A *strategy* for player i is a – possibly partial – function $\sigma : V^*V_i \to V$ s.t. for all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_jE$ for all $j = 0, \dots, n-1$ we have: $\sigma(v_0 \dots v_n) \in v_nE$. A play $v_0 v_1 \dots$ *conforms* to a strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ then $v_{j+1} = \sigma(v_0 \dots v_j)$. Intuitively, conforming to a strategy means to always make those choices that are prescribed by the strategy. A strategy σ for player i is a *winning strategy* in node v if player i wins every play that begins in v and conforms to σ.

A strategy σ for player i is called *positional* if for all $v_0 \dots v_n \in V^*V_i$ and all $w_0 \dots w_m \in V^*V_i$ we have: if $v_n = w_m$ then $\sigma(v_0 \dots v_n) = \sigma(w_0 \dots w_m)$. That is, the choice of the strategy on a finite path only depends on the last node on that path. The set of positional strategies for player i is denoted by $\mathcal{S}_i(G)$. Let (v, w) be an edge of player i and ϱ be a strategy of player i. By $\varrho[(v,w)]$ and $\varrho[v \mapsto w]$, we denote the *update of* ϱ *to* (v,w), i.e. :

$$\varrho[(v,w)] : u \mapsto \begin{cases} w & \text{if } u = v \\ \varrho(u) & \text{otherwise} \end{cases}$$

With G we associate two sets $W_0, W_1 \subseteq V$; W_i is the set of all nodes v s.t. player i has a winning strategy for the game G starting in v. Here we restrict ourselves to positional strategies because it is well-known that a player has a (general) winning strategy iff she has a positional winning strategy for a given game. In fact, parity games enjoy positional determinacy meaning that for every node v in the game either $v \in W_0$ or $v \in W_1$ [4]. Furthermore, it is not difficult to show that, whenever player i has winning strategies σ_v for all $v \in U$ for some $U \subseteq V$, then there is also a single strategy σ that is winning for player i from every node in U.

The problem of solving a parity game is to compute W_0 and W_1 as well as corresponding winning strategies σ_0 and σ_1 for the players on their respective winning regions.

A strategy σ for player i induces a subgame $G|_\sigma := (V, V_0, V_1, E|_\sigma, \Omega)$ where $E|_\sigma := \{(u,v) \in E \mid u \in dom(\sigma) \Rightarrow \sigma(u) = v\}$, called *strategy subgame*. In other words, all transitions from u but $\sigma(u)$ are no longer accessible in $G|_\sigma$ for all $u \in V_i$. Let $U \subseteq V$ s.t. $E|_U := E \cap (U \times U)$ is total. The set U induces a subgame $G|_U := (V \cap U, V_0 \cap U, V_1 \cap U, E|_U, \Omega|_U)$ as well.

3 Strategy Iteration

We briefly recap the basic definitions of the strategy improvement algorithm. For a given parity game $G = (V,\ V_0,\ V_1,\ E,\ \Omega)$, the *reward* of node v is defined as follows: $\texttt{rew}_G(v) := \Omega(v)$ if $\Omega(v)$ is even and $\texttt{rew}_G(v) := -\Omega(v)$ otherwise. The set of *profitable nodes* for player 0 resp. 1 is defined to be $V_\oplus := \{v \in V \mid \Omega(v) \equiv_2 0\}$ resp. $V_\ominus := \{v \in V \mid \Omega(v) \equiv_2 1\}$.

The *relevance ordering* $<$ on V is induced by Ω: $v < u :\iff \Omega(v) < \Omega(u)$; additionally one defines the *reward ordering* $\prec$ on V by $v \prec u :\iff rew_G(v) < rew_G(u)$. Note that both orderings are total due to injectivity of the priority function.

Let v be a node, σ be a positional player 0 strategy and τ be a positional player 1 strategy. Starting in v, there is exactly one path $\pi_{\sigma,\tau,v}$ that conforms to σ and τ. Since σ and τ are positional strategies, this play can be uniquely written as follows:

$$\pi_{\sigma,\tau,v} = v_1 \dots v_k (w_1 \dots w_l)^\omega$$

with $v_1 = v$, $v_i \neq w_1$ for all $1 \leq i \leq k$ and $w_1 > w_j$ for all $1 < j \leq l$. Note that the uniqueness follows from the fact that all nodes on the cycle have different priorities and we choose w_1 to be the node with highest priority.

Discrete strategy improvement relies on a more abstract description of such a play $\pi_{\sigma,\tau,v}$. In fact, we only consider the *dominating cycle node* w_1, the set of *more relevant nodes* – i.e. all $v_i > w_1$ – on the path to the cycle node, and the *length* k of the path leading to the cycle node. More formally, the *node valuation of* v *w.r.t.* σ *and* τ is defined as follows:

$$\vartheta_{\sigma,\tau,v} := (w_1, \{v_i > w_1 \mid 1 \leq i \leq k\}, k)$$

Given a node valuation ϑ, we refer to w_1 as the *cycle component*, to $\{v_i > w_1 \mid 1 \leq i \leq k\}$ as the *path component*, and to k as the *length component* of ϑ.

In order to compare node valuations with each other, we introduce a total ordering on the set of node valuations. For that reason, we need to define a total ordering $\prec$ on the second component of node valuations – i.e. on subsets of V – first. To compare two different sets M and N of nodes, we order all nodes lexicographically w.r.t. to their relevance and consider the first position in which the two lexicographically ordered sets differ, i.e. there is a node $v \in M$ and a node $w \in N$ with $v \neq w$ s.t. $u \in M$ iff $u \in N$ for all $u > v$ and $u > w$. Now N is better w.r.t. $\prec$ than M iff $v \prec w$, i.e. the set which gives the higher reward in the first differing position is superior to the other set.

In other words, to determine which set of nodes is better w.r.t. $\prec$, one considers the node with the highest priority that occurs in only one of the two sets. The set owning that node is greater than the other if and only if that node has an even priority. More formally:

$$M \prec N \ :\iff\ M \triangle N \neq \emptyset \text{ and } \max_{<}(M \triangle N) \in ((N \cap V_\oplus) \cup (M \cap V_\ominus))$$

where $M \triangle N$ denotes the symmetric difference of both sets.

Now we are able to extend the total ordering on sets of nodes to node valuations. The motivation behind this ordering is a lexicographic measurement of the profitability of a positional play w.r.t. player 0: the most prominent part of a positional play is the cycle in which the play eventually stays, and here it is the reward ordering on the dominating cycle node that defines the profitability for player 0. The second important part is the loopless path that leads to the dominating cycle node. Here, we measure the profitability of a loopless path by a *lexicographic* ordering on the *relevancy* of the nodes on the path, applying the *reward* ordering on each component in the lexicographic ordering. Finally, we consider the length, and the intuition behind the definition is that, assuming we have an even-priority dominating cycle node, it is better to reach the cycle fast whereas it is better to stay as long as possible out of the cycle otherwise. More formally:

$$(u, M, e) \prec (v, N, f) :\iff \begin{cases} (u \prec v) \text{ or } (u = v \text{ and } M \prec N) \text{ or} \\ (u = v \text{ and } M = N \text{ and } e < f \text{ and } u \in V_\ominus) \text{ or} \\ (u = v \text{ and } M = N \text{ and } e > f \text{ and } u \in V_\oplus) \end{cases}.$$

Given a player 0 strategy σ, it is our goal to find a best response counterstrategy τ that minimizes the associated node valuations. A strategy τ is an *optimal counterstrategy* w.r.t. σ iff for every opponent strategy τ' and for every node v we have: $\vartheta_{\sigma,\tau,v} \preceq \vartheta_{\sigma,\tau',v}$ (where $\preceq$ denotes the symmetric closure of $\prec$).

It is well-known that an optimal counterstrategy always exists and that it is efficiently computable.

Lemma 1. *[19] Let G be a parity game and σ be a player 0 strategy. An optimal counterstrategy for player 1 w.r.t. σ exists and can be computed in polynomial time.*

A fixed but arbitrary optimal counterstrategy w.r.t. σ will be denoted by τ_σ from now on. The associated *game valuation* Ξ_σ is a map that assigns to each node the node valuation w.r.t. σ and τ_σ:

$$\Xi_\sigma : v \mapsto \vartheta_{\sigma,\tau_\sigma,v}.$$

We also write $v \prec_\sigma u$ to compare the Ξ_σ-valuations of two nodes, i.e. to abbreviate $\Xi_\sigma(v) \prec \Xi_\sigma(u)$.

A run of the strategy improvement algorithm can be expressed by a sequence of *improving* game valuations; a partial ordering on game valuations is quite naturally defined as follows:

$$\Xi \lhd \Xi' :\iff (\Xi(v) \preceq \Xi'(v) \text{ for all } v \in V) \text{ and } (\Xi \neq \Xi').$$

Let σ be a strategy, $v \in V_0$ and $w \in vE$. We say that (v, w) is a *σ-improving switch* iff $\sigma(v) \prec_\sigma w$. We say that σ is *improvable* iff σ has an improving switch. We write I_σ to denote the set of improving switches for σ and write $I_\sigma(v) = \{w \mid (v, w) \in I_\sigma\}$.

The improvement step from one strategy to the next one is carried out by an *improvement rule*. It is a map $\mathcal{I} : \mathcal{S}_0(G) \to \mathcal{S}_0(G)$ s.t. $\Xi_\sigma \unlhd \Xi_{\mathcal{I}(\sigma)}$ for every σ and additionally $\Xi_\sigma \lhd \Xi_{\mathcal{I}(\sigma)}$ if σ is improvable.

We say that a function $\mathcal{I} : \mathcal{S}_0(G) \to \mathcal{S}_0(G)$ is a *standard improvement rule* iff it only selects improving switches for finding a successor strategy, i.e.

1. For every node $v \in V_0$ it holds that $\sigma(v) \preceq_\sigma \mathcal{I}(\sigma)(v)$.

2. If σ is improvable then there is a node $v \in V_0$ s.t. $\sigma(v) \prec_\sigma \mathcal{I}(\sigma)(v)$.

Vöge and Jurdziński proved in their work that every strategy that is improved by an arbitrary, non-empty selection of improving switches can only result in strategies with valuations strictly better than the valuation of the original strategy.

Theorem 2 ([19])**.** *Let G be a parity game, σ be an improvable strategy and $\mathcal{I}$ be a standard improvement rule. Then $\Xi_\sigma \lhd \Xi_{\mathcal{I}(\sigma)}$.*

If a strategy is not improvable, the winning sets for both players as well as associated winning strategies can be easily derived from the given valuation.

Theorem 3 ([19])**.** *Let G be a parity game and σ be a non-improvable strategy. Then the following holds:*

1. *$W_0 = \{v \mid \Xi_\sigma(v) = (w, \_, \_)$ and $w \in V_\oplus\}$,*

2. *$W_1 = \{v \mid \Xi_\sigma(v) = (w, \_, \_)$ and $w \in V_\ominus\}$,*

3. *σ is a winning strategy for player 0 on W_0,*

4. *τ_σ is a winning strategy for player 1 on W_1, and*

5. *σ is $\trianglelefteq$-optimal.*

The strategy iteration starts with an initial strategy σ and runs for a given improvement rule $\mathcal{I}$ as follows and returns the optimal player 0 strategy as outlined in the pseudo-code of Algorithm 1.

Algorithm 1 Strategy Iteration

```
procedure StandardStratIt($\mathcal{I}$, $G$, $\sigma$)
    while $\sigma$ is improvable do
        $\sigma \leftarrow \mathcal{I}(\sigma)$
    end while
    return $\sigma$.
end procedure
```

Given an initial strategy σ, a game G and a rule $\mathcal{I}$, the unique execution trace, called *run*, of strategy iteration is the sequence of strategies $\sigma_1, \ldots, \sigma_k$ s.t. $\sigma_1 = \sigma$, $\sigma_{i+1} = \mathcal{I}(\sigma_i)$ for all $i < k$, σ_k optimal and σ_i improvable for all $i < k$. Hereby $k - 1$ is the *length* of the run. We say that strategy improvement *requires l iterations to find the optimal strategy* iff the run has length l.

The initial strategy can be selected in several ways, usually by a simple, deterministic heuristic. As the number of iterations should not depend on a particularly clever choice of an initial strategy, we select the initial strategy ourselves for the lower bound constructions.

In Fearnley's paper [6], strategy improvement is introduced a bit differently, following the approach by Björklund and Vorobyov [2]. It becomes equivalent to our standard treatment here if we restrict ourselves to so-called *sink games* as introduced in our paper [8] on strategy iteration.

We call a parity game G (in combination with an initial strategy ι) a *sink game* iff the following two properties hold:

1. *Sink Existence*: there is a node v^* (called the *sink* of G) with $v^* E v^*$ and $\Omega(v^*) = 1$ reachable from all nodes; also, there is no other node w with $\Omega(w) \leq \Omega(v^*)$.

