






# A Game of Pawns

Guy Avni   

University of Haifa, Israel

Pranav Ghorpade  

Chennai Mathematical Institute, India

Shibashis Guha   

Tata Institute of Fundamental Research, Mumbai, India

---

## Abstract

We introduce and study *pawn games*, a class of two-player zero-sum turn-based graph games. A turn-based graph game proceeds by placing a token on an initial vertex, and whoever *controls* the vertex on which the token is located, chooses its next location. This leads to a path in the graph, which determines the winner. Traditionally, the control of vertices is predetermined and fixed. The novelty of pawn games is that control of vertices changes dynamically throughout the game as follows. Each vertex of a pawn game is *owned* by a *pawn*. In each turn, the pawns are partitioned between the two players, and the player who *controls* the pawn that owns the vertex on which the token is located, chooses the next location of the token. Control of pawns changes dynamically throughout the game according to a fixed mechanism. Specifically, we define several *grabbing*-based mechanisms in which control of at most one pawn transfers at the end of each turn. We study the complexity of solving pawn games, where we focus on reachability objectives and parameterize the problem by the mechanism that is being used and by restrictions on pawn ownership of vertices. On the positive side, even though pawn games are exponentially-succinct turn-based games, we identify several natural classes that can be solved in PTIME. On the negative side, we identify several EXPTIME-complete classes, where our hardness proofs are based on a new class of games called Lock & Key games, which may be of independent interest.

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

**Keywords and phrases** Graph games, Reachability games, Pawn games, Dynamic vertex control

**Digital Object Identifier** 10.4230/LIPIcs.CONCUR.2023.16

**Related Version** *Full Version*: <https://arxiv.org/pdf/2305.04096.pdf> [7]

**Funding** *Guy Avni*: Partially supported by ISF grant No. 1679/21.

*Shibashis Guha*: Partially supported by the Indian Science and Engineering Research Board (SERB) grant SRG/2021/000466.

## 1 Introduction

Two-player zero-sum *graph games* constitute a fundamental class of games [5] with applications, e.g., in *reactive synthesis* [26], multi-agent systems [4], and more. A graph game is played on a directed graph  $\langle V, E \rangle$ , where  $V = V_1 \cup V_2$  is a fixed partition of the vertices. The game proceeds as follows. A token is initially placed on some vertex. When the token is placed on  $v \in V_i$ , for  $i \in \{1, 2\}$ , Player  $i$  chooses  $u$  with  $\langle v, u \rangle \in E$  to move the token to. The outcome of the game is an infinite path, called a *play*. We focus on *reachability* games: Player 1 wins a play iff it visits a set of target vertices  $T \subseteq V$ .

In this paper, we introduce *pawn games*, which are graph games in which the control of vertices changes dynamically throughout the game as follows. The arena consists of  $d$  *pawns*. For  $1 \leq j \leq d$ , Pawn  $j$  *owns* a set of vertices  $V_j$ . Throughout the game, the pawns are distributed between the two players, and in each turn, the control of pawns determines which player moves the token. Pawn control may be updated after moving the token by running a



© Guy Avni, Pranav Ghorpade, and Shibashis Guha;  
licensed under Creative Commons License CC-BY 4.0

34th International Conference on Concurrency Theory (CONCUR 2023).

Editors: Guillermo A. Pérez and Jean-François Raskin; Article No. 16; pp. 16:1–16:17



Leibniz International Proceedings in Informatics

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



■ **Figure 1** Left: The pawn game  $\mathcal{G}_1$ ; a non-monotonic game under optional-grabbing. Right: The pawn game  $\mathcal{G}_2$  in which Player 1 wins from  $\langle v_0, \{v_0, v_1\} \rangle$ , but must visit  $v_1$  twice.

predetermined *mechanism*. Formally, a *configuration* of a pawn game is a pair  $\langle v, P \rangle$ , where  $v$  denotes the position of the token and  $P$  the set of pawns that Player 1 controls. The player who moves the token is determined according to  $P$ : if Player 1 controls a pawn that owns  $v$ , then Player 1 moves. Specifically, when each vertex is owned by a unique pawn, i.e.,  $V_1, \dots, V_d$  partitions  $V$ , then Player 1 moves iff he controls the pawn that owns  $v$ . We consider the following mechanisms for exchanging control of pawns. For  $i \in \{1, 2\}$ , we denote by  $-i = 3 - i$  the “other player”.

**Optional grabbing.** For  $i \in \{1, 2\}$ , following a Player  $i$  move, Player  $-i$  has the *option* to *grab* one of Player  $i$ ’s pawns; namely, transfer one of the pawns that Player  $-i$  to his control.

**Always grabbing.** For  $i \in \{1, 2\}$ , following *every* Player  $i$  move, Player  $-i$  grabs one of Player  $i$ ’s pawns.

**Always grabbing or giving.** Following a Player  $i$  move, Player  $-i$  either grabs one of Player  $i$ ’s pawns or gives her one of his pawns.

**$k$ -grabbing.** For  $k \in \mathbb{N}$ , Player 1 can grab at most  $k$  pawns from Player 2 throughout the game. In each round, after moving the token, Player 1 has the option of grabbing one of the pawns that is controlled by Player 2. A grabbed pawn stays in the control of Player 1 for the remainder of the game. Note the asymmetry: only Player 1 grabs pawns.

Note that players in pawn games have two types of actions: moving the token and transferring control of pawns. We illustrate the model and some interesting properties of it.

► **Example 1.** Consider the game  $\mathcal{G}_1$  in Fig. 1(left). We consider optional-grabbing and the same reasoning applies for always-grabbing. Each vertex is owned by a unique pawn, and Player 1’s target is  $t$ . Note that Player 2 wins if the game reaches  $s$ . We claim that  $\mathcal{G}_1$  is *non-monotonic*: increasing the set of pawns that Player 1 initially controls is “harmful” for him. Formally, Player 1 wins from configuration  $\langle v_0, \emptyset \rangle$ , i.e., when he initially does not control any pawns, but loses from  $\langle v_0, \{v_0\} \rangle$ , i.e., when controlling  $v_0$ . Indeed, from  $\langle v_0, \emptyset \rangle$ , Player 2 initially moves the token from  $v_0$  to  $v_1$ , Player 1 then uses his option to grab  $v_1$ , and wins by proceeding to  $t$ . Second, from  $\langle v_0, \{v_0\} \rangle$ , Player 1 makes the first move and thus cannot grab  $v_1$ . Since Player 2 controls  $v_1$ , she wins by proceeding to  $s$ . In Thm. 5 and 18, we generalize this observation and show, somewhat surprisingly, that if a player wins from the current vertex  $v$ , then he wins from  $v$  with fewer pawns as long as if he controlled  $v$  previously, then he maintains control of  $v$ .

Consider the game  $\mathcal{G}_2$  in Fig. 1 (right). We consider optional-grabbing, each vertex is owned by a unique pawn, and Player 1’s target is  $t$ . We claim that Player 1 wins from configuration  $\langle v_0, \{v_0, v_2\} \rangle$  and Player 2 can force the game to visit  $v_1$  twice. This differs from turn-based games in which if Player 1 wins, he can force winning while visiting each vertex at most once. To illustrate, consider the following outcome. Player 1 makes the first move, so he cannot grab  $v_1$ . Player 2 avoids losing by moving to  $v_2$ . Player 1 will not grab, move to  $v_3$ , Player 2 moves to  $v_1$ , then Player 1 grabs  $v_1$  and proceeds to  $t$ . We point out that no loop is closed in the explicit *configuration graph* that corresponds to  $\mathcal{G}_2$ .

## Applications

Pawn games model multi-agent settings in which the agent who acts in each turn is not predetermined. We argue that such settings arise naturally.

**Quantitative shield synthesis.** It is common practice to model an *environment* as a Kripke structure (e.g. [27]), which for sake of simplicity, we will think of as a graph in which vertices model environment states and edges model actions. A *policy* chooses an outgoing edge from each vertex. A popular technique to obtain policies is *reinforcement learning* (RL) [29] whose main drawback is lack of worst-case guarantees [13]. In order to regain safety at runtime, a *shield* [19, 6, 13] is placed as a proxy: in each point in time, it can alter the action of a policy. The goal in *shield synthesis* is to synthesize a shield *offline* that ensures safety at runtime while minimizing interventions. We suggest a procedure to synthesize shields based on  $k$ -grabbing pawn games. Player 2 models an unknown policy. We set his goal to reaching an unsafe state. Player 1 (the shield) ensures safety by grabbing at most  $k$  times. Grabbing is associated with a shield intervention. Note that once the shield intervenes in a vertex  $v$ , it will choose the action at  $v$  in subsequent turns. An optimal shield is obtained by finding the minimal  $k$  for which Player 1 has a winning strategy.

We describe other examples that can be captured by a  $k$ -grabbing pawn game in which Player 1 models an “authority” that has the “upper hand”, and aims to maximize freedom of action for Player 2 while using grabs to ensure safety. Consider a concurrent system in which Player 2 models a scheduler and Player 1 can force synchronization, e.g., by means of “locks” or “fences” in order to maintain correctness (see [14]). Synchronization is minimized in order to maximize parallelism and speed. As another example, Player 1 might model an operating system that allows freedom to an application and blocks only unsafe actions. As a final example, in [2], synthesis for a safety specification was enriched with “advice” given by an external policy for optimizing a soft quantitative objective. Again, the challenge is how to maximize accepting advice while maintaining safety.

