Regular Language Inference for Learning Rules of Simplified Boardgames

Jakub Kowalski
Institute of Computer Science, *University of Wrocław*
Wrocław, Poland
jko@cs.uni.wroc.pl

Andrzej Kisielewicz
Institute of Mathematics, *University of Wrocław*
Wrocław, Poland
Andrzej.Kisielewicz@math.uni.wroc.pl

***Abstract*—We deal with the problem of learning game rules by observing the play, the study initiated by Björnsson for the class of Simplified Boardgames, describing a rich family of chess-like games. In this paper we restate the problem in terms of regular language inference and improve Björnsson's algorithm combining applications of existing DFA learning algorithms with our domain-specific approach. We present the results of tests on a number of games, including human-made and artificially generated ones.**

***Index Terms*—general game playing, regular language inference, simplified boardgames, deterministic finite automata**

I. Introduction

The aim of *General Game Playing* (GGP) [1] is to develop a system that can play a variety of games with previously unknown rules. Unlike standard AI game playing, where designing an agent requires special knowledge about the game, in GGP the key is to create a universal algorithm performing well in various situations and environments. After the launch of the annual International General Game Playing Competition (IGGPC) in 2005 [2], [3], many new languages have been developed to describe certain classes of games [4], [5], [6], [7] and other competitions have been proposed [8]. Identified as a new Grand Challenge of Artificial Intelligence, GGP consists of many research challenges requiring combining a number of domains, such as knowledge representation, searching, planning, reasoning, and machine learning [9].

In [4], Björnsson has proposed a new scenario, where the game rules should be learned by observing others play, rather than obtained from a given description. This partially coincides with the rules of GVG-AI competition [8], where a prepared reasoner (java object allowing state manipulation) is given instead of the raw game rules. Björnsson's approach concentrates on learning deterministic finite automata (DFA) which are used to encode learned rules in a class of games introduced by the author and called Simplified Boardgames. The proposed class is substantially narrower than GGP systems mentioned before, yet more general and concise than previous such approaches [10], [11]. As the DFA representation favors the efficiency of game state manipulation, possibilities of implementing such mechanism to support GGP players have been considered.

In this paper, we deal with the problem of learning game rules by observing. We continue the study initiated by Björnsson in [4] for the class of Simplified Boardgames, focusing on efficient learning of the observed boardgame moves. First, we consider applications of existing DFA learning algorithms for the task of learning piece movements from the set of game records and restate the problem in terms of Regular Language Inference [12]. Secondly, we propose our domain-specific algorithms to ensure better efficiency and a higher chance of obtaining a correct approximation of the actual DFA, assuming incompleteness of given training data. We test all presented approaches combining various learning algorithms with different procedures checking DFA consistency in a number of games, both human-made and artificially generated. Following Björnsson's approach we consider two types of training data: GGP-like, where for every position all legal moves are listed, and human-play-like, providing more sparse data where only the move actually made by a player has been recorded.

The paper is organized as follows. Section II provides the necessary background for the class of Simplified Boardgames, learning by observing, and Regular Language Inference. In Section III, we formally state the problem and analyze selected existing approaches from the theoretical point of view. In Section IV, we introduce new consistency checking procedures, and in Section V we present our new algorithm learning piece's moves by observing. Finally, Section VI presents the results of experiments evaluating the performance of all the considered approaches.

II. Preliminaries

In this section we introduce domains relevant to our study, providing necessary algorithms and terminology.

Let Σ^* be the set of all possible words over the alphabet Σ. For DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$, where Q is the set of states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to Q$ the transition function (which, in our study, is usually a partial function), by $L(\mathcal{A})$ we denote the language accepted by $\mathcal{A}$.

A. Simplified Boardgames

Simplified Boardgames is the class of fairy chess-like games introduced by Björnsson in [4]. The language describes turn-based, two-player, zero-sum chess-like games on a rectangular board with piece movements described by regular languages and independent on the move history. It was slightly extended in [13], and used as a comparison class for assessing the level of Stanford's GGP programs.

Here we follow the class formalization from [14] to provide a shortened necessary introduction. The game is played between the two players, *black* and *white*, on a rectangular board of size *width*×*height*. White player is always the first to move. Although it may be asymmetric, the initial position is given from the perspective of the white player, i.e. forward means "up" for white, and "down" for black.

During a single turn, the player has to make a move using one of his pieces. Making a move is done by choosing the piece and changing its position according to the specified movement rule for this piece. At any time, at most one piece can occupy a square, so finishing the move on a square containing a piece (regardless of the owner) results in removing it (capturing). No piece addition is possible. After performing a move, the player gives control to the opponent.