2. *Sink Seeking*: for each player 0 strategy σ with $\Xi_\iota \trianglelefteq \Xi_\sigma$ and each node w it holds that the cycle component of $\Xi_\sigma(w)$ equals v^*.

Obviously, a sink game is won by player 1. Note that comparing node valuations in a sink game can be reduced to comparing the path components of the respective node valuations, for two reasons. First, the cycle component remains constant. Second, the path-length component equals the cardinality of the path component, because all nodes except the sink node are more relevant than the cycle node itself. In the case of a sink game, we will therefore identify node valuations with their path component.

In order to show that a given parity game is a sink game, one simply has to check that the sink existence property holds by looking at the graph, that every node in the game is won by player 1, and that the sink is the cycle component of all nodes of the initial strategy.

Note, however, that sink games are not necessarily easy to solve – you can, in fact, reduce every parity game to a sink game in polynomial time.

Lemma 4 ([8])**.** *Let G be a parity game fulfilling the sink existence property w.r.t. v^*. G is a sink game iff G is completely won by player 1 (i.e. $W_1 = V$) and for each node w it holds that the cycle component of $\Xi_\iota(w)$ equals v^*.*

Let G be a sink game and $v, r \in V_G$. We define $\Xi_\sigma^{>r}(v)$ to denote the path component of $\Xi_\sigma(v)$ by filtering the nodes which are more relevant than r, i.e.

$$\Xi_\sigma^{>r}(v) = \{u \in \Xi_\sigma(v) \mid \Omega(u) > \Omega(r)\}.$$

It is easy to see that for every r, $\Xi_\sigma^{>r}(v) \prec \Xi_\sigma^{>r}(u)$ implies $\Xi_\sigma(v) \prec \Xi_\sigma(u)$.

We assume from now on that every game is a sink game (because our lower bound construction will be a sink game).

A standard deterministic improvement rule is the *locally optimizing rule* $\mathcal{I}_G^{\mathtt{loc}}$ due to Vöge and Jurdziński [19]. It selects a most profitable strategy decision in every node with respect to the current valuation. More formally, it holds for every strategy σ, every player 0 node v and every $w \in vE$ that $w \preceq_\sigma \mathcal{I}_G^{\mathtt{loc}}(\sigma)(v)$.

Lemma 5 ([19])**.** *The locally optimizing rule can be computed in polynomial time.*

Unfortunately, there is an exponential lower bound construction for the locally optimizing rule [7].

Theorem 6 ([7],[8])**.** *There is a family of games of linear size that requires exponentially many iterations to be solved by strategy improvement with the locally optimizing rule.*

4 Cycles and Snares

Remember that we assume every game to be a sink game. Furthermore assume for the sake of this section that any considered strategy σ is at least as good as the initial strategy, i.e. $\iota \trianglelefteq \sigma$. Let x denote the sink. These assumptions particularly imply that the only cycle component throughout the iteration is x and that the path components of every valuation contain the whole path leading to the sink.

Consider a game that contains two nodes $b \in V_0$ and $e \in V_1$ with $\Omega(b) = 3$, $\Omega(e) = 4$, bEe and eEb. In other words, b and e form a player 0 dominated simple cycle (by dominated, we mean that the largest priority on the cycle is even). See Figure 1 for an example of such a situation. Assume further that b is connected to two other nodes s and r, and that e is connected to some node h.

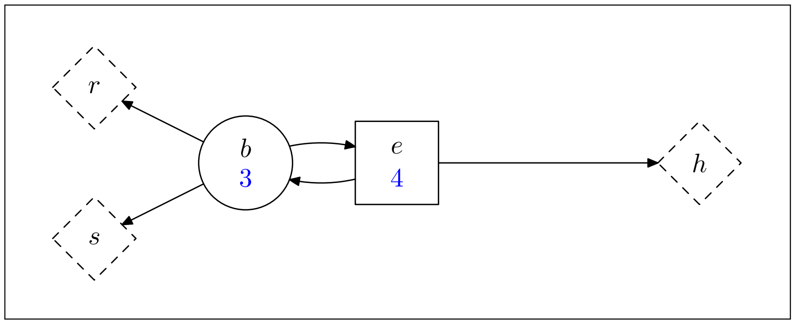

Figure 1: Simple Cycle

Let now σ be a strategy and assume that $\sigma(b) = s$ and $\tau_\sigma(e) = b$. This implies that $b \prec_\sigma h$ (since τ_σ is an optimal counterstrategy), $\Xi_\sigma(e) = \Xi_\sigma(b) \cup \{e\}$ and therefore $\Xi_\sigma(e) = \Xi_\sigma(s) \cup \{b, e\}$. It particularly implies that (b, e) is an improving switch. Assume further that $s \prec_\sigma r$, i.e. (b, r) is an improving switch as well. For choosing the next strategy σ', the locally optimizing rule would compare the valuation of r with the valuation of e and then select the optimal one.

Selecting e would yield a much better valuation than just $\Xi_\sigma(e)$ (i.e. $\Xi_\sigma(e) \prec \Xi_{\sigma[b \mapsto e]}(e)$), since by moving into the simple cycle, player 0 forces player 1 to move out to h (otherwise player 0 would win this cycle which contradicts our assumption that the game is a sink game). In other words, the player 1 nodes of a player 0 dominated cycle can hide (when just considering the current valuation) the effective valuation gain that moving into the cycle by player 0 would give. Schewe's globally optimizing rule [16] is aware of this effect, but a slightly more complicated version of such simple cycles – called *stubborn cycles* – suffices to fool the globally optimizing rule as well [8]. They, in a nutshell, prevent that a cycle can be closed within one iteration – i.e. there are several player 0 edges involved to close the cycle with some of them being no improvement edges.

A variation of *simple cycles* that we use here can be seen in Figure 2. The only addition is that player 0 has an additional node in the cycle which implies that closing a completely opened cycle requires at least two iterations instead of just one.

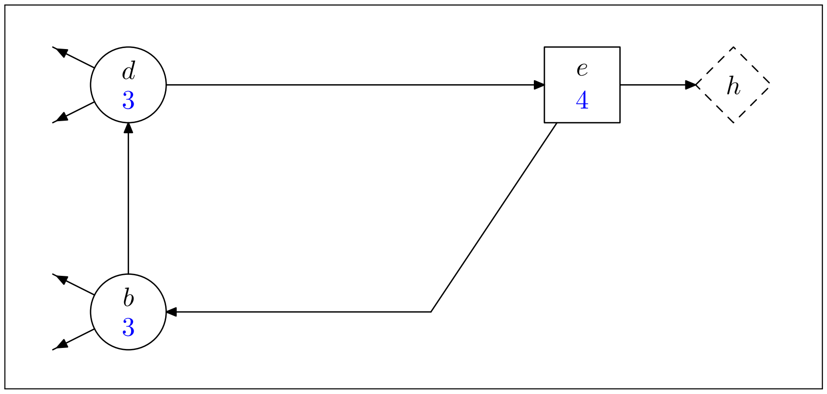

Figure 2: Variation of a simple cycle

Fearnley [6] subsumes all such structures under the more general term *snare*. A *snare* is a tuple (S, ϱ) with $S \subseteq V$ being a set of nodes s.t. $E \cap (S \times S)$ is total, ϱ being a player 0 strategy defined on S s.t. $G|_S$ is completely won by player 0 with the strategy ϱ.

The simple and stubborn cycles, for instance, are snares, and the respective winning strategy is to move all player 0 edges into the cycle.

Since the considered games here are sink games, it must be the case that player 1 escapes from all the snares to other parts of the game. Given a set of nodes S, we define the set of *escapes* as the set of all player 1 controlled edges leaving S, i.e. $\mathrm{Esc}_G(S) = E \cap ((S \cap V_1) \times (V \setminus S))$.

Let (S, ϱ) be a snare. We say that a player 1 strategy τ is (S, ϱ)*-consistent* iff there is an edge $(v, w) \in \mathrm{Esc}_G(S)$ s.t. $\tau(v) = w$.

Lemma 7 ([6])**.** *Let G be a parity game, τ be a player 1 winning strategy for the whole game and (S, ϱ) be a snare. Then τ is (S, ϱ)-consistent.*

Proof. Assume by contradiction that τ is not (S, ϱ)-consistent. Extend ϱ arbitrarily to a total strategy σ on G. Then σ is a player 0 strategy winning at least on S against τ, contradicting that τ is a player 1 winning strategy for the whole game. □

Fearnley's rule depends on a subclass of snares that essentially captures the notion of player 0 dominated cycles with some freedom of choice for player 0 on how to close the cycle. More formally, let σ be a player 0 strategy, $v \in V_0$, $u \in V$ s.t. vEu. We say that (v, u) *is a σ-profitable back edge* iff (v, u) is an improving switch and $v \in \Xi_\sigma(u)$. The notion of a profitable back edge corresponds exactly to closing a player 0 dominated cycle (the reason why this cycle must be player 0 dominated is that we have an improving edge, hence successive valuations can only improve, thus successive cycle components can not degrade).

Given a profitable back edge, we can define the associated snare. The most straight-forward approach would be to select the cycle that is induced by the profitable back edge, σ and τ_σ as a snare, however, it is our goal to end up with snares as large as possible.

Larger snares can be inferred by a profitable back edge by including all σ-consistent paths between the two nodes of the profitable back edge. All these cycles are dominated by player 0 and hence player 1 is forced to move out.

More formally, a profitable back edge (v, u) induces the *critical set* of all nodes on σ-consistent paths from u to v, i.e.

$$\mathrm{Cri}_G(\sigma, (v, u)) := \{w \in V \mid uF^*wF^*v\}$$

where $F = E \cap (\sigma \cup (V_1 \times V))$ and $F^* = \bigcup_{i=0}^{\infty} F^i$. A profitable back edge (v, u) induces the *associated snare* $\mathrm{Sna}_G(\sigma, (v, u))$ by (S, ϱ) where $S = \mathrm{Cri}_G(\sigma, (v, u))$ and $\varrho := \sigma|_S[v \mapsto u]$.

Lemma 8 ([6]). *Let G be a sink game, σ be a player 0 strategy and (v, u) be a σ-profitable back edge. Then $\mathrm{Sna}_G(\sigma, (v, u))$ is a snare.*

Snares can be used to guide strategy iteration in the sense that snare-inconsistent counterstrategies can be forced to be snare-consistent.

Lemma 9 ([6]). *Let G be a sink game, σ be a player 0 strategy and (S, ϱ) be a snare. If τ_σ is not (S, ϱ)-consistent, then there is a σ-improving edge (v, u) with $\varrho(v) = u$.*

Proof. All cycles induced by the snare are dominated by player 0 (i.e. the largest priority is even). As τ_σ is not (S, ϱ)-consistent, no escapes are used by player 1, and hence, place 0 can improve by closing cycles in the snare until player 1 is forced to select one of the escape edges (recall that player 1 will always be able to select escape edges because G is a sink game). □

Given a player 0 strategy σ, we can infer all associated snares by considering all back edges. Define the set of *σ-associated snares* by

$$\mathrm{Snares}_G(\sigma) = \{\mathrm{Sna}_G(\sigma, (v, u)) \mid (v, u) \text{ is a } \sigma\text{-profitable back edge}\}.$$

Given a set of snares $\mathcal{S}$, we can finally define the set of applicable snares as the subset of all snares for which τ_σ is inconsistent, i.e.

$$\mathrm{AppSnares}_G(\mathcal{S}, \sigma) = \{(S, \varrho) \in \mathcal{S} \mid \tau_\sigma \text{ is not } (S, \varrho)\text{-consistent}\}.$$

5 Snare Memorization Rules

The snare memorization or non-oblivious strategy iteration as introduced by Fearnley can be seen either as a rule for the standard strategy iteration or as an iteration scheme on its own. We follow Fearnley's presentation and outline the snare memorization scheme as a complete algorithm.

The essential idea is to perform strategy iteration as usual by some standard improvement rule $\mathcal{I}$ while memorizing all occurring snares along the way. In case that the current counterstrategy is inconsistent with one or more of the learned snares, an additional *snare rule* is asked whether one of the snares should be fixed in the current strategy. If so, the algorithmic scheme performs some strategy iterations that lead to a snare-consistent counterstrategy, otherwise, it invokes the original standard improvement rule again. Formally, a *snare rule* is a function $\mathcal{C}$ that selects, given a game G, a strategy σ, a standard improvement rule $\mathcal{I}$ and a set $\mathcal{T}$ of σ-applicable snares, either the special symbol $\perp$ – meaning do not enforce any snare – or a snare $X \in \mathcal{T}$.

Given a standard improvement rule $\mathcal{I}$, a snare rule $\mathcal{C}$, a sink game G, a strategy σ and a set of snares $\mathcal{S}$, an improvement step of the snare memorizing strategy iteration algorithm can be written as outlined in Algorithm 2.