**Modelling crashes.** A *sabotage game* [31] is a two-player game which is played on a graph. Player 1 (the Runner) moves a token throughout the graph with the goal of reaching a target set. In each round, Player 2 (the Saboteur) crashes an edge from the graph with the goal of preventing Player 1 from reaching his target. Crashes are a simple type of fault that restrict Player 1’s actions. A malicious fault (called *byzantine faults* [21]) actively tries to harm the network, e.g., by moving away from the target. Pawn games can model sabotage games with byzantine faults: each vertex (router) is owned by a unique pawn, all pawns are initially owned by Player 1, and a Player 2 grab corresponds to a byzantine fault. Several grabbing mechanisms are appealing in this context:  $k$ -grabbing restricts the number of faults and optional- and always-grabbing accommodate repairs of routers.

## Our results

We distinguish between three types of ownership of vertices. Let  $V = V_1 \cup \dots \cup V_d$  be a set of vertices, where for  $j \in \{1, \dots, d\}$ , Pawn  $j$  owns the vertices in  $V_j$ . In *one vertex per pawn* (OVPP) games, each pawn owns exactly one vertex, thus  $V_j$  is a singleton, for all  $j \in \{1, \dots, d\}$ . In *multiple vertices per pawn* (MVPP) games,  $V_1, \dots, V_d$  consists of a partition of  $V$ , where the sets might contain more than one vertex. In *overlapping multiple vertices per pawn* (OMVPP) games, the sets might overlap. For example, in the shield synthesis application above, the type of ownership translates to dependencies between interventions:

OVPP models no dependencies, MVPP models cases in which interventions come in “batches”, e.g., grabbing control in all states labeled by some predicate, and OMVPP models the case when the batches overlap. We define that Player 1 moves the token from a vertex  $v$  iff he controls at least one of the pawns that owns  $v$ . Clearly, OMVPP generalizes MVPP, which in turn generalizes OVPP.

We consider the problem of deciding whether Player 1 wins a reachability pawn game from an initial configuration of the game. Our results are summarized below.

Mechanisms	OVPP	MVPP	OMVPP
$k$ -grabbing	PTIME (Thm. 22)	NP-hard (Thm. 23)	PSPACE-C (Thm. 26)
Optional-grabbing	PTIME (Thm. 7)	EXPTIME-C (Thm. 12)	EXPTIME-C (Thm. 12)
Always	PTIME (grab or give; Thm. 21)	PTIME (grab or give; Thm. 21) EXPTIME-C (grab; Thm. 17)	EXPTIME-C (grab; Thm. 17)

Pawn games are succinctly-represented turn-based games. A naive algorithm to solve a pawn game constructs and solves an explicit turn-based game on its configuration graph leading to membership in EXPTIME. We thus find the positive results to be pleasantly surprising; we identify classes of succinctly-represented games that can be solved in PTIME. Each of these algorithms is obtained by a careful and tailored modification to the *attractor-computation* algorithm for turn-based reachability games. For OMVPP  $k$ -grabbing, the PSPACE upper bound is obtained by observing that grabs in a winning strategy must be spaced by at most  $|V|$  turns, implying that a game ends within polynomial-many rounds (Lem. 25).

Our EXPTIME-hardness proofs are based on a new class of games called *Lock & Key* games and may be of independent interest. A Lock & Key game is a turn-based game that is enriched with a set of locks, where each lock is associated with a key. Each edge is labeled by a subset of locks and keys. A lock can either be *closed* or *open*. An edge that is labeled with a closed lock cannot be crossed. A lock changes state once an edge labeled by its key is traversed. We show two reductions. The first shows that deciding the winner in Lock & Key games is EXPTIME-hardness. Second, we reduce Lock & Key games to MVPP optional-grabbing pawn games. The core of the reduction consists of gadgets that simulate the operation of locks and keys using pawns. Then, we carefully analyze the pawn games that result from applying both reductions one after the other, and show that the guarantees are maintained when using always grabbing instead of optional grabbing. The main difficulty in constructing a winning Player  $i$  strategy under always-grabbing from a winning Player  $i$  strategy under optional-grabbing is to ensure that throughout the game, both players have *sufficient* and the *correct* pawns to grab (Lem. 16).

## Related work

The semantics of pawn games is inspired by the seminal paper [4]. There, the goal is, given a game, an objective  $O$ , and a set  $C$  of pawns (called “players” there), to decide whether Player 1 (called a “coalition” there) can ensure  $O$  when he controls the pawns in  $C$ . A key distinction from pawn games is that the set  $C$  that is controlled by Player 1 is fixed. The paper introduced a logic called *alternating time temporal logic*, which was later significantly extended and generalized to *strategy logic* [15, 23, 24]. Multi-player games with rational players have been widely studied; e.g., finding Nash equilibrium [30] or subgame perfect equilibrium [12], and rational synthesis [17, 20, 32, 11]. A key distinction from pawn games is that, in pawn games, as the name suggests, the owners of the resources (pawns) have no individual goals and act as pawns in the control of the players. Changes to multi-player

graph games in order to guarantee existence or improve the quality of an equilibrium have been studied [3, 25, 10]. The key difference from our approach is that there, changes occur *offline*, before the game starts, whereas in pawn games, the transfer of vertex ownership occurs *online*. In *bidding games* [22, 8] (see in particular, *discrete-bidding games* [16, 1, 9]) control of vertices changes online: players have budgets, and in each turn, a *bidding* determines which player moves the token. Bidding games are technically very different from pawn games. While pawn games allow varied and fine-grained mechanisms for transfer of control, bidding games only consider strict auction-based mechanisms, which lead to specialized proof techniques that cannot be applied to pawn games. For example, bidding games are monotonic – more budget cannot harm a player – whereas pawn games are not (see Ex. 1).

## 2 Preliminaries

For  $k \in \mathbb{N}$ , we use  $[k]$  to denote the set  $\{1, \dots, k\}$ . For  $i \in \{1, 2\}$ , we use  $-i = 3 - i$  to refer to the “other player”.

### Turn-based games

Throughout this paper we consider *reachability* objectives. For general graph games, see for example [5]. A *turn-based game* is  $\mathcal{G} = \langle V, E, T \rangle$ , where  $V = V_1 \cup V_2$  is a set of vertices that is partitioned among the players,  $E \subseteq V \times V$  is a set of directed edges, and  $T \subseteq V$  is a set of target vertices for Player 1. Player 1’s goal is to reach  $T$  and Player 2’s goal is to avoid  $T$ . For  $v \in V$ , we denote the *neighbors* of  $v$  by  $N(v) = \{u \in V : E(v, u)\}$ . Intuitively, a *strategy* is a recipe for playing a game: in each vertex it prescribes a neighbor to move the token to. Formally, for  $i \in \{1, 2\}$ , a (memoryless) strategy for Player  $i$  is a function  $f : V_i \rightarrow V$  such that for every  $v \in V_i$ , we have  $f(v) \in N(v)$ .<sup>1</sup> An initial vertex  $v_0 \in V$  together with two strategies  $f_1$  and  $f_2$  for the players, give rise to a unique *play*, denoted  $\pi(v_0, f_1, f_2)$ , which is a finite or infinite path in  $\mathcal{G}$  and is defined inductively as follows. The first vertex is  $v_0$ . For  $j \geq 0$ , assuming  $v_0, \dots, v_j$  has been defined, then  $v_{j+1} = f_i(v_j)$ , where  $v_j \in V_i$ , for  $i \in \{1, 2\}$ . A Player 1 strategy  $f_1$  is *winning* from  $v_0 \in V$  if for every Player 2 strategy  $f_2$ , the play  $\pi(v_0, f_1, f_2)$  ends in  $T$ . Dually, a Player 2 strategy  $f_2$  is winning from  $v_0 \in V$  if for every Player 1 strategy  $f_1$ , the play  $\pi(v_0, f_1, f_2)$  does not visit  $T$ .

► **Theorem 2** ([18]). *Turn based games are determined: from each vertex, one of the players has a (memoryless) winning strategy. Deciding the winner of a game is in PTIME.*

**Proof sketch.** For completeness, we briefly describe the classic *attractor-computation* algorithm. Consider a game  $\langle V, E, T \rangle$ . Let  $W_0 = T$ . For  $i \geq 1$ , let  $W_i = W_{i-1} \cup \{v \in V_1 : N(v) \cap W_i \neq \emptyset\} \cup \{v \in V_2 : N(v) \subseteq W_i\}$ . One can prove by induction that  $W_i$  consists of the vertices from which Player 1 can force reaching  $T$  within  $i$  turns. The sequence necessarily reaches a *fixed point*  $W^1 = \bigcup_{i \geq 1} W_i$ , which can be computed in linear time. Finally, one can show that Player 2 has a winning strategy from each  $v \notin W^1$ . ◀

### Pawn games

A *pawn game* with  $d \in \mathbb{N}$  pawns is  $\mathcal{P} = \langle V, E, T, \mathcal{M} \rangle$ , where  $V = V_1 \cup \dots \cup V_d$  and for  $j \in [d]$ ,  $V_j$  denotes the vertices that Pawn  $j$  *owns*,  $E$  and  $T$  are as in turn-based games, and  $\mathcal{M}$  is a mechanism for exchanging pawns as we elaborate later. Player 1 wins a play if it reaches  $T$ .