For a given piece, the set of its legal moves is defined as the set of words described by a regular expression over an alphabet Σ containing triplets $(\Delta x, \Delta y, on)$, where Δx and Δy are relative column/row distances, and $on \in \{e, p, w\}$ describes the content of the destination square: e indicates an empty square, p a square occupied by an opponent piece, and w a square occupied by an own piece. A positive Δy means forward, which is a subjective direction and differs in meaning depending on the player.

Consider a piece and a word $w \in \Sigma^*$ that belongs to the language described by the regular expression in the movement rule for this piece. Let $w = a_1 a_2 \dots a_k$, where each $a_i = (\Delta x_i, \Delta y_i, on_i)$, and suppose that the piece stands on a square $\langle x, y\rangle$. Then, w describes a move of the piece, which is applicable in the current board position if and only if, for every i such that $1 \le i \le k$, the content condition on_i is fulfilled by the content of the square $\langle x + \sum_{j=1}^{i} \Delta x_j, y + \sum_{j=1}^{i} \Delta y_j\rangle$. The move of w changes the position of the piece piece from $\langle x, y\rangle$ to $\langle x + \sum_{i=1}^{k} \Delta x_i, y + \sum_{i=1}^{k} \Delta y_i\rangle$. An example of how move rules work is shown in Figure 1.

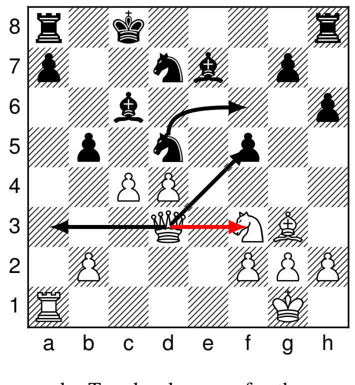

Fig. 1. A chess example. Two legal moves for the queen on $d3$ are shown. The capture to $f5$ is codified by the word $(1, 1, e)(1, 1, p)$, while move to $a3$ is encoded by $(-1, 0, e)(-1, 0, e)(-1, 0, e)$. The move to $f3$ is illegal, as in the language of queen's moves no move can end on a square containing own's piece. The $d5 - f6$ knight move is a direct jump codified by the one-letter word $(2, 1, e)$.

B. Learning by Observing

The initial purpose of introducing Simplified Boardgames was to study capabilities of learning game rules, given records of previously played games [4]. As terminal conditions were deliberately kept simple, the proper task comes to learning the languages of piece movements. This extends the directions of the GGP research, beforehand focused mainly on learning how to play a given game.

Two types of training data were proposed: a real-world *single move known* scenario – where the observer sees only the performed move, and a GGP-like *all legal moves known* scenario – where for every position all possible legal moves are visible. The latter case reflects the situation when we want to learn rules of the game given only the reasoner able to compute moves, advance game position, and detect the terminal states. Such novel, entirely simulation-based approach to general playing is used for General Video Game Playing Competition.

In [4], the algorithm for learning a DFA consistent with the observed game positions has been presented. To improve learning rate, some additional assumptions were made, allowing unseen moves to be accepted as correct if certain conditions were met. An extended approach, using slightly modified Simplified Boardgames domain, has been recently presented in [15]. The language has been modified to support basic piece addition and deletion. The process of learning was performed using the LOCM acquisition system [16], [17], which is an inductive reasoning system that learns planning domain models from action traces.

Other approaches described in the literature are mainly logic-based. In [18], chess variant rules described as first-order logic programs are learned using positive and negative examples, background knowledge, and the theory revision. The system that learns board-based games through the video demonstration using logic and descriptive complexity is presented in [19]. Alternatively, an interaction based approach to learn rules of simple boardgames from the dialogue with a human user is described in [20].

C. Regular Language Inference

Regular Language Inference is the problem of finite automata identification using labeled samples: given a disjoint sets of words S_+ containing words belonging to the target language L, and S_- containing words that do not belong to L, we ask what the size of the minimal DFA consistent with these sets is [12]. Gold ([21]) has proven that this problem is NP-hard, and Pitt and Warmuth ([22]) showed that given labeled samples corresponding to automata with n states, there is no polynomial algorithm that guarantees output DFA with at most $n^{(1-\epsilon)\log\log(n)}$ states for any $\epsilon > 0$.