Algorithm 2 Snare Memorizing Improvement Step

```
 1: procedure MemorizingImprove($\mathcal{I}$, $\mathcal{C}$, $G$, $\sigma$, $\mathcal{S}$)
 2:     $\mathcal{S} \leftarrow \mathcal{S} \cup \mathrm{Snares}_G(\sigma)$
 3:     $\mathcal{T} \leftarrow \mathrm{AppSnares}_G(\mathcal{S}, \sigma)$
 4:     $X \leftarrow \mathcal{C}(\sigma, \mathcal{I}, \mathcal{T})$
 5:     if $X \neq \perp$ then
 6:         while $\tau_\sigma$ is inconsistent with $X = (S, \varrho)$ do
 7:             $e \leftarrow$ some $\sigma$-improving edge $(v, w) \in S$ with $\varrho(v) = w$
 8:             $\sigma \leftarrow \sigma[e]$
 9:         end while
10:     else
11:         $\sigma \leftarrow \mathcal{I}(\sigma)$
12:     end if
13:     return $(\sigma, \mathcal{S})$
14: end procedure
```

Algorithm 3 Snare Memorizing Strategy Iteration

```
1: procedure MemorizingStratIt($\mathcal{I}$, $\mathcal{C}$, $G$, $\sigma$)
2:     $\mathcal{S} \leftarrow \emptyset$
3:     while $\sigma$ is improvable do
4:         $(\sigma, \mathcal{S}) \leftarrow$ MemorizingImprove$(\mathcal{I}, \mathcal{C}, G, \sigma, \mathcal{S})$
5:     end while
6:     return $\sigma$
7: end procedure
```

Given a standard improvement rule $\mathcal{I}$, a snare rule $\mathcal{C}$, a sink game G and an initial strategy σ, the snare memorizing strategy iteration algorithm can be written as outlined in Algorithm 3.

By the lemmas of the previous section, this variation of strategy improvement is clearly correct.

Theorem 10 ([6]). *Let G be a sink game, σ be an initial strategy, $\mathcal{I}$ be an improvement rule and $\mathcal{C}$ be a snare rule.* MemorizingStratIt$(\mathcal{I}, \mathcal{C}, G, \sigma)$ *terminates and returns an optimal player 0 strategy.*

There are several possibilities to define a meaningful snare rule; we describe three rules informally in the following:

1. *Oblivious snare rule*: $\mathcal{C}^{\texttt{Obl}}$ always returns $\bot$ (i.e. has the same effect as running the standard strategy iteration in the first place);
2. *Greedy snare rule*: $\mathcal{C}^{\texttt{Gre}}$ always returns a snare if there is an applicable snare given (if there is more than one applicable snare, use some tie-breaking rule);
3. *Fearnley's snare rule*: $\mathcal{C}^{\texttt{Fea}}$ returns an applicable snare if that would give a higher valuation increase than applying the standard improvement rule (we do not go into detail on how to define the valuation increase).

We leave the definition of Fearnley's specific snare rule implicit here, since its motivation is rather heuristic and of no particular interest in this paper. The reason why we are not interested in the particular snare rule is that our lower bound construction will work indepently of the selected snare rule.

The motivation for non-oblivious strategy iteration, as Fearnley states, is to make strategy iteration aware of reoccurring cyclic structures – as in the original lower bound construction – and to prevent it from iterating exponentially often over a polynomial number of snares. Indeed, Fearnley [6] shows that non-oblivious strategy iteration solves the lower bound games of [7] in polynomial time.

Theorem 11 ([7], [6]). *Let G_n be the family of lower bound games from [7]. The snare memorizing strategy improvement algorithm parameterized with the locally optimizing rule solves the games G_n:*

- *in an exponential number of iterations if parameterized with the oblivious snare rule, and*
- *in a polynomial number of iterations if parameterized with Fearnley's snare rule or with the greedy rule.*

There are two approaches for finding a superpolynomial lower bound for snare memorization. First, we could try to study the specific snare rule and find a binary counter construction in which the snare rule selects the cycles in such a way that the behavior of the binary counter is preserved. Second, we could try to find a binary counter construction that incorporates exponentially many occurring snares s.t. no memorized snare can be reused again. We follow the latter approach here.

6 Lower Bound Games

The lower bound construction is a family of sink parity games that implement binary counters. It is closely related to our original lower bound construction for the locally improving rule [7, 8], and we assume some familiarity with the basic ideas of the construction and how it operates.

We present both the old and the new construction here in one go to show the similarities and differences. In both cases, the implementation of a binary counter is based on *cycles* that allow us to encode a single bit state in a given strategy σ. By having n cycles, we can represent every state of an n-bit binary counter. In order to make them work together as a binary counter, we need to embed the cycles in a more complicated structure called *cycle gate*, connect the cycle gates of the different bits with each other, and with an additional structure, called *deceleration lane*.

This section is organized as follows. First, we briefly remind the reader of the *deceleration lane gadget* without giving all the formal details. Then, we compare the original *simple cycles* with the new *polymorphic cycles* that we use to get a lower bound for Fearnley's rule. We remind the reader of the *cycle gate* structure. Finally, we provide the full constructions.

6.1 Deceleration Lanes

A *deceleration lane* has several, say m, input nodes and two output nodes s and r, called *roots*. More formally, a deceleration lane consists of m internal nodes t_1, ..., t_m, one additional internal node c, m input nodes a_1, ..., a_m and two output nodes s and r, called *roots* of the deceleration lane. All priorities of the deceleration lane are based on some odd priority p. We assume that both root nodes have a priority greater than $p + 2m + 1$.

See Figure 3 for a deceleration lane with $m = 9$ and $p = 15$. (Note that the two roots r and s appear multiple times to simplify the figure - but they all represent the same two roots.) The players, priorities and edges are described in Table 1.

Node	Player	Priority	Successors
t_1	0	p	$\{s,\ r,\ c\}$
$t_{i>1}$	0	$p + 2i - 2$	$\{s,\ r,\ t_{i-1}\}$
c	0	$p + 2m + 1$	$\{s,\ r\}$
a_i	?	$p + 2i - 1$	$\{t_i\}$
s	?	$> p + 2m + 1$	?
r	?	$> p + 2m + 1$	?

Table 1: Description of the Deceleration Lane

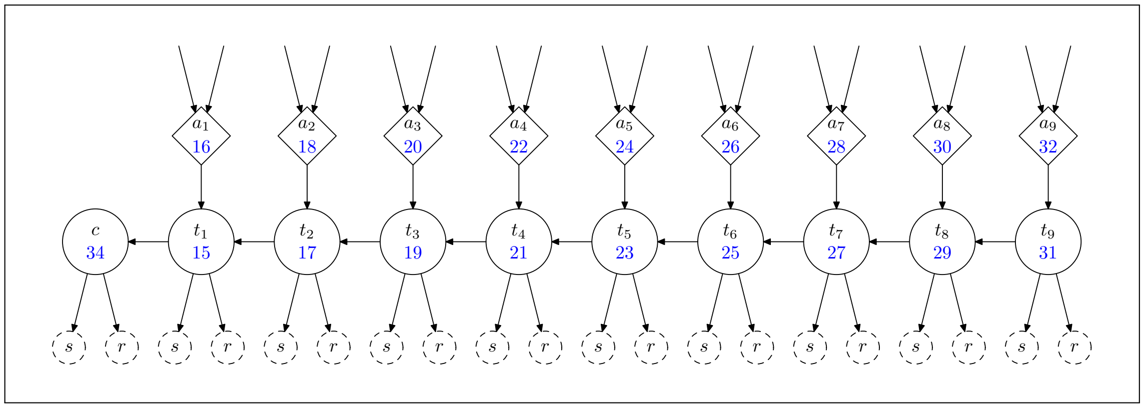

Figure 3: A Deceleration Lane (with $m = 9$ and $p = 15$)

Assume that r has a better valuation than s for a couple of strategy iterations. The input nodes $a_1, \ldots, a_m$ can all reach the best valued root – r – by some internal nodes. Assume that in the beginning, all input nodes a_i reach r via the most direct path (i.e. directly via t_i). By applying the only improving edge (t_1, c), a_1 gets a better valuation than all other input nodes. Then, (t_2, t_1) will be the only improving edge, and applying it will result in a_2 having a better valuation than all other input nodes and so on. This process continues until the other root s has a better valuation than r. Within the next iteration (if we apply all improving switches in the lane), the internal nodes perform a *resetting* step s.t. all input nodes reach the new root node directly via the corresponding internal node. One iteration after that, a_1 has the best valuation compared to all other input nodes again.

In other words, by giving one of the roots, say s, a better valuation than another root, say r, it is possible to reset and therefore reuse the lane again. In fact, the lower bound construction will use a deceleration lane with two roots s and r, and will employ s only for resetting, i.e. after some iterations with $r \succ_\sigma s$, there will be one iteration with $s \succ_\sigma r$ and right after that again $r \succ_\sigma s$.

From an abstract point of view, we describe the state of a deceleration lane by the chosen root and by how many t_i nodes are consecutively reaching c via each other.

We say that σ is *well-behaved* iff there is a node $z \in \{s, r\}$ and there is a $0 \leq k \leq m$ s.t. $\sigma(t_i) = z$ for every $k < i \leq m$ and $\sigma(t_i) \notin \{s, r\}$ for every$1 \leq i \leq k$. We call z the *root of* σ and write $rt(\sigma) = z$. We call k the *index of* σ and write $in(\sigma) = k$. (Well-behaved strategies result in a tree-like configuration of the deceleration lane which is why we call the r and s roots.)

We will use the deceleration lane for the same purpose as in our original construction, namely to postpone closing of player 1 controlled, player 0 dominated cycles representing the bits of the binary counter.

6.2 Simple Cycles

In the original setting of the locally improving rule, we represent the n bits of the binary counter by a gadget called a *simple cycle*. We fix some index i for the simple cycle gadget for the sake of this subsection in order to have consistent node labelings.

Here, a simple cycle consists of two player 0 controlled internal nodes b_i and d_i that are connected to a set of external nodes D_i in the rest of the graph, and one player 1 controlled input node e_i. The node e_i itself is connected to b_i and to one output node $h_i \notin D_i$. All priorities of the simple cycle are based on an odd priority p_i. Intuitively, the p_i is considered to be a very small priority compared to the priorities of the other nodes in the external graph that the simple cycle is connected to.

See Figure 4 for a simple cycle of index 1 with $p_1 = 3$. The players, priorities and edges are described in Table 2.

Node	Player	Priority	Successors
b_i	0	p_i	$\{d_i\} \cup D_i$
d_i	0	p_i	$\{e_i\} \cup D_i$
e_i	1	$p_i + 1$	$\{b_i,\ h_i\}$
h_i	?	$> p_i + 1$	?
$w \in D_i$	?	$> p_i + 1$	?

Table 2: Description of the Simple Cycle

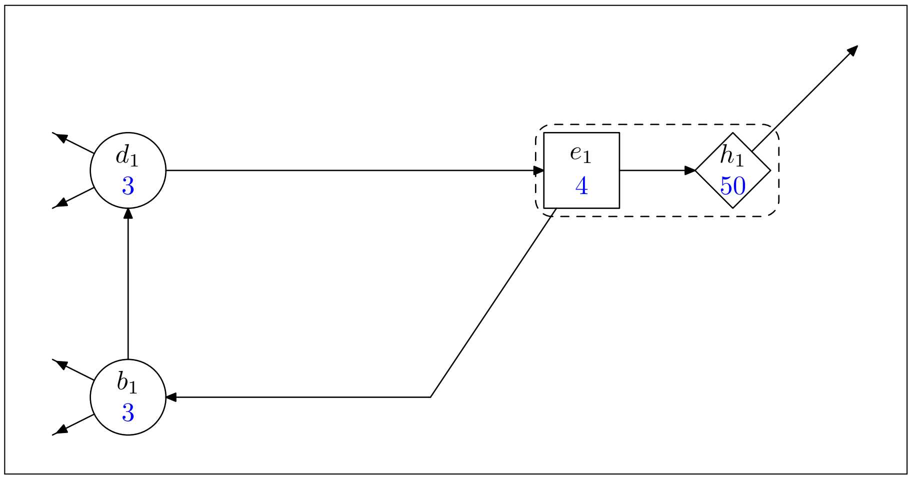

Figure 4: A Simple Cycle (index 1 with $p_1 = 3$)

Given a strategy σ, we say that the cycle is *closed* iff $\sigma(b_i) = d_i$ and $\sigma(d_i) = e_i$, and *open* otherwise. A closed cycle corresponds to a bit which is set while an open cycle corresponds to an unset bit.

Simple cycles are won by player 0, as the most relevant node on the cycle has an even priority. This has important consequences for the behaviour of the player 1 controlled node. We can therefore postpone closing simple cycles by finding nodes $w \in D_i$ in each iteration s.t. $\sigma(b_i) \prec_\sigma w$ and $\sigma(d_i) \prec_\sigma w$. We use this property in the construction of our binary counter: since we do not want to set all bits at the same time, rather one by one, we need to make sure that unset bits which are not supposed to be set remain unset for some time (more precisely, until the respective bit is the least unset bit), and this will be realized by the existence of w.