<sup>1</sup> We restrict to *memoryless* strategies since these suffice for reachability objectives.

We stress that the set of pawns that he controls when reaching  $T$  is irrelevant. We omit  $\mathcal{M}$  when it is clear from the context. We distinguish between classes of pawn games based on the type of ownership of vertices:

**One Vertex Per Pawn (OVPP).** There is a one-to-one correspondence between pawns and vertices; namely,  $|V| = d$  and each  $V_j$  is singleton, for  $j \in [d]$ . For  $j \in [d]$  and  $\{v_j\} = V_j$ , we sometimes abuse notation by referring to Pawn  $j$  as  $v_j$ .

**Multiple Vertices Per Pawn (MVPP).** Each vertex is owned by a unique pawn but a pawn can own multiple vertices, thus  $V_1, \dots, V_d$  is a partition of  $V$ .

**Overlapping Multiple Vertices Per Pawn (OMVPP).** Each pawn can own multiple vertices and a vertex can be owned by multiple pawns, i.e., we allow  $V_i \cap V_j \neq \emptyset$ , for  $i \neq j$ .

Clearly OMVPP generalizes MVPP, which generalizes OVPP. In MVPP, we sometimes abuse notation and refer to a pawn by a vertex that it owns.

A *configuration* of a pawn game is  $\langle v, P \rangle$ , meaning that the token is placed on a vertex  $v \in V$  and  $P \subseteq [d]$  is the set of pawns that Player 1 *controls*. Implicitly, Player 2 controls the complement set  $\bar{P} = [d] \setminus P$ . Player 1 moves the token from  $\langle v, P \rangle$  iff he controls at least one pawn that owns  $v$ . Note that in OVPP and MVPP, let  $j \in [d]$  with  $v \in V_j$ , then Player 1 moves iff  $i \in P$ . Once the token moves, we update the control of the pawns by applying  $\mathcal{M}$ .

**From pawn games to turn-based games.** We describe the formal semantics of pawn games together with the pawn-exchanging mechanisms by describing the explicit turn-based game that corresponds to a pawn game. For a pawn game  $\mathcal{G} = \langle V, E, T, \mathcal{M} \rangle$ , we construct the turn-based game  $\mathcal{G}' = \langle V', E', T' \rangle$ . For  $i \in \{1, 2\}$ , denote by  $V'_i$  Player  $i$ 's vertices in  $\mathcal{G}'$ . The vertices of  $\mathcal{G}'$  consist of two types of vertices: configuration vertices  $C = V \times 2^{[d]}$ , and *intermediate* vertices  $V \times C$ . When  $\mathcal{M}$  is  $k$ -grabbing, configuration vertices include the remaining number of pawns that Player 1 can grab, as we elaborate below. The target vertices are  $T' = \{\langle v, P \rangle : v \in T\}$ . We describe  $E'$  next. For a configuration vertex  $c = \langle v, P \rangle$ , we define  $c \in V'_1$  iff there exists  $j \in P$  such that  $v \in V_j$ . That is, Player 1 moves from  $c$  in  $\mathcal{G}'$  iff he moves from  $c$  in  $\mathcal{G}$ . We define the neighbors of  $c$  to be the intermediate vertices  $\{\langle v', c \rangle : v' \in N(v)\}$ . That is, moving the token in  $\mathcal{G}'$  from  $c$  to  $\langle v', c \rangle$  corresponds to moving the token from  $v$  to  $v'$  in  $\mathcal{G}$ . Moves from intermediate vertices represent an application of  $\mathcal{M}$ . We consider the following mechanisms.

**Optional grabbing.** For  $i \in \{1, 2\}$ , following a Player  $i$  move, Player  $-i$  has the option to grab one of Player  $i$ 's pawns. Formally, for a configuration vertex  $c = \langle v, P \rangle \in V'_1$ , we have  $N(c) \subseteq V'_2$ . From  $\langle v', c \rangle \in N(c)$ , Player 2 has two options: (1) do not grab and proceed to  $\langle v', P \rangle$ , or (2) grab  $j \in P$ , and proceed to  $\langle v', P \setminus \{j\} \rangle$ . The definition for Player 2 is dual.

**Always grabbing.** For  $i \in \{1, 2\}$ , following a Player  $i$  move, Player  $-i$  always has to grab one of Player  $i$ 's pawns. The formal definition is similar to optional grabbing with the difference that option (1) of not grabbing is not available to the players. We point out that Player  $-i$  grabs only after Player  $i$  has moved, which in particular implies that Player  $i$  controls at least one pawn that Player  $-i$  can grab.

**Always grabbing or giving.** Following a Player  $i$  move, Player  $-i$  must either grab one of Player  $i$ 's pawns or *give* her a pawn. The formal definition is similar to always grabbing with the difference that, for an intermediate vertex  $\langle v', \langle v, P \rangle \rangle$ , there are both neighbors of the form  $\langle v', P \setminus \{j\} \rangle$ , for  $j \in P$ , and neighbors of the form  $\langle v', P \cup \{j\} \rangle$ , for  $j \notin P$ .

**$k$ -grabbing.** After each round, Player 1 has the option of grabbing a pawn from Player 2, and at most  $k$  grabs are allowed in a play. A configuration vertex in  $k$ -grabbing is  $c = \langle v, P, r \rangle$ , where  $r \in [k] \cup \{0\}$  denotes the number of grabs remaining. Intermediate vertices are Player 1 vertices. Let  $\langle v', c \rangle \in V_1'$ . Player 1 has two options: (1) do not grab and proceed to the configuration vertex  $\langle v', P, r \rangle$ , or (2) grab  $j \notin P$ , and proceed to  $\langle v', P \cup \{j\}, r - 1 \rangle$  when  $r > 0$ . Note that grabs are not allowed when  $r = 0$  and that Pawn  $j$  stays at the control of Player 1 for the remainder of the game.

Since pawn games are succinctly-represented turn-based games, Thm. 2 implies determinacy; namely, one of the players wins from each initial configuration. We study the problem of determining the winner of a pawn game, formally defined as follows.

► **Definition 3.** Let  $\alpha \in \{OVPP, MVPP, OMVPP\}$  and  $\beta$  be a pawn-grabbing mechanism. The problem  $\alpha \beta$  PAWN-GAMES takes as input an  $\alpha \beta$  pawn game  $\mathcal{G}$  and an initial configuration  $c$ , and the goal is to decide whether Player 1 wins from  $c$  in  $\mathcal{G}$ .

A naive algorithm to solve a pawn game applies attractor computation on the explicit turn-based game, which implies the following theorem.

► **Theorem 4.**  $\alpha \beta$  PAWN-GAMES is in EXPTIME, for all values of  $\alpha$  and  $\beta$ .

### 3 Optional-Grabbing Pawn Games

Before describing our complexity results, we identify a somewhat unexpected property of MVPP optional-grabbing games. Consider a vertex  $v$  and two sets of pawns  $P$  and  $P'$  having  $P' \subseteq P$ . Intuitively, it is tempting to believe that Player 1 “prefers” configuration  $c = \langle v, P \rangle$  over  $c' = \langle v, P' \rangle$  since he controls more pawns in  $c$ . Somewhat surprisingly, the following theorem shows that the reverse holds (see also Ex. 1). More formally, the theorem states that if Player 1 wins from  $c$ , then he also wins from  $c'$ , under the restriction that if he makes the first move at  $c$  (i.e., he controls  $v$  in  $c$ ), then he also makes the first move in  $c'$  (i.e., he controls  $v$  in  $c'$ ).

► **Theorem 5.** Consider a configuration  $\langle v, P \rangle$  of an MVPP optional-grabbing pawn game  $\mathcal{G}$ . Let  $j \in [d]$  such that  $v \in V_j$  and  $P' \subseteq P$ . Assuming that  $j \in P$  implies  $j \in P'$ , if Player 1 wins from  $\langle v, P \rangle$ , he wins from  $\langle v, P' \rangle$ . Assuming that  $j \in \overline{P'}$  implies  $j \in \overline{P}$ , if Player 2 wins from  $\langle v, P' \rangle$ , she wins from  $\langle v, P \rangle$ .

**Proof.** We prove for Player 1 and the proof for Player 2 follows from determinacy. Let  $\mathcal{G}$ ,  $P$ ,  $P'$ ,  $c = \langle v, P \rangle$ , and  $c' = \langle v, P' \rangle$  be as in the theorem statement. Let  $\mathcal{G}'$  be the turn-based game corresponding to  $\mathcal{G}$ . For  $i \geq 0$ , let  $W_i$  be the set of vertices in  $\mathcal{G}'$  from which Player 1 can win in at most  $i$  rounds (see Thm. 2). The following claim clearly implies the theorem. Its proof, which proceeds by a careful induction, can be found in the full version.