The problem may be efficiently solved by polynomial algorithms if sets S_+, S_- fulfill additional restrictions, i.e. they are somehow representative. The first algorithm, known as TB/Gold was introduced by Trakhtenbrot and Barzdin in 1973 [23], and rediscovered by Gold five years later [21]. In 1992, RPNI (Regular Positive and Negative Inference) algorithm was proposed independently by Oncina and Garcia [24], and Lang [25]. It guarantees that the obtained DFA is consistent, and is equivalent to the target DFA if S_+, S_- contains the so-called *characteristic set*. Both theoretical analysis and experiments support the thesis that the results obtained by RPNI are better, and it converges faster than TB/Gold algorithm [26]. The pseudocode and detailed description of RPNI algorithm are provided in Section III-C. Several modifications of RPNI have been studied, e.g. incremental version [27], with faster convergence for some languages subfamilies [28], and suitable for PAC learning [29].

III. Boardgame Rules Inference

We are dealing with the situation where an agent observes a number of plays between some players and, by observing, it should learn the rules of this game. Assuming we restrict possible games to the class of Simplified Boardgames, where the main challenge consists of discovering the rules of piece movements, the problem can be stated as a specific variant of the Regular Language Inference.

On the one hand, this gives us a possibility to take advantage of well-known generally applicable algorithms. On the other hand, as the learning domain is narrow and characterized by certain properties, it should be possible to create more effective domain-specific learning algorithms.

A. Problem Statement and Model

Let Σ^* be the set of moves for a given Simplified Boardgame, i.e. it contains words over the alphabet of $(\Delta x, \Delta y, on)$ triplets. Given data obtained by observing movements of a piece p, there exist the partition of Σ^* into the following languages: the language S_+ of observed legal moves, the language S_- of known illegal moves, the language S_0 containing all words that are impossible to perform from any square on the board, and $S_?$ containing all the other words. So, if $w \in S_?$, then it is theoretically possible that w belongs to L_p (the language of legal movements of p), yet there is no evidence in the data that it is legal or not.

Let us consider learning scenarios proposed in [4]. In the *single move known* scenario the S_- set is always empty, and $S_?$ may be non-empty. In the *all legal moves known* scenario S_- is not empty, yet we can still have words in $S_?$.

To meet the reality of Simplified Boardgames, the aforementioned languages have to satisfy the following properties. Languages S_+, S_- and $S_?$ are finite and closed on taking substrings, i.e. for any $w \in S_+ \cup S_- \cup S_?$ every substring of w also belongs to $S_+ \cup S_- \cup S_?$. Language S_0 is infinite, and such that for all $w \in S_0$ and $a \in \Sigma$, we have $aw \in S_0 \wedge wa \in S_0$. Also, there exist a procedure $\mathcal{O} : \Sigma^* \to \{T, F\}$ which, for given w, decide in time $\Theta(|w|)$ if $w \in S_0$. (We can iterate through the word summing relative distances and checking if the board size was exceeded.)

To approximate the size of all substring-closed sets we present the following observation. We use standard $|S|$ notation to denote the cardinality of the set S, and $||S||$ to denote the sum of the lengths of words in S.

Observation 1. *Given board of size $n \times n$, and $S = S_+ \cup S_- \cup S_?$. We have that:*

$$2\sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!} \le |S| \le n^2 \sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!},$$

which estimates the total number of moves that can be made on such a board.

Notice that in the case of standard chess-like games, the number of legal moves (which is a superset of S_+) is very small compared to the left-hand side of (1). This causes that explicit occurrence search in S_- or $S_?$ is highly inefficient in comparison to checking set membership via $\mathcal{O}$ function.

For a given piece p, let $\mathcal{A}$ be a DFA approximating L_p based on given observations. It is required that $S_+ \subseteq L(\mathcal{A})$ and $L(\mathcal{A}) \cap S_- = \emptyset$. However, there is some freedom concerning relations between $L(\mathcal{A})$ and the remaining sets. Whether $L(\mathcal{A}) \cap S_0$ will be empty or not, in practice do not influence the results generated by $\mathcal{A}$. During the move generation phase, movements that are impossible to be made on the current board position are simply excluded.

The question whether $L(\mathcal{A})$ could contain some words (and which one) from $S_?$ depends on the chosen policy. It is safe to assume that every unobserved move is treated as illegal. This ensures that the player based on $\mathcal{A}$ will never produce a move that could cause e.g. an instant loss. On the other hand, admitting some words from $S_?$ allows to simplify the language definition and minimize produced DFA. Moreover, without extending the set of accepted words, it is impossible to obtain the optimal automaton given data containing only partial knowledge about L_p.

B. Björnsson's Algorithm

First, we make some observations concerning the algorithm for learning piece rules proposed by Björnsson in [4]. Because of lack of the space, we have to refer the reader to [4] for details. We describe only a general structure of this algorithm.