However, simple cycles of such kind are exactly the structure that Fearnley's improvement rule tries to tackle. Whenever $\sigma(b_i) = d_i$ or $\sigma(d_i) = e_i$, and $\tau_\sigma(e_i) = b_i$, we have that $(\{b_i, d_i, e_i\}, \{b_i \mapsto d_i, d_i \mapsto e_i\}) \in \mathrm{Snares}_G(\sigma)$, i.e. an almost-closed cycle will be identified as being a snare and will therefore be learned. Since we want to reuse cycles that represent bits exponentially often, we have to find a variation of the cycle structure that changes its sequence of edges exponentially often as well s.t. Fearnley's rule is not able to reapply learned snare configurations.

6.3 Polymorphic Cycles

The variation on simple cycles that we propose to circumvent the application of learned cycles is called *polymorphic cycles*. The purpose of this structure is exactly the same as with simple cycles, namely representing the setting of a bit, acting as a pass-through structure. The difference is that the *sequence of edges* of the cyclic structure depends on the bit settings of all higher bits, so that no matter whether a sequence has been learned by Fearnley's rule, every time we set a certain bit, the associated cycle looks different. A polymorphic cycle is built quite similar to the simple cycle, however, the cycle is parameterized by a number of pass-through structures. In the full construction later, there will be one such pass-through structure for every higher bit j relative to bit i. We fix the total number of bits n here which we will call the *top index* of the structure. In other words, bit structure of index i will feature pass-throughs with index $i+1$, $i+2$, …, n.

Formally, a polymorphic cycle again consists of two player 0 controlled internal nodes b_i and d_i that are connected to a set of external nodes D_i in the rest of the graph, and one player 1 controlled input node e_i. The node e_i itself is connected to b_i and to one output node $h_i \notin D_i$. Additionally, for every $i < j \leq n$, there is an internal player 0 node $u_{i,j}$ on the cycle, connected to two player 1 controlled nodes $v_{i,j}$ and $w_{i,j}$ which itself are connected to the next node in the cycle (i.e. $u_{i,j-1}$ or e_i respectively) and to output nodes m_j resp. q_j. All priorities of the polymorphic cycle are based on an odd priority p_i. Intuitively, the p_i is considered to be a very small priority compared to the priorities of the other nodes in the external graph that the polymorphic cycle is connected to.

See Figure 5 for a polymorphic cycle of index 1 with $p_1 = 3$. The players, priorities and edges are described in Table 3.

Node (with $u_{i,i} = e_i$)	Player	Priority	Successors
b_i	0	p_i	$\{d_i\} \cup D_i$
d_i	0	p_i	$\{u_{i,n}\} \cup D_i$
e_i	1	$p_i + 1$	$\{b_i,\ h_i\}$
h_i	?	?	?
$u_{i,j}$	0	2	$\{v_{i,j}, w_{i,j}\}$
$v_{i,j}$	1	3	$\{u_{i,j-1}, m_j\}$
$w_{i,j}$	1	2	$\{u_{i,j-1}, q_j\}$
m_i	?	?	$\{e_i\}$
q_i	?	?	$\{h_i\}$
$m_{j>i}$	?	?	?
$q_{j>i}$	?	?	?

Table 3: Description of a Polymorphic Cycle of index i and top index n

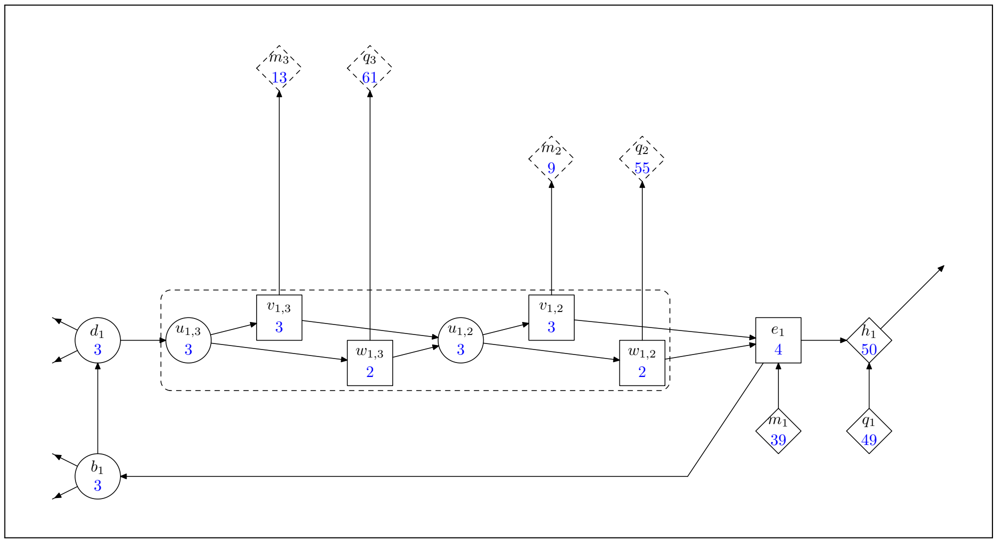

Figure 5: A Polymorphic Cycle (index 1 and top index 3 with $p_1 = 3$)

The essential idea of all pass-through structures of a polymorphic cycle is that at least one of the associated player 1 controlled nodes $v_{i,j}$ and $w_{i,j}$ points into the cyclic structure by the optimal counterstrategy. This implicitly requires that one of the two exits m_j and q_j is always better w.r.t. player 0 – in terms of the valuation – than moving into the cyclic structure.

6.4 Cycle Gates

In the original setting of the locally improving rule, we represent the single bits of the binary counter by simple cycles that appear in the context of a gadget called *cycle gate*.

Formally, a cycle gate consists of two internal nodes e_i and h_i, two input nodes f_i and g_i, and three output nodes b_i, d_i and k_i. The output nodes b_i and d_i will be connected to a set of other nodes D_i in the game graph, and k_i to some other set K_i as well. The three nodes b_i, d_i and e_i form a simple cycle as described earlier. All priorities of the cycle gate are based on two odd priorities p_i and p'_i.

See Figure 6 for a cycle gate of index 1 with $p_1 = 45$ and $p'_1 = 3$. The players, priorities and edges are described in Table 4.

Node	Player	Priority	Successors
b_i	0	p'_i	$\{d_i\} \cup D_i$
d_i	0	p'_i	$\{e_i\} \cup D_i$
e_i	1	$p'_i + 1$	$\{b_i,\ h_i\}$
g_i	0	$p'_i + 3$	$\{f_i,\ k_i\}$
k_i	0	p_i	K_i
f_i	?	$p_i + 2$	$\{e_i\}$
h_i	?	$p_i + 5$	$\{k_i\}$

Table 4: Description of the Cycle Gate

The main idea behind a cycle gate is to have a transit structure (*controlled by the simple cycle*) that is either very profitable or quite unprofitable. The transit structure of the cycle gate has two input nodes, named g_i and f_i, and one output node, named k_i. The input node g_i is controlled by player 0 and connected via two paths with the output node k_i; there is a direct edge and a longer path leading through the interior $(g_i, f_i, e_i, h_i, k_i)$ of the cycle gate. However, the longer path only leads to the output node if the simple cycle, consisting of two

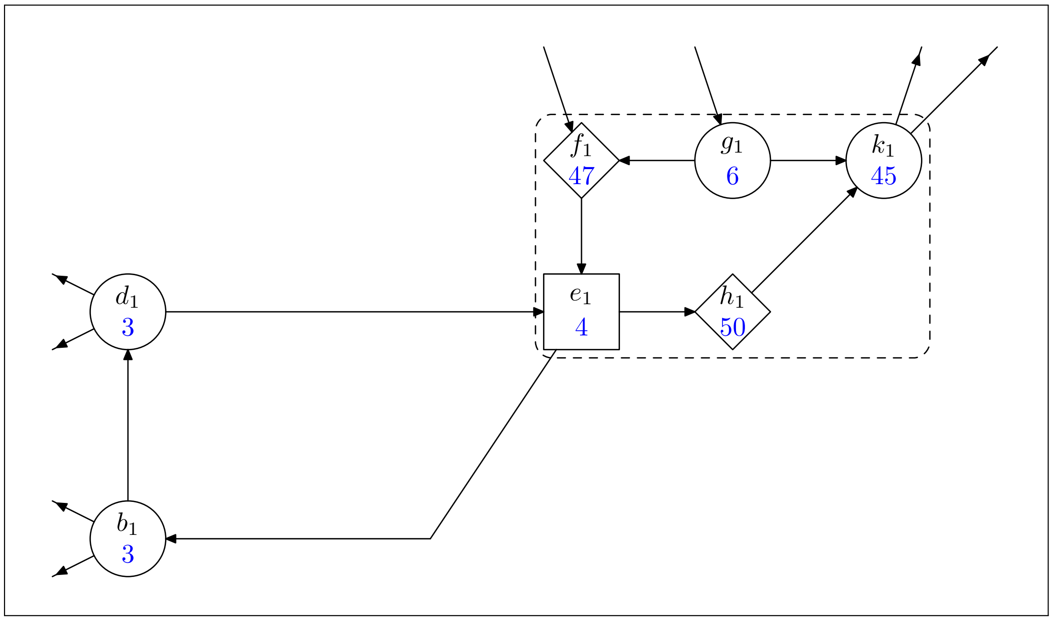

Figure 6: A Cycle Gate (index 1 with $p'_1 = 3$ and $p_1 = 45$, transit structure highlighted)

player 0 nodes b_i and d_i, and one player 1 node e_i, is *closed*. In this case, it is possible and profitable to reach the output node via the internal path; otherwise, this path is not accessible, and hence, the input node has to select the unprofitable direct way to reach the output node. The additional input node f_i can only access the path leading through the interior of the cycle gate, for the following purpose: assume that the simple cycle has just been closed and now the path leading through the interior becomes highly profitable. Hence, the next switching event to happen will be the node g_i switching from the direct path to the path through the interior.

The central idea of cycle gates is that while the gate is open, the player 1 node is able to "hide" the actual improvement that would be realized by moving into the cycle. Schewe's globally optimizing rule is aware of such effects *unless* there are edges involved that are not improving in the current valuation. By constructing larger cycles, for instance cycles with at least three player 0 nodes, it is possible to preserve the invariant that there is one unselected player 0 edge belonging to the cycle that is no improving edge, and hence the globally optimizing rule is equally blind.

Fearnley's snare learning rule tries to circumvent such shortcomings by remembering the exact way a cycle *can* be closed, essentially when the strategy iteration closes the cycle for the first time (more precisely, when the algorithm would be able to close the cycle by making one improvable switch). Hence, the rule checks for all further iterations whether it would be beneficial to apply the learned snare (by beneficial we mean that the current counterstrategy is inconsistent with the learned snare and the snare rule also selects the respective snare), and if so, to enforce the respective strategy update.

Our improved construction will forbid any application of a learned snare by making the sequence of edges of the cyclic structure representing bit number i depending on the state of all higher bits s.t. no learned snare can be reapplied. For every higher bit $j > i$, we include some additional pass-through structures into the cycle of bit number i s.t. the respective sequence of edges of the pass-through structure associated with bit number j directly depends on whether bit j is set. This gadget will be called *polymorphic cycle gate*, see Figure 7 for an example of such a structure.

6.5 Full Construction

In this subsection, we provide the complete construction of both lower bound families. It essentially consists of a sink x, a deceleration lane of length $3n$ that is connected to the two roots s and r, and n cycle gates. The simple cycles (and polymorphic cycles as well) of the cycle gates are connected to the roots and to the deceleration lane with the important detail, that cycle gates for lower bits have less edges to the deceleration lane. This

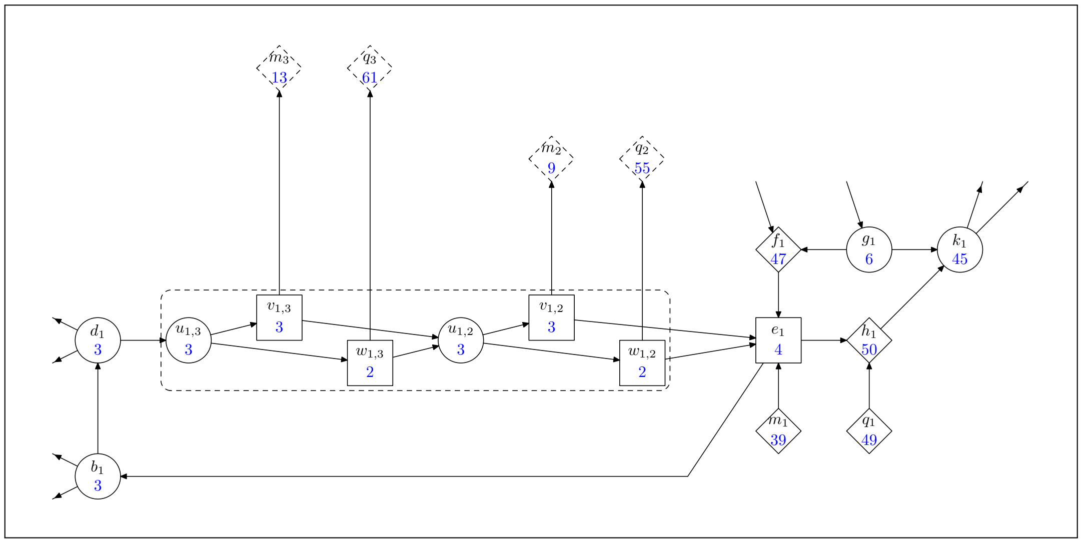

Figure 7: A Polymorphic Cycle Gate (top 3, index 1 with $p'_1 = 3$ and $p_1 = 47$, pass-through highlighted)

construction ensures that open cycle gates for lower bits will close before open cycle gates of higher bits. As for the polymorphic cycles, we introduce additional outgoing edges leading to the cycle gates of higher bits in order to have different looking polymorphic cycle configurations during a run of the strategy iteration.