**Claim:** Configuration vertices: for  $i \geq 0$ , if  $\langle v, P \rangle \in W_i$ , then  $\langle v, P' \rangle \in W_i$ . Intermediate vertices: for  $i \geq 1$  and every vertex  $u \in N(v)$ , if  $\langle u, c \rangle \in W_{i-1}$ , then  $\langle u, c' \rangle \in W_{i-1}$ . ◀

Thm. 5 implies that we can restrict attention to strategies that only “grab locally”. The assumption  $j \in P$  implies  $j \in P'$  also implies that if Player 1 wins from  $\langle v, P \rangle$  when Player 2 controls  $v$  then Player 1 also wins from  $\langle v, P' \rangle$  since  $P' \subseteq P$ .

► **Corollary 6.** Consider an MVPP optional-grabbing game. Suppose that Player 1 controls  $P \subseteq [d]$ , and that Player 2 moves the token to a vertex  $v$  owned by Pawn  $j$ , i.e.,  $v \in V_j$ . Player 1 has the option to grab. If Player 1 can win by grabbing a pawn  $j' \neq j$ , i.e., a pawn that does not own the next vertex, he can win by not grabbing at all. Formally, if Player 1 wins from  $\langle v, P \cup \{j'\} \rangle$ , he also wins from  $\langle v, P \rangle$ . And dually for Player 2.

■ **Algorithm 1** Given an OVPP optional-grabbing pawn game  $\mathcal{G} = \langle V, E, T \rangle$  and an initial configuration  $c = \langle v, P_0 \rangle$ , determines which player wins  $\mathcal{G}$  from  $c$ .

---

```

1:  $W_0 = T, i = 0$ 
2: while True do
3:   if  $v_0 \in W_i$  then return Player 1
4:    $W_{i+1} = W_i \cup \{u : N(u) \subseteq W_i\}$ 
5:   if  $W_i \neq W_{i+1}$  then  $i := i + 1$ ; Continue
6:    $B := \{u : N(u) \cap W_i \neq \emptyset\}$ 
7:   if  $B = \emptyset$  then return Player 2
8:   if  $v_0 \in B$  and  $v_0 \in P_0$  then return Player 1
9:    $B' := \{u \in B : N(u) \subseteq B \cup W_i\}$ 
10:  if  $B' \neq \emptyset$  then  $W_{i+1} := W_i \cup B'$ ;  $i := i + 1$ ; Continue
11:   $R = \{u : N(u) \subseteq B\}$ 
12:  if  $R \setminus P_0 \neq \emptyset$  then  $W_{i+1} = W_i \cup (R \setminus P_0)$ ;  $i := i + 1$ 
13:  else return Player 2

```

---

One can show that Thm. 5 and Cor. 6 do not hold for OMVPP optional-grabbing games.

### 3.1 OVPP: A PTIME algorithm

We turn to study complexity results, and start with the following positive result.

► **Theorem 7.** *OVPP optional-grabbing PAWN-GAMES is in PTIME.*

**Proof.** We describe the intuition of the algorithm, the pseudo-code can be found in Alg. 1, and its correctness is proven in the full version. Recall that in turn-based games (see Thm. 2), the attractor computation iteratively “grows” the set of states from which Player 1 wins: initially  $W_0 = T$ , and in each iteration, a vertex  $u$  is added to  $W_i$  if (1)  $u$  belongs to Player 2 and all its neighbors belong to  $W_i$  or (2)  $u$  belongs to Player 1 and it has a neighbor in  $W_i$ . In optional-grabbing games, applying attractor computation is intricate since vertex ownership is dynamic. Note that the reasoning behind (1) above holds; namely, if  $N(u) \subseteq W_i$ , no matter who controls  $u$ , necessarily  $W_i$  is reached in the next turn. However, the reasoning behind (2) fails. Consider a Player 1 vertex  $u$  that has two neighbors  $v_1 \in W_i$  and  $v_2 \notin W_i$ . While  $u$  would be in  $W_{i+1}$  according to (2), under optional-grabbing, when Player 1 makes the move into  $u$ , Player 2 can avoid  $W_i$  by grabbing  $u$  and proceeding to  $v_2$ .

In order to overcome this, our algorithm operates as follows. Vertices that satisfy (1) are added independent of their owner (Line 4). The counterpart of (2) can be seen as two careful steps of attractor computation. First, let  $B$  denote the *border* of  $W_i$ , namely the vertices who have a neighbor in  $W_i$  (Line 6). Second, a vertex  $u$  is in  $W_{i+1}$  in one of two cases. (i)  $u \in B$  and all of its neighbors are in  $B \cup W_i$  (Line 10). Indeed, if Player 1 controls  $u$  he wins by proceeding to  $W_i$  and if Player 2 owns  $u$ , she can avoid  $W_i$  by moving to  $B$ , then Player 1 grabs and proceeds to  $W_i$ . (ii) Player 2 controls  $u$  in the initial configuration and all of its neighbors are in  $B$  (Line 12). Indeed, Player 2 cannot avoid proceeding into  $B$ , and following Player 2’s move, Player 1 grabs and proceeds to  $W_i$ . Finally, note that the algorithm terminates once a fixed point is reached, thus it runs for at most  $|V|$  iterations. ◀



### 3.2 MVPP: EXPTIME-hardness via Lock & Key games

We prove hardness of MVPP optional-grabbing pawn games by reduction through a class of games that we introduce and call *Lock & Key* games, and may be of independent interest. A Lock & Key game is  $\mathcal{G} = \langle V, E, T, L, K, \lambda, \kappa \rangle$ , where  $\langle V, E, T \rangle$  is a turn-based game,  $L = \{\ell_1, \dots, \ell_n\}$  is a set of locks  $K = \{k_1, \dots, k_n\}$  is a set of keys, each  $\ell_j$  is associated to key  $k_j \in K$  for  $j \in [n]$ , and each edge is labeled by a set of locks and keys respectively given by  $\lambda : E \rightarrow 2^L$  and  $\kappa : E \rightarrow 2^K$ . Note that a lock and a key can appear on multiple edges.

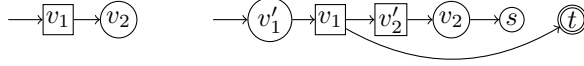
Intuitively, a Lock & Key game is a turn-based game, only that the locks impose restrictions on the edges that a player is allowed to cross. Formally, a configuration of a Lock & Key game is  $c = \langle v, A \rangle \in V \times 2^L$ , meaning that the token is placed on  $v$  and each lock in  $A$  is *closed* (all other locks are *open*). When  $v \in V_i$ , for  $i \in \{1, 2\}$ , then Player  $i$  moves the token as in turn-based games with the restriction that he cannot choose an edge that is labeled by a closed lock, thus  $e = \langle v, u \rangle \in E$  is a legal move at  $c$  when  $\lambda(e) \subseteq (L \setminus A)$ . Crossing  $e$  updates the configuration of the locks by “turning” all keys that  $e$  is labeled with. Formally, let  $\langle u, A' \rangle$  be the configuration after crossing  $e$ . For  $k_j \in \kappa(e)$  (“key  $k_j$  is turned”), we have  $\ell_j \in A$  iff  $\ell_j \notin A'$ . For  $k_j \notin \kappa(e)$  (“key  $k_j$  is unchanged”), we have  $\ell_j \in A$  iff  $\ell_j \in A'$ .

Note that, similar to pawn games, each Lock & Key game corresponds to an exponentially sized two-player turn-based game. Thus, membership in EXPTIME is immediate. For the lower bound, we show a reduction for the problem of deciding whether an *alternating polynomial-space Turing machine* (ATM) accepts a given word.

► **Theorem 8.** *Given a Lock & Key game  $\mathcal{G}$  and an initial configuration  $c$ , deciding whether Player 1 wins from  $c$  in  $\mathcal{G}$  is EXPTIME-complete.*

**Proof.** We briefly describe the syntax and semantics of ATMs, see for example [28], for more details. An ATM is  $\mathcal{A} = \langle Q, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle$ , where  $Q$  is a collection of states that is partitioned into  $Q = Q_1 \cup Q_2$  owned by Player 1 and Player 2 respectively,  $\Gamma$  is a tape alphabet,  $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$  is a transition function,  $q_0, q_{acc}, q_{rej} \in Q$  are respectively an initial, accepting, and rejecting states. A configuration of  $\mathcal{A}$  is  $c = \langle q, i, \langle \gamma_1, \dots, \gamma_m \rangle \rangle$ , meaning that the control state is  $q$ , the head position is  $i$ , and  $\langle \gamma_1, \dots, \gamma_m \rangle$  is the tape content, where  $m$  is polynomial in the length of the input word  $w$ . In order to determine whether  $\mathcal{A}$  accepts  $w$  we construct a (succinctly-represented) turn-based game over the possible configurations of  $\mathcal{A}$ , the neighbors of a configuration are determined according to  $\delta$ , and, for  $i \in \{1, 2\}$ , Player  $i$  moves from states in  $Q_i$ . We say that  $\mathcal{A}$  accepts  $w$  iff Player 1 has a winning strategy from the initial configuration for the target state  $q_{acc}$ .