Two main parts of the algorithm are the automaton learning and the consistency checking. The automaton learning ([4, Algorithm 2]) begins with constructing for a given piece a *Prefix Tree Acceptor* (PTA) from the training data. The whole procedure is based on the priority queue, containing candidates for the smallest consistent DFA. The maximal number of iterations of the main while loop is limited by the *MaxExpansions* constant. In each step, the smallest DFA is taken from the queue, and it is compared against the smallest DFA obtained so far. We iterate through the pairs of states, which are then merged (collapsed) forming a new automaton. There is a constant K that prevents considering the states which are too distant from each other.

The resulting automaton is the subject of the consistency checking procedure ([4, Algorithm 1])[1]. An accepted definition

[1]In [4], the algorithm immediately returns *True* if the first of the considered positions fulfills the desired property. The line 8 of the algorithm containing "**return** $movesDFA \subseteq pos.moves(sq)$" should be rewritten to "**if** $(movesDFA \supset pos.moves(sq))$ **then return** $False$ **end if**".

of *consistency* with the training data is that DFA should generate all moves known to be legal, and no moves known to be illegal. It is checked straightforwardly in the all legal moves known scenario. To handle the single move known scenario additional assumptions have been made. An unobserved move is accepted if it was observed for some other piece or certain essential parts of this move were observed.

If the automaton passes the consistency test it is inserted into the priority queue. Since the merging procedure can return a non-deterministic automaton, an additional determinization procedure is used. The theoretical time complexity of the algorithm is exponential. More precisely, we have

Observation 2. *Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency checking the given training data td and a DFA with k states. Then, assuming that the $MaxExpansions$ and K parameters are constant, the complexity of LearnDFA(pt,td) may be bounded by*

$$O(||S_+||(2^{||S_+||} + \mathcal{C}(||S_+||, td))).$$

The part $2^{||S_+||}$ is the worst case rarely achieved in practice. The dominant part is the consistency checking, which requires browsing through all the observed positions, generating movements, and performing subset checking operations.

An important observation is that in some situations, DFA's returned by [4, Algorithm 2] generate illegal moves. For example, consider the *limited rider* piece that moves in one direction for a given limited distance. Examples of such pieces, like Short Rook or Cloud Eagle can be found in various fairy-chess games [30]. Let $\mathcal{D}$ be a training data in the *all legal moves* scenario, such that for a limited rider $\mathcal{Q}$ all its legal moves were observed (they belong to S_+), but any extension of its movements (i.e. one-step longer rides) are in $S_?$. We have the following.

Theorem 1. *If $\mathcal{Q}$ is a limited rider in a game $\mathcal{G}$, and $\mathcal{D}$ a training data described above, then the DFA returned by the [4, Algorithm 2] generates illegal moves.*

Theorem 2. *Consider* single move known *scenario and a game $\mathcal{G}$ containing a limited rider $\mathcal{Q}$ and another piece $\mathcal{R}$ with unlimited ride, i.e. satisfying $\{a_i^j, a_i^{j-1}b_i\} \subseteq L_{\mathcal{R}} \cup S_0$ for all $j > 0$. Let training data $\mathcal{D}$ be such that all the moves of the form a_i^j and $a_i^{j-1}b_i$ are observed for the piece $\mathcal{R}$. Then, the DFA returned by [4, Algorithm 2] generates illegal moves.*

(Proofs of these results are presented in [31]). The practical consequence of Theorem 2 is that given any limited rider piece occurring next to a similar not limited rider (e.g. Short Rook and Rook, Lion Dog and Queen) the learned rules of these figures can be indistinguishable.

The problem addressed in Theorems 1 and 2 lies mainly in the construction of the provided training data, which is hard to detect and handle on the algorithm's side. However, by careful designing of learning and checking procedures, we should be able to guarantee more safety, and more intuitive restrictions on the produced DFA. We will present approaches that try to fix the problem from two sides.

First, we establish dependencies between actually discovered legal moves and hypothesis concerning additional moves from $S_?$. By that, we add additional safeguard and can restrict consistency checking to discard „highly improbable" candidates even if they are consistent with the given data. Secondly, we force consistency checking function to check only those automata we strongly believe they may be good, by using a proper automata learning algorithm, based on more sophisticated heuristic strategy of merging states.

C. RPNI

An alternative solution is to use one of the existing polynomial algorithms for identification DFA's from samples, e.g. Gold [21] or RPNI [24]. Due to its better performance ([26]), we have chosen RPNI as our test algorithm for boardgame move learning.