The output node of a cycle gate is connected to the sink and to the g_*-input nodes of all higher cycle gates. The s root node is connected to all f_*-input nodes, the r root node is connected to all g_*-input nodes.

We now give the formal construction. The original games are denoted by G_n and the modified games are denoted by H_n. The set of nodes of G_n is

$$V_n^G := \{x, s, c, r\} \cup \{t_i, a_i \mid 1 \leq i \leq 3n\} \cup \{b_i, d_i, e_i, g_i, k_i, f_i, h_i \mid 1 \leq i \leq n\}$$

and additionally of H_n is

$$V_n^H := V_n^G \cup \{q_i, m_i \mid 1 \leq i \leq n\} \cup \{u_{i,j}, v_{i,j}, w_{i,j} \mid 1 \leq i < j \leq n\}.$$

The players, priorities and edges of both G_n and H_n are described in Table 5. For convenience of notation, we identify $u_{i,i} = e_i$ in H_n-context and $u_{i,n} = e_i$ in G_n-context. The game G_3 resp. H_3 is depicted in Figure 8 resp. Figure 9 [1].

Lemma 12. *The game G_n has $13 \cdot n + 4$ nodes, $0.5 \cdot (7 \cdot n^2 + 59 \cdot n + 10)$ edges and $18 \cdot n + 8$ as highest priority. In particular, $|G_n| = \mathcal{O}(n^2)$.*

The game H_n has $0.5 \cdot (3 \cdot n^2 + 27 \cdot n + 8)$ nodes, $0.5 \cdot (13 \cdot n^2 + 57 \cdot n + 10)$ edges and $18 \cdot n + 8$ as highest priority. In particular, $|H_n| = \mathcal{O}(n^2)$.

As an initial strategy we select the following ι. It will correspond to the global counter state in which no bit has been set:

$$\iota(t_1) = c, \quad \iota(g_i) = k_i, \quad \iota(* \in \{t_{i>1}, c, b_i\}) = r, \quad \iota(* \in \{k_i, s, r\}) = x.$$

In the context of H_n, we also demand $\iota(u_{i,j}) = v_{i,j}$.

Lemma 13. *Let $n > 0$.*

1. *Both games G_n and H_n are completely won by player 1.*

2. *x is the sink of both G_n and H_n, and the cycle component of $\Xi_\iota(w)$ equals x for all w.*

[1]See http://oliverfriedmann.de/2012/02/11/fearnley-lower-bound for animations of the runs of policy iteration algorithms on these games.

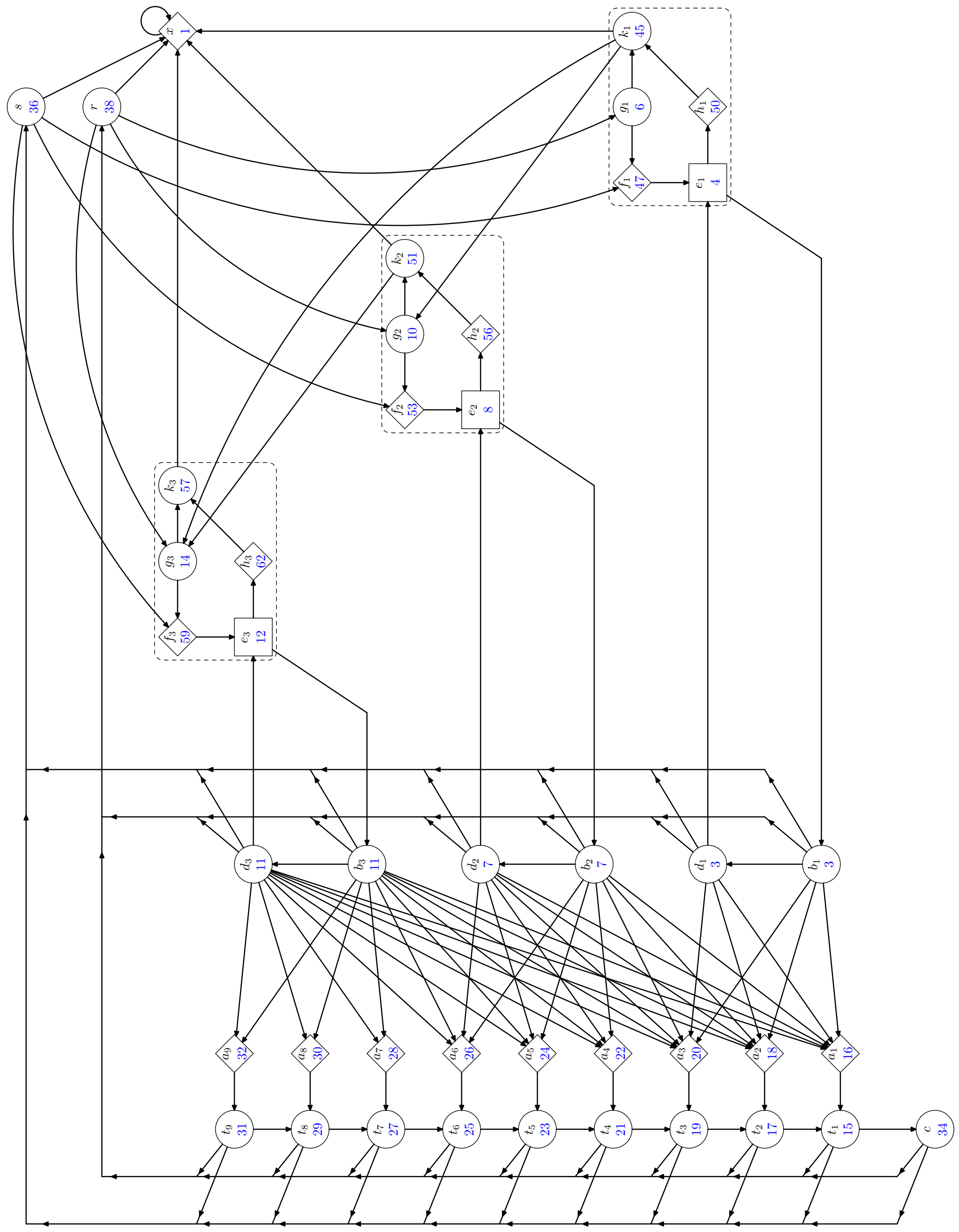

Figure 8: Original Lower Bound Game G_3 (transit structures highlighted)

Proof. Let $n > 0$.

1. Consider the player 1 strategy τ which selects to move to h_i from e_i for all i. Now it is the case that $G_n|_\tau$ contains exactly one cycle that is eventually reached no matter what player 0 does, namely the self-cycle at x which is won by player 1. In the context of H_n, the other choices of player 1 can be arbitrary, the only cycle that is eventually reached in $H_n|_\tau$ is still x.

2. The self-cycle at x obviously is the sink since it can be reached from all other nodes and has the smallest priority 1. Since xEx is the only cycle won by player 1 in $G_n|_\iota$ as well as in $H_n|_\iota$, x must be the cycle component of each node valuation w.r.t. ι.

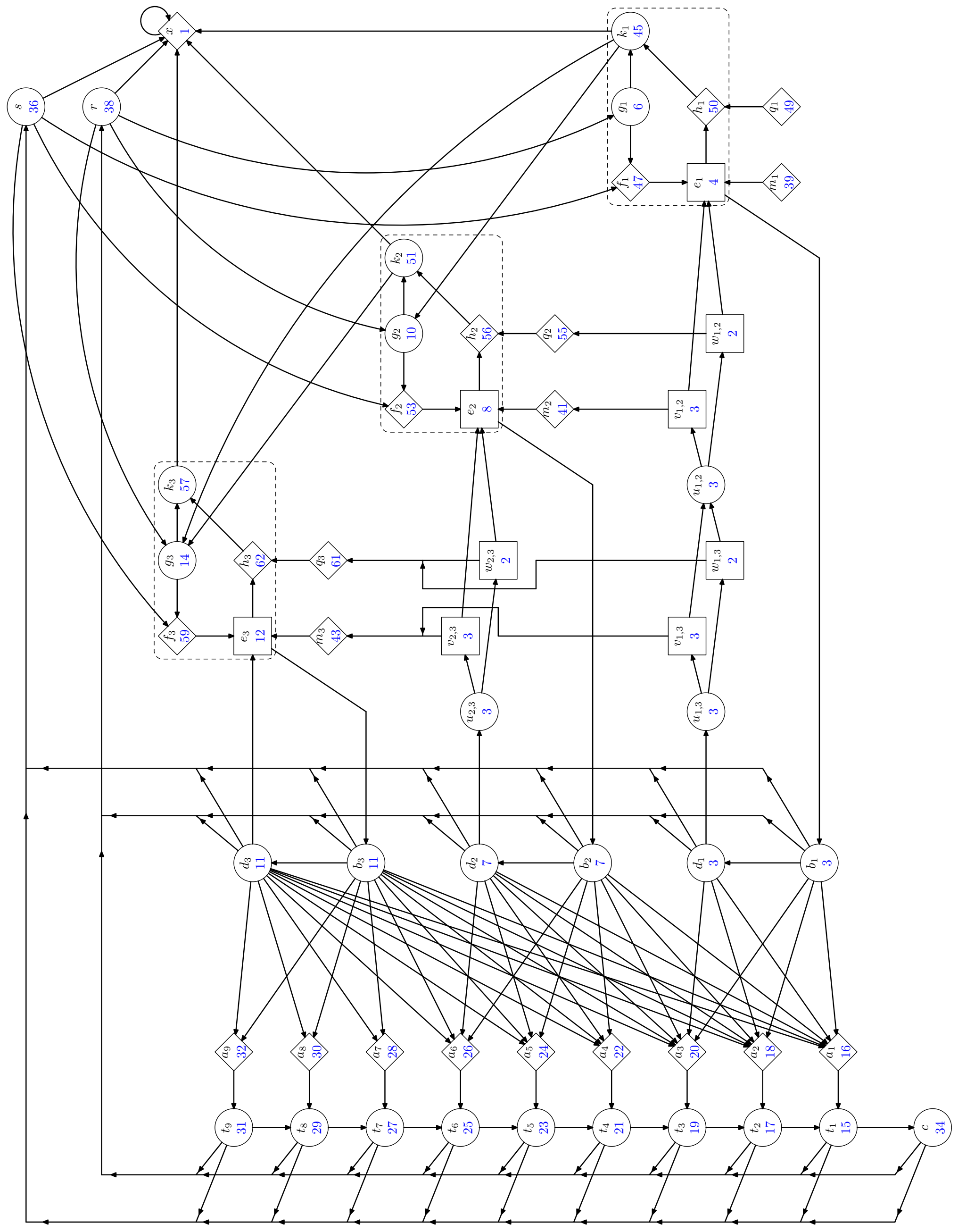

Figure 9: Modified Lower Bound Game H_3 (transit structures highlighted)

□

By Lemma 4 it follows that both G_n and H_n are a sink games, hence it is safe to identify the valuation of a node with its path component from now on. Recall that we can do without the path length component as the cardinality of the path component always equals the path length in sink games.

Note that we reuse some priorities in the constructed games in order to clarify which priority differences matter for the determination of the improving switches. Any *equivalent* parity game (in the sense of Section 2) will have the same sets of improving switches.