Given  $\mathcal{A}$  and  $w$ , we construct a Lock & Key game  $\mathcal{G} = \langle V, E, T, L, K, \lambda, \kappa \rangle$  and an initial configuration  $\langle v_0, A \rangle$  such that Player 1 wins from  $\langle v, A \rangle$  in  $\mathcal{G}$  iff  $w$  is accepted by  $\mathcal{A}$ . The vertices of  $\mathcal{G}$  consist of *main* and *intermediate* vertices. Consider a configuration  $c = \langle q, i, \langle \gamma_1, \dots, \gamma_m \rangle \rangle$  of  $\mathcal{A}$ . We simulate  $c$  in  $\mathcal{G}$  using  $c' = \langle v, A \rangle$  as follows. First, the main vertices are  $Q \times \{1, \dots, m\} \times \Gamma$  and keep track of the control state and position on the tape. The main vertex that simulates  $c = \langle q, i, \langle \gamma_1, \dots, \gamma_m \rangle \rangle$  is  $v = \langle q, i, \gamma_i \rangle$ . We define  $v \in V_i$  iff  $q \in Q_i$ . Second, we use locks to keep track of the tape contents. For each  $1 \leq i \leq m$  and  $\gamma \in \Gamma$ , we introduce a lock  $\ell_{i, \gamma}$ . Then, in the configuration  $c' = \langle v, A \rangle$  that simulates  $c$ , the only locks that are open are  $\ell_{i, \gamma_i}$ , for  $i \in \{1, \dots, m\}$ . Next, we describe the transitions, where intermediate vertices are used for book-keeping. The neighbors of a main vertex  $v$  are the intermediate vertices  $\{\langle v, t \rangle : t \in \delta(q, \gamma)\}$ , where a transition of  $\mathcal{A}$  is  $t = \langle q', \gamma', B \rangle$ , meaning that the next control state is  $q'$ , the tape head moves to  $i + 1$  if  $B = R$  and to  $i - 1$  if  $B = L$ , and the  $i$ -th tape content changes from  $\gamma$  to  $\gamma'$ . We update the state of the locks so that they reflect the tape contents: for the edge  $\langle v, \langle v, t \rangle \rangle$ , we have  $\kappa(\langle v, \langle v, t \rangle \rangle) = \{k_{i, \gamma}, k_{i, \gamma'}\}$ .



■ **Figure 2** From turn-based to optional-grabbing games.

That is, traversing the edge turn the keys to close  $\ell_{i,\gamma}$  and open  $\ell_{i,\gamma'}$ . The neighbors of  $\langle v, t \rangle$  are main vertices having control state  $q'$  and head position  $i'$ . Recall that the third component of a main vertex is the tape content at the current position. We use the locks' state to prevent moving to main vertices with incorrect tape content: outgoing edges from  $\langle v, t \rangle$  are of the form  $\langle \langle v, t \rangle, \langle q', i', \gamma'' \rangle \rangle$  and is labeled by the lock  $\ell_{i',\gamma''}$ . That is, the edge can only be traversed when the  $i'$ -th tape position is  $\gamma''$ . It is not hard to verify that there is a one-to-one correspondence between runs of  $\mathcal{A}$  and plays of  $\mathcal{G}$ . Thus, Player 1 forces  $\mathcal{A}$  to reach a configuration with control state  $q_{acc}$  iff Player 1 forces to reach a main vertex with control state  $q_{acc}$ . Note that the construction is clearly polynomial since  $\mathcal{G}$  has  $|Q| \cdot m \cdot |\Gamma|$  main vertices. ◀

### 3.2.1 From Lock & Key games to optional-grabbing pawn games

Throughout this section, fix a Lock & Key game  $\mathcal{G}$  and an initial configuration  $c$ . We construct an optional-grabbing pawn game  $\mathcal{G}'$  over a set of pawns  $[d]$ , and identify an initial configuration  $c'$  such that Player 1 wins in  $\mathcal{G}$  from  $c$  iff Player 1 wins from  $c'$  in  $\mathcal{G}'$ .

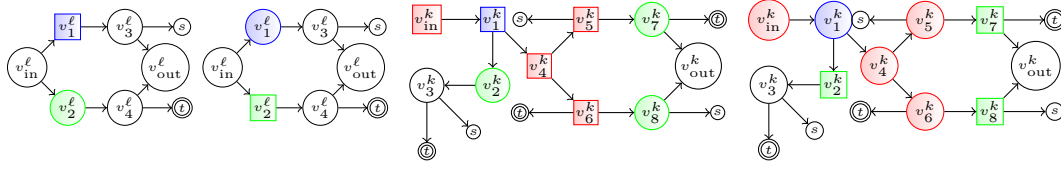
#### From turn-based games to optional-grabbing games

In this section, we consider the case in which  $\mathcal{G}$  has no keys or locks, thus  $\mathcal{G}$  is a turn-based game. The reduction is depicted in Fig. 2. Denote the turn-based game  $\mathcal{G} = \langle V, E, T \rangle$  with  $V = V_1 \cup V_2$  and initial vertex  $v_0$ . We construct an OVPP optional-grabbing  $\mathcal{G}' = \langle V', E', T' \rangle$ , where  $V' = V \cup \{v' : v \in V\} \cup \{s, t\}$ . We add edges to ensure that the player who owns a vertex  $v \in V$  is the player who moves from  $v$  in  $\mathcal{G}'$ : we have  $\langle v', v \rangle \in E'$ , and if  $v \in V_1$ , then  $\langle v, s \rangle \in E'$ , and if  $v \in V_2$ , then  $\langle v, t \rangle \in E'$ . We redirect each edge  $\langle u, v \rangle$  in  $\mathcal{G}$  to  $\langle u, v' \rangle$  in  $\mathcal{G}'$ . Intuitively, for  $v \in V_1$ , a Player 1 winning strategy will guarantee that  $v'$  is always in the control of Player 2, and following her move at  $v'$ , Player 1 must grab  $v$  otherwise Player 2 wins and choose the next location. And dually for  $v \in V_2$ . Let  $V'_1 = V_1 \cup \{v' : v \in V_2\}$ , the initial configuration of  $\mathcal{G}'$  is  $\langle v_0, V'_1 \rangle$ , that is Player 2 controls  $V_2 \cup \{v' : v \in V_1\}$ . Formally, we prove the following in the full version.

► **Lemma 9.** *For a turn-based game  $\mathcal{G}$ , Player 1 wins  $\mathcal{G}$  from a vertex  $v_0 \in V$  iff Player 1 wins the optional-grabbing game  $\mathcal{G}'$  from configuration  $\langle v_0, V'_1 \rangle$ .*

#### Gadgets for simulating locks and keys

For each lock  $\ell \in L$  and its corresponding key  $k \in K$ , we construct gadgets  $\mathcal{G}_\ell$  and  $\mathcal{G}_k$  that simulate the operations of  $\ell$  and  $k$  in  $\mathcal{G}'$ . The gadgets in two *states* are depicted in Fig. 3. We highlight three pawns colored blue, green, and red, respectively owning,  $\{v_1^\ell, v_1^k\}$ ,  $\{v_2^\ell, v_2^k, v_7^k, v_8^k\}$ , and  $\{v_{in}^k, v_4^k, v_5^k, v_6^k\}$ . Each of the other vertices (colored white) is owned by a fresh pawn. Intuitively, for each lock  $\ell$ , we identify two sets  $\mathcal{P}_O^\ell, \mathcal{P}_C^\ell \subseteq 2^{[d]}$ , respectively representing an open and closed state of  $\ell$ . We will ensure that when entering and exiting a gadget, the configuration is in  $\mathcal{P}_O^\ell \cup \mathcal{P}_C^\ell$ . When the set of pawns that Player 1 controls is in  $\mathcal{P}_O^\ell$  and  $\mathcal{P}_C^\ell$ , we respectively say that  $\mathcal{G}_\ell$  is in open and closed state, and similarly for  $\mathcal{G}_k$  as stated below. Formally, we define  $\mathcal{P}_O^\ell = \{P \in 2^{[d]} : v_1^\ell \notin P \wedge v_2^\ell \in P\}$  and  $\mathcal{P}_C^\ell = \{P \in 2^{[d]} : v_1^\ell \in P \wedge v_2^\ell \notin P\}$ .



■ **Figure 3** From left to right:  $\mathcal{G}_\ell$  in open and closed state and  $\mathcal{G}_k$  in open and closed state.