The arguments of this algorithm are the set of positive samples S_+ and the set of negative samples S_-. Initially, a prefix tree acceptor based on S_+ is constructed. The algorithm searches for a pair of states, such that after merging these states the automaton does not accept any word from S_-. The states are chosen so that one of them is a root of a subtree of the original PTA. The merging procedure disconnects this subtree, merges the states, and then folds the subtree into the constructed DFA so that the resulting DFA remains deterministic. The complexity of the procedure is linear in the size of the folded subtree. For the detailed description, the reader is referred to [12, Section 12.4].

Given the sets of positive samples S_+ and negative samples S_-, the time complexity of RPNI is $O((||S_+|| + ||S_-||)||S_+||^2)$. However, considering the application to Simplified Boardgames and the estimation given in Observation 1, this complexity is unpractical. For this reason, we have to modify consistency checking part of the algorithm from simple iteration through S_- set to some more complex function (e.g. the one used in Björnsson's algorithm). Then we have

Observation 3. *Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. Then, the complexity of RPNI algorithm is*

$$O((||S_+|| + \mathcal{C}(||S_+||, td))||S_+||^2).$$

The algorithm remains polynomial in the size of the initial prefix tree acceptor, yet again the dominant part depends on the construction of the consistency checking procedure. For that reason, in the next section, we propose alternative procedures focused on efficiency.

IV. Efficient Consistency Checking

Our definition of consistency is that a language has to contain all positive samples (S_+) and reject the negative ones (S_-). In the case of boardgame movements learning, there are many interpretations of what the negative sample is. If we are able to observe all legal moves in any position, every unlisted move fitting within board has to be considered as negative. All the other moves that have not been rejected, can be labeled in any way.

This is the same problem (of fitting into an unknown target language) like in the standard language inference problem, yet here we know the set S_0 that does not matter at all. Also, we should be able to predict correct and incorrect moves basing on our boardgame related intuition.

A. Fast Consistency Check

Assume the scenario when our priority is to ensure our algorithm learns only correct moves, i.e. we treat $S_?$ in the same way as S_-. We are looking for the language L such that

$$S_+ \subseteq L \quad \text{and} \quad L \cap (S_- \cup S_?) = \emptyset, \tag{1}$$

and the minimal DFA representing L has the smallest number of states.

For the given prefix tree acceptor T defining S_+ and DFA $\mathcal{A}$ approximating L, the optimal procedure checking the consistency of $\mathcal{A}$ is described as Algorithm 1. Starting with the initial state of $\mathcal{A}$, and the root of T, we traverse through $\mathcal{A}$, simultaneously matching visited states with the states in T. The algorithm returns *False* if there is a mismatch in the state acceptance within the T or there is an accepting state outside T but within the board.

Algorithm 1 FastCheck($\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$, $x \in Q$, $T = \langle Q', \Sigma, \delta', q'_0, F'\rangle$, $x' \in Q'$, $w \in \Sigma^*$)

```
1:  for all $a \in \Sigma$ do
2:    if $\exists y, y'.\ \delta(x,a) = y \wedge \delta'(x',a) = y'$ then
3:      if $F(y) \neq F'(y')$ then return False end if
4:      if ¬FastCheck($\mathcal{A}$, $y$, $T$, $y'$, $wa$) then
5:        return False
6:      end if
7:    end if
8:    if $\mathcal{O}(wa)$ then continue end if
9:    if $F(\delta(x,a))$ then return False end if
10:   if ¬FastCheck($\mathcal{A}$, $\delta(x,a)$, $T$, null, $wa$) then
11:     return False
12:   end if
13: end for
14: return True
```

Let as notice, that in the case of Simplified Boardgames, complexity of the $\mathcal{O}$ function is additive, i.e. given w_1, w_2 and intermediate result of $\mathcal{O}(w_1)$, the value of $\mathcal{O}(w_1 + w_2)$ can be computed in time $O(|w_2|)$. The conclusion is that the check in line 7 can be done in $O(1)$. So the runtime of the algorithm is linear in $||L(\mathcal{A}) \cap (S_+ \cup S_?)||$, which is the upper bound for the worst case complexity from Observation 1.

B. Fractional Acceptance Consistency Check

Another reasonable assumption is that if a DFA representing a language of piece movements is small and does not contradict given data, then with a high probability it is correct. This may not be entirely true when taking into account artificially generated rules, yet in the vast majority, fairy chess pieces can be represented by automata with a simple construction.

Basing on that, and assuming that we have „big enough" training data, we can easily extend *FastCheck* algorithm to allow some fraction of moves from $S_?$. For our *FractionalCheck*$_\alpha$ algorithm we assume that if it produces language L, then at most $(1-\alpha)|L \setminus S_0|$ generated words belong to $S_?$.