	Node	Player	Priority	Successors
	t_1	0	$4n+3$	$\{s,\ r,\ c\}$
	$t_{i>1}$	0	$4n+2i+1$	$\{s,\ r,\ t_{i-1}\}$
	a_i	1	$4n+2i+2$	$\{t_i\}$
	c	0	$10n+4$	$\{s,\ r\}$
	b_i	0	$4i-1$	$\{s,\ d_i,\ r\} \cup \{a_j \mid j \leq 3i\}$
	d_i	0	$4i-1$	$\{s,\ u_{i,n},\ r\} \cup \{a_j \mid j \leq 3i\}$
	e_i	1	$4i$	$\{b_i,\ h_i\}$
	g_i	0	$4i+2$	$\{f_i,\ k_i\}$
	k_i	0	$12n+6i+3$	$\{x\} \cup \{g_j \mid i < j \leq n\}$
	f_i	1	$12n+6i+5$	$\{e_i\}$
	h_i	1	$12n+6i+8$	$\{k_i\}$
	s	0	$10n+6$	$\{f_j \mid j \leq n\} \cup \{x\}$
	r	0	$10n+8$	$\{g_j \mid j \leq n\} \cup \{x\}$
	x	1	1	$\{x\}$
H_n only	$u_{i,j}$	0	3	$\{v_{i,j}, w_{i,j}\}$
	$v_{i,j}$	1	2	$\{u_{i,j-1}, m_j\}$
	$w_{i,j}$	1	2	$\{u_{i,j-1}, q_j\}$
	m_i	1	$10n+2i+7$	$\{e_i\}$
	q_i	1	$12n+6i+7$	$\{h_i\}$

Table 5: Lower Bound Construction

7 Lower Bound Proof

Here, we describe how the binary counter performs the task of counting by strategy improvement. Our games implement binary counters in which every bit is represented by a cycle encapsulated in a cycle gate. An unset bit i corresponds to an open cycle in cycle gate i, a set bit i corresponds to a closed cycle in cycle gate i.

Bit Configurations We introduce notation to succinctly describe binary counters. It will be convenient for us to consider counter configurations with an *infinite* tape, where unused bits are zero. The set of n-bit configurations is formally defined as $\mathcal{B}_n = \{\mathfrak{b} \in \{0,1\}^\infty \mid \forall i > n : \mathfrak{b}_i = 0\}$.

We start with index one, i.e. $\mathfrak{b} \in \mathcal{B}_n$ is essentially a tuple $(\mathfrak{b}_n, \ldots, \mathfrak{b}_1)$, with $\mathfrak{b}_1$ being the least and $\mathfrak{b}_n$ being the most significant bit. By $\mathbf{0}$, we denote the configuration in which all bits are set to zero, and by $\mathbf{1}_n$, we denote the configuration in which the first n bits are one. We write $\mathcal{B} = \bigcup_{n>0} \mathcal{B}_n$ to denote the set of all counter configurations.

The *integer value* of a $\mathfrak{b} \in \mathcal{B}$ is defined as usual, i.e. $|\mathfrak{b}| := \sum_{i>0} \mathfrak{b}_i \cdot 2^{i-1} < \infty$. For two $\mathfrak{b}, \mathfrak{b}' \in \mathcal{B}$, we induce the lexicographic linear ordering $\mathfrak{b} < \mathfrak{b}'$ by $|\mathfrak{b}| < |\mathfrak{b}'|$. It is well-known that $\mathfrak{b} \in \mathcal{B} \mapsto |\mathfrak{b}| \in \mathbb{N}$ is a bijection. For $\mathfrak{b} \in \mathcal{B}$ and $k \in \mathbb{N}$ let $\mathfrak{b} + k$ denote the unique $\mathfrak{b}'$ s.t. $|\mathfrak{b}'| = \mathfrak{b} + k$. If $k \leq |\mathfrak{b}|$, let $\mathfrak{b} - k$ denote the unique $\mathfrak{b}'$ s.t. $|\mathfrak{b}'| + k = |\mathfrak{b}|$.

Given a configuration $\mathfrak{b}$, we access the *i-next set bit* by $\nu_i^n(\mathfrak{b}) = \min(\{n+1\} \cup \{j \geq i \mid \mathfrak{b}_j = 1\})$, and the *i-next unset bit* by $\mu_i(\mathfrak{b}) = \min\{j \geq i \mid \mathfrak{b}_j = 0\}$.

Convenience Notation Let σ be a strategy. We use the following abbrevations for convenience:

- $\bar{\sigma}(s) = i$ if $\sigma(s) = f_i$, and $\bar{\sigma}(s) = n+1$ if $\sigma(s) = x$;
- $\bar{\sigma}(r) = i$ if $\sigma(r) = g_i$, and $\bar{\sigma}(r) = n+1$ if $\sigma(r) = x$;
- $\bar{\sigma}(g_i) = 1$ if $\sigma(g_i) = f_i$, and $\bar{\sigma}(g_i) = 0$ if $\sigma(g_i) = k_i$;

- $\bar{\sigma}(k_i) = j$ if $\sigma(k_i) = g_j$, and $\bar{\sigma}(k_i) = n + 1$ if $\sigma(k_i) = x$;
- $\bar{\sigma}(u_{i,j}) = 1$ if $\sigma(u_{i,j}) = v_{i,j}$, and $\bar{\sigma}(u_{i,j}) = 0$ otherwise;
- $\bar{\sigma}(z) = \sigma(z)$ otherwise.

7.1 Phases

From the most abstract point of view, our lower bound construction performs counting on $\mathcal{B}_n$. However, a single increment of the counter requires the transition through five different phases, named phases 1–5. Phase 1 has variable length depending on the current bit configuration of the counter while phases 2–5 always require one step each phase.

We now define when a well-behaved strategy σ corresponds to one of the phases. Every phase is parameterized by a bit configuration $\mathfrak{b} \in \mathcal{B}_n$, see Table 6.

Phase		1	2	3	4	5
$rt(\sigma)$		r	r	r	r	s
$\bar{\sigma}(s)$		$\nu_1^n(\mathfrak{b})$	$\nu_1^n(\mathfrak{b})$	$\nu_1^n(\mathfrak{b})$	$\mu_1(\mathfrak{b})$	$\mu_1(\mathfrak{b})$
$\bar{\sigma}(r)$		$\nu_1^n(\mathfrak{b})$	$\nu_1^n(\mathfrak{b})$	$\nu_1^n(\mathfrak{b})$	$\nu_1^n(\mathfrak{b})$	$\mu_1(\mathfrak{b})$
$\bar{\sigma}(g_i)$,	$i \neq \mu_1(\mathfrak{b})$	$\mathfrak{b}_i$	$\mathfrak{b}_i$	$\mathfrak{b}_i$	$\mathfrak{b}_i$	$\mathfrak{b}_i$
	$i = \mu_1(\mathfrak{b})$	0	0	0	1	1
$\bar{\sigma}(k_i)$,	$i \geq \mu_1(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$
	$i < \mu_1(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\nu_{i+1}^n(\mathfrak{b})$	$\mu_1(\mathfrak{b})$
$\sigma(b_i)$,	$i > \mu_1(\mathfrak{b})$, $\mathfrak{b}_i = 1$	d_i	d_i	d_i	d_i	d_i
	$i > \mu_1(\mathfrak{b})$, $\mathfrak{b}_i = 0$	$\neq d_i$	$\neq d_i$	$\neq d_i$	$\neq d_i$	s
	$i = \mu_1(\mathfrak{b})$	$\neq d_i$	$\neq d_i$	d_i	d_i	d_i
	$i < \mu_1(\mathfrak{b})$	d_i	d_i	d_i	d_i	s
$\sigma(d_i)$,	$i > \mu_1(\mathfrak{b})$, $\mathfrak{b}_i = 1$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$
	$i > \mu_1(\mathfrak{b})$, $\mathfrak{b}_i = 0$	$\neq u_{i,n}$	$\neq u_{i,n}$	$\neq u_{i,n}$	$\neq u_{i,n}$	s
	$i = \mu_1(\mathfrak{b})$	$\neq u_{i,n}$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$
	$i < \mu_1(\mathfrak{b})$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$	$u_{i,n}$	s
$\bar{\sigma}(u_{i,j})$	H_n only	$\mathfrak{b}_j$	$\mathfrak{b}_j$	$\mathfrak{b}_j$	$\mathfrak{b}_j$	$\mathfrak{b}_j$

Table 6: Strategy Phases (with $u_{n,n} = e_n$)

Intuitively, the phases correspond to the following settings:

1. The first phase, called the *waiting* phase, corresponds to well-behaved strategies σ in which open cycles are busy waiting to be closed while the deceleration lane is assembling.

   The first phase ends, when a cycle corresponding to some unset bit (with index i) has no more improving edges leading to the deceleration lane, and updates the strategy from node d_i to $u_{i,n}$. Due to the fact that the set of edges going to the lane of a lower bit is strictly contained in the set of edges going to the lane of a higher bit, it follows that the least unset bit $i = \mu_1(\mathfrak{b})$ will be the first one to update the strategy from node d_i to $u_{i,n}$.

2. In the second phase, called the *waiting-2* phase, the other open cycles are still busy waiting while the deceleration lane is updating.

   The least unset bit now has a profitable back-edge from b_i to d_i, yielding a potential snare. If the particular snare rule returns $\perp$, the locally improving rule would close the cycle, since the profitable back-edge is the only improving edge for node b_i. If the snare rule on the other hand decides to enforce this snare directly, the cycle would be closed as well.

   The transition from phase 2 to phase 3, therefore, always results in closing cycle i.

3. The third phase, called the *set* phase, corresponds to a strategy σ in which the least unset bit has just been set.

The deceleration lane is still assembling, and the improving switches again include edges of open cycles and edges of the deceleration lane. Additionally, it is improving for the cycle gate of the least recently set bit to update.

4. The forth phase, called the *access* phase, is defined by a renewed correspondence of the cycle gate structure with the bit structure. The s root is connected to the cycle gate of the least recently set bit while r is still connected to the formerly least recently set bit. This implies that s now has a much better valuation than r.

   The cycle gate with the best valuation is now $\mu_1(\mathfrak{b})$, hence, there are many improving switches, that eventually lead to cycle gate $\mu_1(\mathfrak{b})$. First, there are all nodes of the deceleration lane that have improving switches to s. Second, r has an improving switch to $\mu_1(\mathfrak{b})$. Third, lower closed cycles (all lower cycles should be closed!) have an improving switch to $\mu_1(\mathfrak{b})$ (opening them again). Fourth, all lower selector nodes have an improving switch to $\mu_1(\mathfrak{b})$. By performing all these switches, we enter phase five.

5. The fifth and last phase, called the *reset* phase, corresponds to a strategy σ that performed the full increment. However, the cycle gates of the lower bits and the deceleration still have to reset.

   By performing these improving switches, we end up in phase 1 again with incremented binary counter.

7.2 Improving Switches

In this subsection, we determine the sets of improving switches for strategies corresponding to the defined phases. To determine the sets of improving switches, we derive the optimal counterstrategy first, as we can then follow the paths induced by σ and τ_σ to read off the node valuations, allowing us to compare them and select the improving switches.

The first lemma derives the optimal counterstrategy's decisions w.r.t. e_i. The inductive proof shows that the paths originating from e_i only contain nodes with greater indices $j \geq i$, enabling us to prove it by backwards induction indeed. The proof technique therefore is simple as we just follow the paths induced by σ, and the τ_σ given by the induction hypothesis.

Lemma 14. *Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Then the following holds:*

$$\tau_\sigma(e_i) = h_i \iff \begin{cases} \sigma \text{ in P1–2} & \Rightarrow \mathfrak{b}_i = 1, \\ \sigma \text{ in P3–4} & \Rightarrow \mathfrak{b}_i = 1 \text{ or } i = \mu_1(\mathfrak{b}), \text{ and} \\ \sigma \text{ in P5} & \Rightarrow (\mathfrak{b}+1)_i = 1. \end{cases}$$

Proof. Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. We prove an equivalent claim, namely

$$\tau_\sigma(e_i) = h_i \quad \iff \quad \sigma(b_i) = d_i \text{ and } \sigma(d_i) = u_{i,n}$$

for every $1 \leq i \leq n$ by backwards induction on i. Let $B(i) = \{j \geq i \mid \mathfrak{b}_j = 1\}$ and $B'(i) = \{j \geq i \mid (\mathfrak{b}+1)_j = 1\}$.

Let $1 \leq i \leq n$ and let the induction hypothesis hold for $i < j \leq n$. It follows immediately from Table 6 that

$$\Xi_\sigma^{>f_i}(h_i) = \begin{cases} \{h_i\} \cup \{h_j, k_j, f_j \mid j \in B(i+1)\} & \text{if } \sigma \text{ in P1–4} \\ \{h_i\} \cup \{h_j, k_j, f_j \mid j \in B'(i+1)\} & \text{if } \sigma \text{ in P5}\ . \end{cases}$$

If $\sigma(b_i) \neq d_i$ or $\sigma(d_i) \neq u_{i,n}$, it follows from Table 6 that

$$\Xi_\sigma^{>f_i}(b_i) \preceq \begin{cases} \{h_j, k_j, f_j \mid j \in B(i+1)\} & \text{if } \sigma \text{ in P1–4} \\ \{h_j, k_j, f_j \mid j \in B'(i+1)\} & \text{if } \sigma \text{ in P5}\ . \end{cases}$$

It follows that $\Xi_\sigma^{>f_i}(b_i) \prec \Xi_\sigma^{>f_i}(h_i)$, and hence $\tau_\sigma(e_i) = b_i$.

If $\sigma(b_i) = d_i$ and $\sigma(d_i) = u_{i,n}$, note that either σ is in phase 1–4 or $i \geq \mu_1(\sigma)$.