► **Lemma 10.** *Let  $i \in \{1, 2\}$ . An open lock stays open: If Player  $i$  enters  $\mathcal{G}_\ell$  in  $\mathcal{P}_O^\ell$ , then he has a strategy that guarantees that either he wins  $\mathcal{G}'$  or  $\mathcal{G}_\ell$  is exited in  $\mathcal{P}_O^\ell$ . A closed lock cannot be crossed: If Player  $i$  enters  $\mathcal{G}_\ell$  in  $\mathcal{P}_C^\ell$ , then Player  $-i$  has a strategy that guarantees that Player  $i$  loses  $\mathcal{G}'$ .*

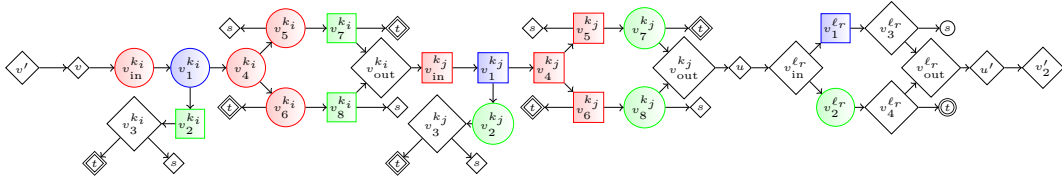
**Proof.** We prove for Player 1 and the proof is dual for Player 2. First, suppose Player 1 enters  $\mathcal{G}_\ell$  in  $\mathcal{P}_O^\ell$ . Player 2 may or may not grab  $v_{in}^\ell$ , and the game can proceed to either  $v_1^\ell$  or  $v_2^\ell$ . We argue that if the game proceeds to  $v_1^\ell$ , then Player 1 will not grab  $v_1^\ell$ . We can also similarly show that if the game proceeds to  $v_2^\ell$ , then Player 2 will not grab  $v_2^\ell$ . Player 2 controls  $v_1^\ell$ . We claim that if Player 1 grabs  $v_1^\ell$ , he will lose the game. Indeed, following Player 1's move in  $v_1^\ell$ , Player 2 will grab  $v_3^\ell$  and move the token to the sink vertex  $s$  to win the game. Thus, Player 1 does not grab  $v_1^\ell$  and keeps it in the control of Player 2. Following Player 2's move in  $v_1^\ell$ , Player 1 grabs  $v_3^\ell$  and proceeds to exit  $\mathcal{G}_\ell$ . Note that when  $\mathcal{G}_\ell$  is exited, Player 1 maintains control of  $v_2^\ell$  and Player 2 maintains control of  $v_1^\ell$ , thus the configuration is in  $\mathcal{P}_O^\ell$ . Second, suppose that Player 1 enters  $\mathcal{G}_\ell$  in  $\mathcal{P}_C^\ell$ . Then, Player 2 grabs  $v_{in}^\ell$  and moves the token to  $v_1^\ell$ . Since Player 1 controls  $v_1^\ell$  he must make the next move. Player 2 then grabs  $v_3^\ell$  and moves the token to  $s$  to win the game. ◀

Next, we present the gadget  $\mathcal{G}_k$  for simulating the operation of a key  $k$  (see Fig. 3). Intuitively, we maintain that  $\mathcal{G}_k$  is in open state iff  $\mathcal{G}_\ell$  is in open state, and traversing  $\mathcal{G}_k$  swaps the state of both. We define sets of configurations  $\mathcal{P}_O^k = \{P \in 2^{[d]} : \{v_{in}^k, v_1^k, v_4^k, v_5^k, v_6^k\} \cap P = \emptyset \wedge \{v_2^k, v_7^k, v_8^k\} \subseteq P\}$  and  $\mathcal{P}_C^k = \{P \in 2^{[d]} : \{v_{in}^k, v_1^k, v_4^k, v_5^k, v_6^k\} \subseteq P \wedge \{v_2^k, v_7^k, v_8^k\} \cap P = \emptyset\}$  (see Fig. 3). Note that  $\mathcal{P}_O^k \subseteq \mathcal{P}_O^\ell$  and  $\mathcal{P}_C^k \subseteq \mathcal{P}_C^\ell$  since  $v_i^k$  and  $v_i^\ell$  are owned by the same pawn for  $i \in [2]$ . In the full version, we prove the following.

► **Lemma 11.** *Turning  $k$  closes an open  $\ell$ : Let  $i \in \{1, 2\}$ . If Player  $i$  enters  $\mathcal{G}_k$  in  $\mathcal{P}_O^k$ , then he has a strategy that ensures that either Player  $i$  wins  $\mathcal{G}'$  or  $\mathcal{G}_k$  is exited in  $\mathcal{P}_C^k$ . Turning  $k$  opens a closed  $\ell$ : when Player  $i$  enters  $\mathcal{G}_k$  in  $\mathcal{P}_C^k$ , Player  $i$  ensures that either he wins  $\mathcal{G}'$  or  $\mathcal{G}_k$  is exited in  $\mathcal{P}_O^k$ .*

### Putting it all together

We describe the construction of a pawn game  $\mathcal{G}'$  from a Lock & Key game  $\mathcal{G}$ . We assume w.l.o.g. that each edge  $\langle u, v \rangle$  in  $\mathcal{G}$  is labeled by at most one lock or key since an edge that is labeled by multiple locks or keys can be split into a chain of edges, each labeled by a single lock or a key. We describe the construction of  $\mathcal{G}'$ . We first apply the construction for turn-based games on  $\mathcal{G}$  while “ignoring” the locks and keys. Recall that the construction introduces fresh vertices so that an edge  $e = \langle u, v \rangle$  in  $\mathcal{G}$  is mapped to an edge  $e' = \langle u', v \rangle$  in  $\mathcal{G}'$ . We re-introduce the locks and keys so that the labeling of  $e'$  coincides with the labeling of  $e$ . Next, we replace an edge  $e'$  that is labeled by a lock  $\ell$ , by a copy of  $\mathcal{G}_\ell$ , and if  $e$  is labeled by a key  $k$ , we replace  $e'$  by a copy of  $\mathcal{G}_k$ . Note that multiple edges could be labeled by the same lock  $\ell$ . In such a case we use fresh vertices in each copy of  $\mathcal{G}_\ell$ , but crucially, all gadgets



■ **Figure 4** A  $\delta$ -path is a path between two primed main vertices in an optional- or always-grabbing game, and it crosses two key gadgets and one lock gadget.

share the same pawns so that they share the same state. And similarly for keys. For an illustration of this construction, see Fig. 4, which applies the construction on a Lock & Key game that is output from the reduction in Thm. 8.

Finally, given an initial configuration  $c = \langle v, A \rangle$  of  $\mathcal{G}$  we define an initial configuration  $c' = \langle v, P \rangle$  of  $\mathcal{G}'$ . Note that the initial vertex is the entry point of the gadget that simulates  $v$  in  $\mathcal{G}'$ . For each lock  $\ell$  and corresponding key  $k$ , if  $\ell$  is open according to  $A$ , then  $P \in \mathcal{P}_O^\ell$ , i.e., both  $\mathcal{G}_\ell$  and  $\mathcal{G}_k$  are initially in open state. And similarly when  $\ell$  is closed according to  $A$ . Combining the properties in Lemmas 9, 10, and 11 implies that Player 1 wins  $\mathcal{G}$  from  $c$  iff Player 1 wins  $\mathcal{G}'$  from  $c'$ . Thus, by Thm. 8, we have the following.

► **Theorem 12.** *MVPP optional-grabbing PAWN-GAMES is EXPTIME-complete.*

#### 4 Always-Grabbing Pawn Games

In this section, we study always-grabbing pawn games and show that MVPP always-grabbing pawn-games are EXPTIME-complete. The main challenge is proving the lower bound. We proceed as follows. Let  $\mathcal{M}$  be an ATM. Apply the reduction in Thm 8 and the one in Thm. 12 to obtain pairs  $\langle \mathcal{G}, c \rangle$  and  $\langle \mathcal{G}', c' \rangle$ , where  $\mathcal{G}$  and  $\mathcal{G}'$  are respectively Lock & Key and optional-grabbing games with initial configurations  $c$  and  $c'$  respectively. We devise a construction that takes  $\langle \mathcal{G}', c' \rangle$  and produces an always-grabbing game  $\mathcal{G}''$  and a configuration  $c''$  such that Player 1 wins from  $c''$  in  $\mathcal{G}''$  iff he wins from  $c$  in  $\mathcal{G}$ .

Our analysis heavily depends on the special structure of  $\mathcal{G}'$ . The construction in Thm. 8 outputs a game  $\mathcal{G}$  with main vertices of the form  $\langle q, i, \gamma \rangle$  ( $q$  is a state,  $i$  is a tape position, and  $\gamma$  is a letter in the tape alphabet). A play of  $\mathcal{G}$  can be partitioned into paths between main vertices. Each such path corresponds to one transition of the Turing machine and traverses two keys and a lock before again reaching a main vertex. Recall that when constructing  $\mathcal{G}'$  from  $\mathcal{G}$ , we replace locks and keys with their respective gadgets, and for every vertex  $v$  that belongs to  $\mathcal{G}$ , we add a new primed vertex  $v'$  such that if  $v$  is controlled by Player  $i$  then  $v'$  is controlled by Player  $-i$ . We call a path in  $\mathcal{G}'$  that corresponds to a path in  $\mathcal{G}$  between two successive main vertices, say  $v$  and  $v'$ , a  $\delta$ -path. Fig. 4 depicts a  $\delta$ -path. An important property of the specific optional-grabbing game  $\mathcal{G}'$  that is constructed in Thm 8 from an ATM is that every play of  $\mathcal{G}'$  consists of a sequence of  $\delta$ -paths. More details on  $\delta$ -paths can be found in the full version. The following observation can easily be verified:

► **Observation 13.** *A  $\delta$ -path from  $v'$  to  $v'_2$  consists of 20 turns.*

The following lemma is crucial for the construction of  $\mathcal{G}''$ .