The consequence of this assumption is that if L_p is the language of legal moves and S_+ observed valid moves then, given $\frac{|S_+|}{|L_p|} \geq \alpha$, the consistency check will allow optimal DFA despite acceptance of moves from $S_?$. On the other hand, if the sample is small and $\frac{|S_+|}{|L_p|} < \alpha$, the optimal automata will be rejected, as the evidence supporting its correctness are judged as too weak.

The procedure of *FractionalCheck*$_\alpha$ can be described by comparing with *FastCheck* as follows. Instead of returning *False* in case of finding an accepted move from $S_?$, the algorithm has to track the number of such moves. If their number exceeds an established threshold, then the function has to return *False*, at best finishing immediately, e.g. using the exception mechanism. In the *all legal moves* scenario, the additional consistency check with S_- is necessary. This requires browsing through all the observed positions and subset checking between observed and DFA-generated moves (which is equivalent to [4, Algorithm 1], lines 7–8).

V. Spine Compaction Algorithm

Both RPNI and Björnsson's LearnDFA algorithms are in fact general purpose methods, i.e. they do not benefit in any way from the fact that they are applied to learning boardgame piece rules. We would like to present our approach that, in contrast, is entirely based on the assumption that it has to learn rules of the chess-like piece. The goal of this algorithm is fast learning of probable piece movements by performing only specialized state merges and thus minimizing the number of required consistency checks.

The main idea uses the observation that piece PTA's usually have a form of multiple *spines* with similar subtrees attached (see Figure 2 to examine a rook-like piece example). Actual cases may be more complicated (e.g. spine's cycle period greater than one), yet the idea remains similar, also in the cases such as Checkers *man* piece.

Formally, given a (partial function based) DFA $\mathcal{A}$ and a word $w \in \Sigma^*$, we define a w-*spine* as a path in $\mathcal{A}$ starting in some state q and going along the longest prefix v of w^* that determines a valid path in $\mathcal{A}$. In such a case w is called the *vertebra* of the spine, $|w|$ its *period*, and $|v|$ its length. Each spine is encoded in the triple $(vertebra, initial\ state, length)$ (see Figure 3). We are not interested in spines where $|v| < |w|$, and discard them as not proper.

The main procedure (Algorithm 2) consists of two crucial parts. The first one is responsible for finding all spines in a given prefix tree automata. The other one analyzes the obtained spines in proper order and finds the pairs of states being candidates to merge. First, we describe both these procedures and then present a detailed description of the outline algorithm.

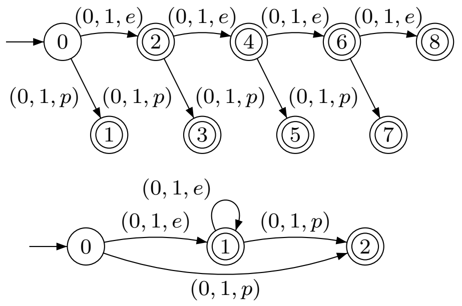

Fig. 2. An example of a chess-like piece sliding only forward. On the top the prefix tree acceptor, on the bottom the optimal DFA for this piece (assuming board of height 5).

A. Spines Selection and Pairs Selection

Given a prefix tree acceptor T, the task of the *spine selection* is to find all proper spines with a period k. Starting from the root r and traversing T using depth-first order, reaching depth k provides the first candidate w for a vertebra. However, if $F(r) \neq F(\delta(r,w))$, then we drop the first letter of w and continue searching for a proper vertebra. Otherwise, we have to analyze outgoing edges. If one is labeled by the first letter of w (say, a), and $F(\delta(r,a)) \neq F(\delta(r,wa))$ we mark $(w, r, |w|)$ as a proper spine, and start a new search from the current node. If state acceptances match, we can continue extending the current spine as long as the traversed path matches w^*. For any other edge labeled by $b \neq a$, we create w' by dropping the first letter of w and appending b at the end and proceed further down with w' as a new vertebra candidate.

The procedure works recursively for a given state considered as the actual root, candidate for vertebra, and length of the longest spine matching so far. Example of the outcome of the given procedure is shown in Figure 3. The time complexity is the same as in the case of the standard DFS, i.e. $\Theta(|Q|)$ given that we traverse only trees.