It follows immediately that $\tau_\sigma(e_i) = h_i$ in the context of G_n, as G_n is a sink game, and since $\tau_\sigma(e_i) = b_i$ implies that there is a player 0 dominated cycle b_i-d_i-e_i which is impossible.

In the context of H_n, we need to show that the h_i-exit of the cycle yields the least valuation, i.e. we need to show:

1. For every $j > i$ with $\mathfrak{b}_j = 1$, we have $\Xi_\sigma^{>f_i}(h_i) \prec \Xi_\sigma^{>f_i}(m_j)$.
2. For every $j > i$ with $\mathfrak{b}_j = 0$, we have $\Xi_\sigma^{>f_i}(h_i) \prec \Xi_\sigma^{>f_i}(q_j)$.

Let therefore $j > i$. If $\mathfrak{b}_j = 1$, it follows immediately from Table 6 that

$$\Xi_\sigma^{>f_i}(m_j) = \{h_j, k_j\} \cup \{h_l, k_l, f_l \mid l \in B(j+1)\}.$$

It follows that $\Xi_\sigma^{>f_i}(h_i) \prec \Xi_\sigma^{>f_i}(m_j)$.
If $\mathfrak{b}_j = 0$, it follows from Table 6 that

$$\Xi_\sigma^{>f_i}(q_j) = \{q_j, h_j, k_j\} \cup \{h_l, k_l, f_l \mid l \in B(j+1)\}.$$

It follows again that $\Xi_\sigma^{>f_i}(h_i) \prec \Xi_\sigma^{>f_i}(m_j)$. □

The second lemma is based on the first one and describes the behaviour of the remaining player 1 controlled nodes in the pass-through structures. It shows that the behaviour of every pass-through is as intended: either one of the two player 1 controlled nodes is moving from bit i up to bit j, depending on whether bit j is set. This way, the sequence of edges on the cycle corresponding to bit i is realized by exponentially many different settings, depending on the states of bits $i+1, \ldots, n$.

Lemma 15. *Let σ be a well-behaved strategy in the context of H_n corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Then the following holds:*

$$\tau_\sigma(w_{i,j}) = q_j \iff \begin{cases} \sigma \textit{ in P1–4} : & \mathfrak{b}_j = 1 \\ \sigma \textit{ in P5} : & (\mathfrak{b}+1)_j = 1 \end{cases}$$

$$\tau_\sigma(v_{i,j}) = m_j \iff \begin{cases} \sigma \textit{ in P1–2} : & \mathfrak{b}_j = 0 \\ \sigma \textit{ in P3–5} : & \mathfrak{b}_j = 0 \textit{ and } j > \mu_1(\mathfrak{b}) \end{cases}$$

Proof. Let σ be a well-behaved strategy in the context of H_n corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Let $1 \le i \le n$.

It follows from Table 6 and Lemma 14 that

$$\Xi_\sigma^{>f_i}(h_i) = \begin{cases} \{h_i\} \cup \{h_j, k_j, f_j \mid j \in B(i+1)\} & \text{if } \sigma \text{ in P1–4} \\ \{h_i\} \cup \{h_j, k_j, f_j \mid j \in B'(i+1)\} & \text{if } \sigma \text{ in P5}, \end{cases}$$

and if $\sigma(b_i) \neq d_i$ or $\sigma(d_i) \neq u_{i,n}$, it follows that

$$\Xi_\sigma^{>f_i}(b_i) = \begin{cases} \{h_j, k_j, f_j \mid j \in B(i+1)\} & \text{if } \sigma \text{ in P1–4} \\ \{h_j, k_j, f_j \mid j \in B'(i+1)\} & \text{if } \sigma \text{ in P5}. \end{cases}$$

It is immediate to see by Lemma 14 that

$$\Xi_\sigma^{>f_i}(e_i) = \begin{cases} \Xi_\sigma^{>f_i}(h_i) & \text{if } \tau_\sigma(e_i) = h_i \\ \Xi_\sigma^{>f_i}(b_i) & \text{if } \tau_\sigma(e_i) = b_i. \end{cases}$$

Furthermore it follows that for $i < l$, we have

$$\Xi_\sigma^{>f_i}(q_l) = \begin{cases} \{q_l, h_l, k_l\} \cup \{h_j, k_j, f_j \mid j \in B(l+1)\} & \text{if } \sigma \text{ in P1–4} \\ \{q_l, h_l, k_l\} \cup \{h_j, k_j, f_j \mid j \in B'(l+1)\} & \text{if } \sigma \text{ in P5}, \end{cases}$$

as well as

$$\Xi_\sigma^{>f_i}(m_l) = \begin{cases} \Xi_\sigma^{>f_i}(q_l) \setminus \{q_l\} & \text{if } \tau_\sigma(e_l) = h_l \\ \Xi_\sigma^{>f_i}(b_l) & \text{if } \tau_\sigma(e_l) = b_l. \end{cases}$$

We now show both claims simultaneously by induction on $j = (i+1), \dots, n$. Let $i < j \le n$ and assume that the induction hypothesis holds for all lower j. By induction hypothesis and Table 6, it follows that

$$\Xi_\sigma^{>f_i}(u_{i,j-1}) = \begin{cases} \Xi_\sigma^{>f_i}(h_i) & \text{if } \tau_\sigma(e_i) = h_i \\ \Xi_\sigma^{>f_i}(b_i) & \text{if } \tau_\sigma(e_i) = b_i \text{ and } (i \ge \mu_1(\mathfrak{b}) \text{ or } \sigma \text{ in P1–4}) \\ \Xi_\sigma^{>f_i}(m_{j-1}) & \text{if } \tau_\sigma(e_i) = b_i,\ j \le \mu_1(\mathfrak{b}) \text{ and } \sigma \text{ in P5} \\ \Xi_\sigma^{>f_i}(q_{j-1}) & \text{if } \tau_\sigma(e_i) = b_i,\ i < \mu_1(\mathfrak{b}) < j \text{ and } \sigma \text{ in P5.} \end{cases}$$

In order to show both claims, we need to compare $\Xi_\sigma^{>f_i}(u_{i,j-1})$ with $\Xi_\sigma^{>f_i}(q_j)$ and $\Xi_\sigma^{>f_i}(m_j)$. The claims follow by case distinctions. ☐

Phase		1	2	3	4	5	Note
c		-	-	-	s	r	
s		-	-	$\mu_1^n(\mathfrak{b})$	-	-	
r		-	-	-	$\mu_1^n(\mathfrak{b})$	-	
g_i,	$i > \mu_1(\mathfrak{b})$	-	-	-	-	-	
	$i = \mu_1(\mathfrak{b})$	-	-	1	-	-	
	$i < \mu_1(\mathfrak{b})$	-	-	-	-	0	
k_i,	$i \ge \mu_1(\mathfrak{b})$	-	-	-	-	-	
	$i < \mu_1(\mathfrak{b})$	-	-	-	$\mu_1(\mathfrak{b})$	-	
$u_{i,j}$,	$j > \mu_1(\mathfrak{b})$	-	-	-	-	-	
	$j = \mu_1(\mathfrak{b})$	-	-	-	-	1	H_n only
	$j < \mu_1(\mathfrak{b})$	-	-	-	-	0	

Table 7: Improving Switches Table

We last two lemmata described how the optimal counterstrategy looks like, given a player 0 strategy belonging to one of the phases. This allows us to follow the paths from each node leading to the sink and corresponding to both σ and τ_σ. Hence, we can directly read off the valuations and thus compare them with each other to determine the improving switches.
The first lemma describes the improving switches describes the pass-through structures and the cycle gate transit structures.

Lemma 16. *Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$.*
The improving switches are exactly those specified in Table 7, except for the nodes b_i, d_i and t_i.

Proof. Follows directly from Table 6, Lemma 14 and Lemma 15 by determining the valuations of each node and comparing them with each other. ☐

The second lemma describes the deceleration lane.

Lemma 17. *Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$.*

1. *Let σ be in phase 1–3 and $in(\sigma) = 0$. Then:*

$$c \succ_\sigma a_{3n} \succ_\sigma \ldots \succ_\sigma a_1 \succ_\sigma r \succ_\sigma \{s, t_1, \ldots, t_{3n}\}$$

2. *Let σ be in phase 1–3 and $in(\sigma) > 0$. Then:*

$$a_{in(\sigma)} \succ_\sigma \ldots \succ_\sigma a_1 \succ_\sigma \{c, t_1, \ldots, t_{in(\sigma)}\} \succ_\sigma r \succ_\sigma \{s, t_{in(\sigma)+1}, \ldots, t_{3n}, a_{in(\sigma)+1}, \ldots, a_{3n}\}$$

3. *Let σ be in phase 4. Then:*

$$s \succ_\sigma \{r, c, t_1, \ldots, t_{3n}, a_1, \ldots, a_{3n}\}$$

4. *Let σ be in phase 5. Then:*
$$r \succ_\sigma \{s, c, t_1, \ldots, t_{3n}, a_1, \ldots, a_{3n}\}$$

Proof. See [8] for a full proof describing the improving switches of decelaration lanes. □

The third lemma describes the improving switches of the cycles that control the cycle gates.

Lemma 18. *Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Let $1 \le i \le n$.*

1. *If $u_{i,n}$ is an improving switch in $I_\sigma(d_i)$, it is the worst improving switch in $I_\sigma(d_i)$.*
2. *If d_i is an improving switch in $I_\sigma(b_i)$, it is the worst improving switch in $I_\sigma(b_i)$.*
3. *If $\sigma(d_i) \neq u_{i,n}$ and σ is not in phase 5, then $u_{i,n}$ is an improving switch in $I_\sigma(d_i)$.*
4. *If $\sigma(b_i) \neq d_i$, then d_i is an improving switch in $I_\sigma(b_i)$ iff $\sigma(d_i) = u_{i,n}$.*
5. *If $\sigma(d_i) = u_{i,n}$, then $I_\sigma(d_i) = \emptyset$ iff σ is not in phase 4 or $(\mathfrak{b}+1)_i = 1$.*
6. *If $\sigma(b_i) = d_i$, then $I_\sigma(b_i) = \emptyset$ iff σ is not in phase 4 or $(\mathfrak{b}+1)_i = 1$.*

Proof. See [8] for similar proofs describing the improving switches of cycles. □

7.3 Snares

First, we give a classification of all potential snares that can occur in a run of the strategy iteration algorithm on G_n resp. H_n. In order to describe all snares of H_n concisely, we define the following abbreviation given a bit configuration $\mathfrak{b} \in \mathcal{B}_n$ with $\mathfrak{b} \neq \mathbf{1}_n$ and $i = \mu_1(\mathfrak{b})$:

$$S_\mathfrak{b} := \{e_i, b_i, d_i\} \cup \{u_{i,j}, z_j \mid i < j \le n\}$$

where $z_j = v_{i,j}$ if $\mathfrak{b}_j = 0$ and $z_j = w_{i,j}$ if $\mathfrak{b}_j = 1$.

In the context of G_n, we simply set: $S_\mathfrak{b} := \{e_i, b_i, d_i\}$.

Lemma 19. *Let σ be a player 0 strategy and (y, z) be a σ-profitable back edge. Then $(S, \_) = \mathrm{Sna}(\sigma, (y, z))$ is a snare s.t. there is an $i \le n$ with $S = S_\mathfrak{b}$ for some $\mathfrak{b} \in \mathcal{B}_n$ with $\mathfrak{b} \neq \mathbf{1}_n$ and $i = \mu_1(\mathfrak{b})$.*

Proof. Since every profitable back edge in the context of sink games is based on a player 0 dominated cycle, it is immediate to see that in the context of G_n, the e_i-b_i-d_i-simple cycles are the only ones dominated by player 0.

In the context of H_n, the situation is almost the same. Here, every configuration of the polymorphic cycles gives rise to a potential snare. □

The next lemma shows that the only snares that can be learned throughout a run of the strategy iteration algorithm appear in phase 2 w.r.t. the least unset bit.

Lemma 20. *Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$ with $\mathfrak{b} \neq \mathbf{1}_n$. Then the following holds:*

1. *If σ is not a phase 2 strategy then* $\mathrm{Snares}(\sigma) = \emptyset$.
2. *If σ is a phase 2 strategy then* $\mathrm{Snares}(\sigma) = \{S_\mathfrak{b}\}$.

Proof. Let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Let $i = \mu_1(\mathfrak{b})$.

By Lemma 19, it follows that if $S \in \mathrm{Snares}(\sigma)$, we have $S = S_{\mathfrak{b}'}$ for some $\mathfrak{b}'$ with $i' = \mu_1(\mathfrak{b}')$.