► **Lemma 14.** *For  $i \in \{1, 2\}$ , if Player  $i$  has a strategy in the optional-grabbing game  $\mathcal{G}'$  to cross a  $\delta$ -path from  $v'$  to  $v'_2$ , then Player  $i$  has a strategy that moves the token in at least 10 rounds and Player  $-i$  moves the token in at most 10 rounds in the  $\delta$ -path.*

Let  $\mathcal{G}' = \langle V', E', T' \rangle$  with  $d$  pawns. The game  $\mathcal{G}''$  is constructed from  $\mathcal{G}'$  by adding  $2(d+10)$  fresh isolated vertices each owned by a fresh unique pawn. Formally,  $\mathcal{G}'' = \langle V'', E', T' \rangle$ , where  $V'' = V' \cup \{v_1, v_2, \dots, v_{2(d+10)}\}$  such that  $v_j \notin V'$ , for  $j \in [2(d+10)]$ . Consider a configuration  $c' = \langle v, P \rangle$  in  $\mathcal{G}'$ . Let  $c'' = \langle v, P \cup \{1, 2, \dots, d+10\} \rangle$  be a configuration in  $\mathcal{G}''$ . Note that Lemma 14 also applies to the always-grabbing game  $\mathcal{G}''$ , and we get the following.

► **Corollary 15.** *For  $i \in \{1, 2\}$ , if Player  $i$  has a strategy in the always-grabbing game  $\mathcal{G}''$  to cross a  $\delta$ -path from  $v'$  to  $v'_2$ , then Player  $i$  has a strategy such that out of the 20 rounds in the  $\delta$ -path, the following hold.*

1. Player  $-i$  grabs a pawn in at least 10 rounds, and
2. Player  $i$  grabs a pawn in at most 10 rounds.

Corollary 15 follows directly from Lemma 14 since in an always-grabbing game, the number of times Player  $-i$  grabs equals the number of times Player  $i$  moves. In the remaining part of this section, we show that Player 1 wins  $\mathcal{G}'$  from  $c'$  iff Player 1 wins  $\mathcal{G}''$  from the configuration  $c''$  described above.

► **Lemma 16.** *For  $i \in \{1, 2\}$ , Player  $i$  wins from  $c'$  in the optional-grabbing game  $\mathcal{G}'$  iff he wins from  $c''$  in the always-grabbing game  $\mathcal{G}''$ .*

**Proof sketch.** We prove that if Player 1 has a winning strategy  $f'$  in  $\mathcal{G}'$  from  $c'$ , then he has a winning strategy  $f''$  from  $c''$  in  $\mathcal{G}''$ . The case for Player 2 is analogous and the other direction follows from determinacy (Thm. 2). We construct  $f''$  to mimic  $f'$  with the following difference. Whenever  $f'$  chooses not to grab, in order to follow the rules of the always-grabbing mechanism,  $f''$  grabs a pawn owning an isolated vertex. This is possible since we show that we maintain the invariant that along a play in  $\mathcal{G}''$  that consists of sequences of  $\delta$ -paths, at the beginning of each  $\delta$ -path, Player 2 has at least 10 isolated pawns. Note that the invariant holds initially due to the definition of  $c''$ . We show that it is maintained. Recall from the proof of Theorem 8 that crossing a  $\delta$ -path simulates a transition in the Turing machine. Since Player 1 has a winning strategy in  $\mathcal{G}'$ , in a winning play, the strategy enables her to cross the  $\delta$ -path. Thus, by Lem. 14, Player 1 moves in at least 10 rounds. Thus, Player 2 moves in at most 10 rounds, and during each such round, Player 1 grabs a pawn. Hence, Player 1 grabs at most 10 times which thus maintains the invariant. In the full version, we show that  $f''$  is a winning Player 1 strategy. ◀

We now state the following theorem. While the lower bound follows from Thm. 12 and Lem. 16, the upper bound follows from Thm. 4.

► **Theorem 17.** *MVPP always-grabbing PAWN-GAMES is EXPTIME-complete.*

We conclude this section by adapting Thm. 5 to always-grabbing. Namely, we show that adding pawns to a player is never beneficial in MVPP always-grabbing games. (with the exception of the pawn that owns the current vertex). The proof can be found in the full version.

► **Theorem 18.** *Consider a configuration  $\langle v, P \rangle$  of an MVPP always-grabbing pawn game  $\mathcal{G}$ . Let  $j \in [d]$  such that  $v \in V_j$  and  $P' \subseteq P$ . Assuming that  $j \in P$  implies  $j \in P'$ , if Player 1 wins from  $\langle v, P \rangle$ , he wins from  $\langle v, P' \rangle$ . Assuming that  $j \in \overline{P'}$  implies  $j \in \overline{P}$ , if Player 2 wins from  $\langle v, P' \rangle$ , she wins from  $\langle v, P \rangle$ .*

## 5 Always Grabbing-or-Giving Pawn Games

In this section, we show that MVPP always grabbing or giving games are in PTIME. We find it intriguing that a seemingly small change in the mechanism – allowing a choice between grabbing and giving instead of only grabbing – reduces the complexity to PTIME from EXPTIME-complete. We make the following simple observation.

► **Observation 19.** *In an always grabbing or giving game, every time Player  $i$  makes a move from a vertex  $v$  to a vertex  $u$ , Player  $-i$  can decide which player controls  $u$ .*

If Player  $-i$  does not control  $p_u$  that owns  $u$  and he wants to control  $u$ , he can grab  $p_u$  from Player  $i$ . If he does not want to control  $u$  and if he has  $p_u$ , he can give it to Player  $i$ .

Consider an always-grabbing-or-giving game  $\mathcal{G} = \langle V, E, T \rangle$  and an initial configuration  $c$ . We construct a turn-based game  $\mathcal{G}'$  and an initial vertex  $v_0$  so that Player 1 wins in  $\mathcal{G}$  from  $c$  iff he wins in  $\mathcal{G}'$  from  $v_0$ . Let  $\mathcal{G}' = \langle V', E', T' \rangle$ , where  $V' = \{\langle v, i \rangle, \langle \widehat{v}, i \rangle \mid v \in V, i \in \{1, 2\}\}$  with  $V'_1 = \{\langle v, 1 \rangle, \langle \widehat{v}, 1 \rangle \mid v \in V\}$  and  $V'_2 = \{\langle v, 2 \rangle, \langle \widehat{v}, 2 \rangle \mid v \in V\}$ ,  $T' = T \times \{1, 2\}$ , and  $E' = \{(\langle v, i \rangle, \langle \widehat{u}, 3-i \rangle), (\langle \widehat{u}, 3-i \rangle, \langle u, i \rangle), (\langle \widehat{u}, 3-i \rangle, \langle u, 3-i \rangle) \mid (v, u) \in E, i \in \{1, 2\}\}$ . We call each vertex  $\langle v, i \rangle$  a *main* vertex and each  $\langle \widehat{v}, i \rangle$  an *intermediate* vertex. Suppose that Player  $i$  moves the token from  $v$  to  $u$  in  $\mathcal{G}$ . If Player  $-i$  decides to control  $u$ , then in  $\mathcal{G}'$ , the token moves from the main vertex  $\langle v, i \rangle$  to the main vertex  $\langle u, 3-i \rangle$ , else from  $\langle v, i \rangle$  to the main vertex  $\langle u, i \rangle$ , and in each case, through the intermediate vertex  $\langle \widehat{u}, 3-i \rangle$  that models the decision of Player  $-i$  on the control of  $u$ . The target vertices  $T'$  are main vertices. The proof of the following lemma appears in the full version.

► **Lemma 20.** *Suppose Player 1 wins from configuration  $\langle v, P \rangle$  in  $\mathcal{G}$ . If he controls  $v$ , he wins from  $\langle v, 1 \rangle$  in  $\mathcal{G}'$ , and if Player 2 controls  $v$ , Player 1 wins from  $\langle v, 2 \rangle$  in  $\mathcal{G}'$ . Dually, suppose that Player 2 wins from  $\langle v, P \rangle$  in  $\mathcal{G}$ . If she controls  $v$ , then she wins from  $\langle v, 2 \rangle$  in  $\mathcal{G}'$ , and if Player 1 controls  $v$ , Player 2 wins from  $\langle v, 1 \rangle$  in  $\mathcal{G}'$ .*

Since the size of  $\mathcal{G}'$  is polynomial in the size of  $\mathcal{G}$ , Thm. 2 implies the following.

► **Theorem 21.** *MVPP always-grab-or-give PAWN-GAMES is in PTIME.*

## 6 $k$ -Grabbing Pawn Games

In this section, we consider pawn games under  $k$ -grabbing in increasing level of generality of the mechanisms. We start with positive news.

► **Theorem 22.** *OVPP  $k$ -grabbing PAWN-GAMES is in PTIME.*

**Proof.** Let  $k \in \mathbb{N}$ , an OVPP  $k$ -grabbing game  $\mathcal{G} = \langle V, E, T \rangle$ , and an initial configuration  $c = \langle v_0, P_0 \rangle$ , where we refer to  $P_0$  as a set of vertices rather than pawns. For a vertex  $u \in V$ , let  $\eta(u)$  denote the minimum number of grabs with which Player 1 can guarantee winning  $\mathcal{G}$  from configuration  $\langle u, P_0 \rangle$ . The algorithm recursively computes  $\eta$  based on repeated calls to an algorithm to solve turn-based games (see Thm. 2).