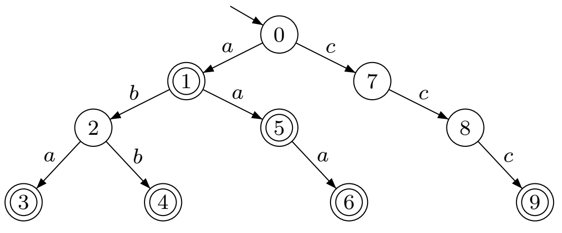

Fig. 3. For the given PTA, the spine selection procedure returns spines as $(vertebra, initial\ state, length)$ triples. Result for $k = 1$: $(a, 1, 2)$, $(c, 0, 2)$. Result for $k = 2$: $(ab, 0, 3)$, $(bb, 1, 2)$, $(aa, 1, 2)$, $(cc, 0, 2)$.

Given a w-spine, the next task is to choose a pair of states (within geodesic distance $|w|$), which will be the best candidate for merging. Our goal is to minimize a chance that the merge result will be rejected by a consistency checking procedure and simultaneously maximize the state reduction.

Let q and q' be two corresponding states in a spine (i.e. the length of the path from q to q' is a multiple of $|w|$), with outgoing vertebra edges labeled by a. In theory, it is safe to merge q and q' if the subtrees rooted in these states are equal except branches initiated with a. If the property is fulfilled for every two corresponding states in a spine, such merge does not add any new words to the language, except for the words longer then v.

In practice, it is enough to have this property only approximately fulfilled. First of all, it is too costly to check the equality of two subtrees. Instead, we check only the first level equality, i.e. if for every letter $b \neq a$, $(q,b) \in \mathrm{Dom}(\delta) \Leftrightarrow (q',b) \in \mathrm{Dom}(\delta) \wedge F(\delta(q,b)) \Leftrightarrow F(\delta(q',b))$. Moreover, we should be aware that given data might be sparse, and it is less probable that we will have valid data concerning longer moves. Given that, we apply a heuristic that allows corresponding subtrees more distance from the spine root to be smaller.

The overall *pair selection* procedure starts with a spine root r as a candidate for merging with $\delta(r,w)$. Then, we traverse through the spine comparing corresponding subtrees and replace candidate for merging if we meet a larger corresponding subtree. Final candidate, if the spine length remains longer then the vertebra, is the base for the selected pair. Figure 4 presents the visualization of the process. The complexity of the procedure is $\Theta(|v| \cdot d)$, assuming d is the maximal degree of a node in a spine.

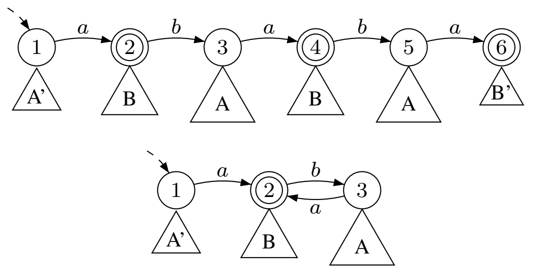

Fig. 4. Example of the spine compaction process. On the top ab-spine rooted in 1, with subtrees such that $A' \subset A$ and $B' \subset B$. On the bottom the situation after compaction: pair $(1, 3)$ is not a safe candidate for merging, so the states 2 and 4 were selected and merged instead.

B. The Main Algorithm

The main part of the Spine Compaction procedure is presented as Algorithm 2. It uses a constant K indicating, similarly as in Björnsson's algorithm, the maximal allowed length of the cycle. After constructing initial prefix tree acceptor, in lines 2–7 we search for all spines not exceeding period K, starting from the children of the root. Thus, we explicitly exclude PTA root for being selected as a spine root, to prevent it from being a part of a cycle (which is an assumption supported by our experiments). In lines 8–11, we investigate each spine to select pairs for further merging.

What remains, is to try to merge every pair and check for consistency (lines 13–18). Very important is the order of applying merge operations. Our strategy is to compact spines with shorter vertebra first, starting with the longest spines.

The merging function (line 15) is a deterministic merge used in RPNI algorithm. At this moment, we operate on states being the sets of original states, to be able to track merging process. We maintain a set of pairs that resulted in inconsistent automata (lines 12, 17), which is used to skip unnecessarily repeated computations (line 14). Additionally, we prevent pairs to create a multiloop, which is an experience-based heuristic, as fairy-chess pieces representations rarely have one. To check this condition it is enough to analyze δ values for the states in current pair. Finally, we minimize the resulting automaton.