If $S_{\mathfrak{b}'} \in \mathrm{Snares}(\sigma)$, we have that $\tau_\sigma(e_{i'}) = b_{i'}$. By Lemma 14 and Lemma 15, it follows that if $S_{\mathfrak{b}'} \in \mathrm{Snares}(\sigma)$, we need to have:
$$\begin{cases} \sigma \text{ in P1–2} : & \mathfrak{b}_{i'} = 0 \\ \sigma \text{ in P3–4} : & \mathfrak{b}_{i'} = 0 \text{ and } i' \neq i \\ \sigma \text{ in P5} : & (\mathfrak{b}_{i'} = 0 \text{ or } i' < i) \text{ and } i' \neq i \end{cases}$$

In order to have a profitable back edge corresponding to such an S, we necessarily need to have $\sigma(b_{i'}) = d_{i'}$ or $\sigma(d_{i'}) = u_{i',n}$.

It follows by Table 6, that if $S_{\mathfrak{b}'} \in \text{Snares}(\sigma)$, σ cannot be of phases 1, 3, 4 and 5.

If σ is of phase 2, it follows that $i = i'$ on order to have $S_{\mathfrak{b}'} \in \text{Snares}(\sigma)$. By Table 6, Lemma 14 and Lemma 15, it follows that $S_{\mathfrak{b}'} \in \text{Snares}(\sigma)$ iff $\mathfrak{b} = \mathfrak{b}'$. □

The next lemma shows that snares that could have been learned while counting through lower bit settings cannot be reapplied again.

Lemma 21. *Let σ be a well-behaved strategy in H_n corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$. Let $\mathfrak{b}' < \mathfrak{b}$ be another bit configuration. Then σ is $S_{\mathfrak{b}'}$-consistent.*

Proof. Let σ be a well-behaved strategy in H_n corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$ with $i = \mu_1(\mathfrak{b})$. Let $\mathfrak{b}' < \mathfrak{b}$ be another bit configuration with $i' = \mu_1(\mathfrak{b}')$.

Assume by contradiction that σ is $S_{\mathfrak{b}'}$-inconsistent. By definition of $S_{\mathfrak{b}'}$ we have:

1. $\tau_\sigma(e_{i'}) = b_{i'}$,
2. $\mathfrak{b}'_j = 0$ for $j > i'$ implies $\tau_\sigma(w_{i',j}) = u_{i',j-1}$, and
3. $\mathfrak{b}'_j = 1$ for $j > i'$ implies $\tau_\sigma(v_{i',j}) = u_{i',j-1}$.

It follows by Lemma 14 and Lemma 15 that:

$$\begin{cases} \sigma \text{ in P1–2} & : \quad \mathfrak{b}_{i'} = 0 \\ \sigma \text{ in P3–4} & : \quad \mathfrak{b}_{i'} = 0 \text{ and } i' \neq i \\ \sigma \text{ in P5} & : \quad i' < i \text{ or } (\mathfrak{b}_{i'} = 0 \text{ and } i' > i) \end{cases}$$

$$\mathfrak{b}'_{j>i'} = 0 \quad \Rightarrow \quad \begin{cases} \sigma \text{ in P1–4} & : \quad \mathfrak{b}_j = 0 \\ \sigma \text{ in P5} & : \quad j < i \text{ or } (\mathfrak{b}_j = 0 \text{ and } j > i) \end{cases}$$

We now distinguish two cases.

- *Case σ is in phases 1–4 or $i' > i$*: It immediately follows that $\mathfrak{b}'_j \geq \mathfrak{b}_j$ for all $j \geq i'$, which gives the contradiction, since in that case, we have $\mathfrak{b}' \geq \mathfrak{b}$ as $\mathfrak{b}'_j = 1$ for all $j < i'$.
- *Case σ is in phase 5 and $i' < i$*: It is immediate to see that $\mathfrak{b}'_j = 0$ implies $\mathfrak{b}_j = 0$ for all $i < j$. It also follows that $\mathfrak{b}'_i = 1$, since $\mathfrak{b}'_i = 0$ implies $i' < i$ or $i' > i$. We conclude that $\mathfrak{b}'_j \geq \mathfrak{b}_j$ for all $j > i$ and $\mathfrak{b}'_i > \mathfrak{b}_i$, which gives the contradiction.

□

7.4 Transitioning

Here, we describe the transitioning between the phases which eventually results in a full increment of the binary counter. We need to consider the transitioning from phase 1 to phase 1, from phase 1 to phase 2, from phase 2 to phase 3, from phase 3 to phase 4, from phase 4 to phase 5, and from phase 5 to phase 1.

For the sake of this subsection, let σ be a well-behaved strategy corresponding to one of the phases as described in Table 6 w.r.t. a bit configuration $\mathfrak{b} \in \mathcal{B}_n$ with $\mathfrak{b} < \mathbf{1}_n$. Let $\mathcal{S}$ be a set of snares and $\mathcal{C}$ be a snare rule. Let $(\sigma', \mathcal{S}') \leftarrow \text{MemorizingImprove}(\mathcal{I}^{\text{loc}}, \mathcal{C}, F_n, \sigma, \mathcal{S})$, where $F_n = G_n$ or $F_n = H_n$.

Lemma 22. *If σ is in phase 2, then $\mathcal{S}' = \mathcal{S} \cup \{S_{\mathfrak{b}}\}$. Otherwise $\mathcal{S}' = \mathcal{S}$.*

Proof. Follows immediately from Lemma 20. □

Next, we show that in the context of H_n, learned snares can only be applied in phase 2. Define $\mathcal{S}^<_{\mathfrak{b}} = \{S_{\mathfrak{b}'} \mid \mathfrak{b}' < \mathfrak{b}\}$ and $\mathcal{S}^\leq_{\mathfrak{b}} = \{S_{\mathfrak{b}'} \mid \mathfrak{b}' \leq \mathfrak{b}\}$.

Lemma 23. *Assume H_n-context. If σ is in phase 1 and $\mathcal{S} \subseteq \mathcal{S}^<_{\mathfrak{b}}$ or σ is in phases 3–5 and $\mathcal{S} \subseteq \mathcal{S}^\leq_{\mathfrak{b}}$, then $\mathcal{C}(\sigma, \mathcal{I}^{\text{loc}}, \mathcal{S}) = \perp$.*

Proof. Follows immediately from Lemma 21. □

The reason why Fearnley's snare rule leads to a polynomial number of steps in the context of G_n is that Lemma 23 does not hold there.

The next lemma shows how transitioning proceeds when no snares are applied.

Lemma 24. *Let $\mathcal{C}(\sigma, \mathcal{I}^{\mathtt{loc}}, \mathcal{S}) = \perp$. Then the following holds:*

1. *Let σ be in phase 1, $in(\sigma) = 0$ and $\sigma'(b_i) = \sigma'(d_i) = r$ for all i with $\mathfrak{b}_i = 0$. Then σ' is in phase 1 w.r.t. $\mathfrak{b}$, $in(\sigma') = 1$ and $\sigma'(b_i) = \sigma'(d_i) = a_{3i}$ for all i with $\mathfrak{b}_i = 0$.*
2. *Let σ be in phase 1, $1 \leq in(\sigma) \leq 3\mu_1(\mathfrak{b})$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma)}$ for all i. Then σ' is in phase 1 w.r.t. $\mathfrak{b}$, $in(\sigma') = in(\sigma) + 1$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma')}$ for all i.*
3. *Let σ be in phase 1, $3\mu_1(\mathfrak{b}) \leq in(\sigma) \leq 3\mu_1(\mathfrak{b}) + 1$, $\sigma(d_{\mu_1(\mathfrak{b})}) = a_{3\mu_1(\mathfrak{b})}$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma)}$ for all $i > \mu_1(\mathfrak{b})$. Then σ' is in phase 2 w.r.t. $\mathfrak{b}$, $in(\sigma') = in(\sigma) + 1$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma')}$ for all $i > \mu_1(\mathfrak{b})$.*
4. *Let σ be in phase 2, $3\mu_1(\mathfrak{b}) + 1 \leq in(\sigma) \leq 3\mu_1(\mathfrak{b}) + 2$, $\sigma(b_{\mu_1(\mathfrak{b})}) = a_{3\mu_1(\mathfrak{b})}$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma)}$ for all $i > \mu_1(\mathfrak{b})$. Then σ' is in phase 3 w.r.t. $\mathfrak{b}$, $in(\sigma') = in(\sigma) + 1$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma')}$ for all $i > \mu_1(\mathfrak{b})$.*
5. *Let σ be in phase 3, $in(\sigma) \leq 3\mu_1(\mathfrak{b}) + 3$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma)}$ for all $i > \mu_1(\mathfrak{b})$. Then σ' is in phase 4 w.r.t. $\mathfrak{b}$, $in(\sigma') = in(\sigma) + 1$ and $\sigma(b_i), \sigma(d_i) \neq a_{in(\sigma')}$ for all $i > \mu_1(\mathfrak{b})$.*
6. *Let σ be in phase 4. Then σ' is in phase 5 w.r.t. $\mathfrak{b}$.*
7. *Let σ be in phase 5. Then σ' is in phase 1 w.r.t. $\mathfrak{b} + 1$, $in(\sigma') = 0$ and $\sigma'(b_i) = \sigma'(d_i) = r$ for all i with $(\mathfrak{b} + 1)_i = 0$.*

Proof. This follows from Lemma 16, Lemma 17 and Lemma 18. See [8] for similar proofs. □

The last lemma shows how transitioning proceeds when a snare is applied in phase 2.

Lemma 25. *Let σ be in phase 2 and $\mathcal{C}(\sigma, \mathcal{I}^{\mathtt{loc}}, \mathcal{S}) \neq \perp$. Then σ' is in phase 3 with $\sigma' = \sigma[b_{\mu_1(\mathfrak{b})} \mapsto d_{\mu_1(\mathfrak{b})}]$.*

Proof. By Lemma 20 and Lemma 21, it follows that $\mathcal{C}(\sigma, \mathcal{I}^{\mathtt{loc}}, \mathcal{S}) = S_{\mathfrak{b}}$. Hence, the only edge that will be applied to σ is $b_{\mu_1(\mathfrak{b})} \mapsto d_{\mu_1(\mathfrak{b})}$. □

The lemmata show that no matter whether the snare rule selects a snare, we transition from phase 2 to phase 3 while learning the same set of snares. Hence, our lower bound is completely independent from the applied snare rule.

7.5 Lower Bound

Transitioning through the different phases results in a full binary counter.

Theorem 26. *The snare memorizing strategy improvement algorithm parameterized with the locally optimizing rule requires time exponential in n to solve H_n, no matter which particular snare rule is applied.*

Proof. We show by induction on $|\mathfrak{b}|$ that starting with a strategy σ in phase 1 corresponding to $\mathfrak{b}$ with a set of learned snares $\mathcal{S}^{<}_{\mathfrak{b}}$, we end up in phase 1 again after some iterations corresponding to $\mathfrak{b} + 1$ with a set of learned snares $\mathcal{S}^{<}_{\mathfrak{b}+1}$.

This follows directly from Lemma 22, Lemma 23, Lemma 24, and Lemma 25. □

8 Conclusion

We have presented a family of games of polynomial size on which non-oblivious strategy iteration requires exponentially many iterations. It shows that memorization of profitable cyclic structures is not solving the long-standing question whether strategy iteration gives rise to a polynomial time algorithm.

Our construction raises the more general question whether any kind of memorization rule is a viable approach for further research, since the optimal strategy of a game can look very differently from intermediate strategies.

We think that people should try to classify the cyclic structure of parity games more properly. Given the optimal strategy for a game, one could say for each contained snare substrategy, for instance, whether it depends itself on an inner snare substrategy. This approach could yield a snare hiearchy that is bounded by the number of nodes. One could then investigate which rules are able to solve games with a certain maximal snare hierarchy in polynomial time.

References

[1] *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002.

[2] H. Björklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Appl. Math.*, 155(2):210–229, 2007.

[3] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[4] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, 1991. IEEE.

[5] E. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th Conf. on CAV, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[6] J. Fearnley. Non-oblivious strategy improvement. In *Proc. of the 16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'2010, Dakar, Senegal*, Lecture Notes in Computer Science. Springer, 2010.

[7] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS*, pages 145–156, 2009.

[8] O. Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7(3), 2011.

[9] A. J. Hofmann and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[10] R. Howard. *Dynamic Programming and Markov Processes.* The M.I.T. Press, 1960.

[11] M. Jurdziński. Deciding the winner in parity games is in $\texttt{UP} \cap \texttt{coUP}$. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[12] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proc. 17th Ann. Symp. on Theo. Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

[13] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithm, SODA'06*, pages 117–123. ACM, 2006.

[14] A. Puri. *Theory of Hybrid Systems and Discrete Event Systems.* PhD thesis, University of California, Berkeley, 1995.

[15] S. Schewe. Solving parity games in big steps. In *Proc. FST TCS.* Springer-Verlag, 2007.

[16] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *17th Annual Conference on Computer Science Logic (CSL 2008)*, 2008.

[17] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proc. 4th Int. Conf. on Tools and Alg. for the Constr. and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[18] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[19] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proc. 12th Int. Conf. on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.

[20] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1–2):135–183, 1998.

[21] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.