For the base case, consider the turn-based game  $\mathcal{G}_0 = \langle V, E, T \rangle$  with  $V_1 = P_0$ . Let  $W_0^1 \subseteq V$  denote Player 1's winning region in  $\mathcal{G}_0$ . Clearly, for every  $u \in W_0^1$ , we have  $\eta(v_0) = 0$ , and for every  $u \notin W_0^1$ , we have  $\eta(v_0) \geq 1$ . For the inductive step, suppose that for  $\ell \geq 0$ , the set  $W_\ell^1 = \{u \in V : \eta(u) \leq \ell\}$  has been found. That is, for every  $u \notin W_\ell^1$ , Player 2 has a strategy that wins the  $\ell$ -grabbing pawn game  $\mathcal{G}$  from configuration  $\langle u, P_0 \rangle$ . We show how to find  $W_{\ell+1}^1$  in linear time. Let the *border* of  $W_\ell^1$ , denoted  $B_\ell$ , be the set of vertices from which  $W_\ell^1$  can be reached in one step, thus  $B_\ell = \{v \in V : v \notin W_\ell^1 : N(v) \cap W_\ell^1 \neq \emptyset\}$ .

Note that the vertices in  $B_\ell$  are all controlled by Player 2 since otherwise, such vertices will be in the set  $W_\ell^1$ . In the full version, we show that a vertex  $u \notin W_\ell^1$  has  $\eta(u) = \ell + 1$  iff Player 1 can force the game from configuration  $\langle u, P_0 \rangle$  to a vertex in  $B_\ell$  in one or more rounds without making any grab. Player 1 wins from such a vertex  $u$  by forcing the game into  $B_\ell$ , grabbing the pawn in  $B_\ell$ , and proceeding to  $W_\ell$ , where by the induction hypothesis, he wins with the remaining grabs. Computing  $W_{\ell+1}^1$  roughly entails a solution to a turn-based game with target set  $B_\ell \cup W_\ell^1$ . ◀

The proof of the following theorem, which can be found in the full version, is obtained by a reduction from SET-COVER.

▶ **Theorem 23.** *MVPP  $k$ -grabbing game PAWN-GAMES is NP-hard.*

We conclude this section by studying OMVPP games.

▶ **Lemma 24.** *OMVPP  $k$ -grabbing PAWN-GAMES is PSPACE-hard.*

**Proof.** Consider an input  $\phi = Q_1 x_1 \dots Q_n x_n C_1 \wedge \dots \wedge C_m$  to TQBF, where  $Q_i \in \{\exists, \forall\}$ , for  $1 \leq i \leq n$ , each  $C_j$ , for  $1 \leq j \leq m$ , is a clause over the variables  $x_1, \dots, x_n$ . We construct an OMVPP  $n$ -grabbing pawn game  $\mathcal{G} = \langle V, E, T \rangle$  such that Player 1 wins iff  $\phi$  is true. We describe the intuition and the details can be found in the full version. The structure of  $\mathcal{G}$  is chain-like. Player 1 needs to cross the chain in order to win. The first part of the chain requires Player 1 to grab, for each variable  $x_i$ , either a pawn  $p_i$  or a pawn  $\neg p_i$ . For existentially-quantified variables, Player 1 decides which of the two is grabbed, and for universally-quantified variables, Player 2 decides. In the second part of  $\mathcal{G}$ , we verify that the corresponding assignment is valid. Certain positions of  $\mathcal{G}$  correspond to a clause  $C_j$ , for  $j \in [m]$ , which Player 1 must take control over during the first part of  $\mathcal{G}$ . The key is to use OMVPP: We define that if  $x_i$  appears in  $C_j$ , then  $p_i$  is an owner of  $C_j$ , and if  $\neg x_i$  appears in  $C_j$ , then  $\neg p_i$  is an owner of  $C_j$ . Thus, Player 1 controls  $C_j$  iff he grabbed a pawn that owns  $C_j$  which is iff the assignment satisfies  $C_j$ . ◀

We turn to study the upper bound. The following lemma bounds the provides a polynomial bound on the length of a winning play for Player 1. The core of the proof, which can be found in the full version, intuitively shows that we can restrict attention to Player 1 strategies that grab at least once in a sequence of  $|V|$  rounds. Otherwise, the game enters a cycle that is winning for Player 2.

▶ **Lemma 25.** *Consider an OMVPP  $k$ -grabbing PAWN-GAME  $\mathcal{G} = \langle V, E, T \rangle$ , and an initial configuration  $c$  that is winning for Player 1. Then, Player 1 has a strategy such that, for every Player 2 strategy, a target in  $T$  is reached within  $|V| \cdot (k + 1)$  rounds.*

For the upper bound, in the full version, we describe an algorithm performing a depth-first traversal of the configuration graph of a game while storing, at a time, only a branch in PSPACE. By Lem. 25, each branch of such a traversal has polynomial length, leading to the PSPACE upper bound. We thus have the following.

▶ **Theorem 26.** *OMVPP  $k$ -grabbing PAWN-GAMES is PSPACE-complete.*

## 7 Discussion

We introduce pawn games, a class of two-player turn-based games in which control of vertices changes dynamically throughout the game. Pawn games constitute a class of succinctly-represented turn-based games. We identify natural classes that are in PTIME.

Our EXPTIME-hardness results are based on Lock & Key games, which we hope will serve as a framework for proving lower bounds. We mention directions for future research. First, we leave several open problems; e.g., for MVPP  $k$ -grabbing pawn games, we only show NP-hardness and membership in PSPACE. Second, we focused on reachability games. It is interesting to study pawn games with richer objectives such as parity or quantitative objectives. Third, it is interesting to consider other pawn-transferring mechanisms and to identify properties of mechanisms that admit low-complexity results. Finally, grabbing pawns is a general concept and can be applied to more involved games like stochastic or concurrent games.

---

## References

- 1 M. Aghajohari, G. Avni, and T. A. Henzinger. Determinacy in discrete-bidding infinite-duration games. *Log. Methods Comput. Sci.*, 17(1), 2021.
- 2 P. Alizadeh Alamdari, G. Avni, T. A. Henzinger, and A. Lukina. Formal methods with a touch of magic. In *Proc. 20th FMCAD*, 2020.
- 3 S. Almagor, G. Avni, and O. Kupferman. Repairing multi-player games. In *Proc. 26th CONCUR*, pages 325–339, 2015.
- 4 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
- 5 K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.
- 6 G. Avni, R. Bloem, K. Chatterjee, T. A. Henzinger, B. Könighofer, and S. Pranger. Run-time optimization for learned controllers through quantitative games. In *Proc. 31st CAV*, pages 630–649, 2019.
- 7 G. Avni, P. Ghorpade, and S. Guha. A game of pawns. *CoRR*, abs/2305.04096, 2023. [arXiv:2305.04096](https://arxiv.org/abs/2305.04096).
- 8 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. *J. ACM*, 66(4):31:1–31:29, 2019.
- 9 G. Avni and S. Sadhukhan. Computing threshold budgets in discrete-bidding games. In *Proc. 42nd FSTTCS*, volume 250 of *LIPICs*, pages 30:1–30:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- 10 G. Bielous and O. Kupferman. Coverage and vacuity in network formation games. In *Proc. 28th CSL*, 2020.
- 11 R. Brenguier, L. Clemente, P. Hunter, G. A. Pérez, M. Randour, J.-F. Raskin, O. Sankur, and M. Sassolas. Non-zero sum games for reactive synthesis. In *Proc. 10th LATA*, pages 3–23, 2016.
- 12 L. Brice, J.-F. Raskin, and M. van den Bogaard. Subgame-perfect equilibria in mean-payoff games. In *In Proc. 32nd CONCUR*, volume 203 of *LIPICs*, pages 8:1–8:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 13 D. Busatto-Gaston, D. Chakraborty, S. Guha, G. A. Pérez, and J.-F. Raskin. Safe learning for near-optimal scheduling. In *QEST, Proceedings*, volume 12846 of *Lecture Notes in Computer Science*, pages 235–254. Springer, 2021.
- 14 P. Cerný, E. M. Clarke, T. A. Henzinger, A. Radhakrishna, L. Ryzhyk, R. Samanta, and T. Tarrach. From non-preemptive to preemptive scheduling using synchronization synthesis. *Formal Methods Syst. Des.*, 50(2-3):97–139, 2017.
- 15 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.
- 16 M. Develin and S. Payne. Discrete bidding games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.
- 17 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th TACAS*, pages 190–204, 2010.



- 18 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
- 19 B. Könighofer, M. Alshiekh, R. Bloem, L. Humphrey, R. Könighofer, U. Topcu, and C. Wang. Shield synthesis. *Formal Methods in System Design*, 51(2):332–361, 2017.
- 20 O. Kupferman, G. Perelli, and M. Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.
- 21 L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
- 22 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.
- 23 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014.
- 24 F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In *Proc. 30th FSTTCS*, pages 133–144, 2010.
- 25 G. Perelli. Enforcing equilibria in multi-agent systems. In *In Proc. 18th AAMAS*, pages 188–196. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- 26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
- 27 S. A. Seshia, N. Sharygina, and S. Tripakis. Modeling for verification. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 75–105. Springer, 2018.
- 28 Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, 2013.
- 29 R. S. Sutton and A. G. Barto. *Reinforcement learning – An introduction*. Adaptive computation and machine learning. MIT Press, 1998.
- 30 M. Ummels. *Stochastic multiplayer games: theory and algorithms*. PhD thesis, RWTH Aachen University, 2011.
- 31 J. van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, pages 268–276, 2005.
- 32 M. J. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, and A. Toumi. Rational verification: From model checking to equilibrium checking. In *Proc. of the 30th AAAI*, pages 4184–4191, 2016.