Observation 4. *Let S_+ be the language accepted by the piece's PTA pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. We can estimate upper bound on the number of spines selected in Algorithm 2 by $O(||S_+||^2)$. Then, the complexity of the SpineCompaction algorithm is*

$$O((||S_+||^2 + \mathcal{C}(||S_+||, td))||S_+||^2).$$

Algorithm 2 SpineCompaction(Piece pt, TrainingData td)

```
1:  $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle \leftarrow constructPTA(pt, td)$
2:  $spines \leftarrow \emptyset$
3:  for all $a \in \Sigma$ if $\langle q_0, a \rangle \in \mathrm{Dom}(\delta)$ do
4:      for $k \leftarrow 1$ to $K$ do
5:          $spines \leftarrow spines \cup \mathcal{A}.SelectSpines(k, \delta(q_0, a), \varepsilon, 0)$
6:      end for
7:  end for
8:  $pairs \leftarrow \emptyset$
9:  for all $s \in spines$ do
10:     $pairs \leftarrow pairs \cup \{\mathcal{A}.SelectPair(s)\}$
11: end for
12: $forbidden \leftarrow \emptyset$
13: for all $p \in pairs.OrderByPriority()$ do
14:     if $p \in forbidden \vee isMultiloop$ then continue end if
15:     $\mathcal{A}' \leftarrow MergeAndFold(\mathcal{A}, p)$
16:     if $consistent(pt, \mathcal{A}', td)$ then $\mathcal{A} \leftarrow \mathcal{A}'$
17:     else $forbidden \leftarrow forbidden \cup \{p\}$ end if
18: end for
19: return $minimize(\mathcal{A})$
```

VI. Experiments and Empirical Evaluation

We have performed experiments to compare the three presented learning algorithms paired with different consistency check functions in both the *single move known* and *all legal moves* scenarios.

For the *all legal moves* scenario we have prepared 20 datasets per game, each containing record of 50 plays generated using two random agents playing against each other. In the case of the *single move* scenario, the number of play records in each dataset was increased to 1000. The maximum game length was set to 80 moves per player in all cases. Due to the symmetry of games, we have performed learning only for the white player (results presented in [4], [15] show no significant difference for white and black in the case of such games). If not stated otherwise, the *MaxExpansions* constant of Björnsson's algorithm was set to 20, and K used in Björnsson's and Spine Compaction algorithms was set to 2. All experiments were run on 2.60GHz Intel Core i7-6700HQ processor.

TABLE I
Experiment results for the *all legal moves* scenario.

Consistency Check	Correct size (%) Bjö.	RPNI	SC.	Errors (%) Bjö.	RPNI	SC.
Björnsson	87.6	**91.5**	87.3	**4.9**	9.1	7.6
Fractional$_{0.5}$	87.6	**89.9**	84.9	**4.9**	6.3	7.6
Fractional$_{0.6}$	89.0	**89.8**	86.6	**2.4**	4.0	5.0
Fractional$_{0.7}$	**87.1**	**87.1**	**87.1**	**2.4**	3.5	3.2
Fractional$_{0.8}$	**85.6**	85.5	**85.6**	**2.4**	3.7	3.2
Fractional$_{0.9}$	**73.7**	73.5	**73.7**	**2.4**	2.7	3.9
Fast	**69.3**	**69.3**	**69.3**	**0**	**0**	**0**

A. Games

We have used as a testbed 8 fairy chess games, containing 41 pieces altogether. The chosen games can be divided into three categories. The first one consists of simplified versions of known boardgames or their variants. It includes *chess*, *Los Alamos* (small chess variants without bishops), *Tamerlane* (large 10×11 game containing many non-standard pieces like giraffe, camel, picket), and a breakthrough variant of *Checkers*. (See [32], [4] for more detailed game descriptions.)

The other three games are the result of procedural content generation algorithms [33], [34]. Thus, they all contain only non-standard pieces. The last game was handcrafted especially to fool unaware learning algorithms. It contains two special pieces that can move horizontally like a rook without capturing moves, yet with limited movement lengths. The initial position is constructed such that every move exceeding these lengths is impossible due to the obstacles. Thus, the setup fulfills prerequisites given in the Section III-B, and there is no evidence of nonconsistency when assuming the $(1, 0, e)^* + (-1, 0, e)^*$ language for these pieces.

B. Results and Conclusions

Tables I and II show the results for the *all legal moves* and *single move known* scenarios (Spine Compaction algorithm is abbreviated as SC). For every *(learning algorithm, consistency check)* pair, we have computed the percent of generated automata with the optimal size (which is the task of regular language inference), and the percent of automata generating consistency *errors*. By an *error*, in this context, we mean a non-empty intersection with S_- of the piece's true language. These data visualize the trade-off between the rapid language expansion followed by the DFA size reduction and performing only minor adjustments to the initial prefix tree acceptor. Runtimes of all experiments are presented in Table III.

The experiments show that our *SpineCompaction* learning algorithm provides most accurate results in the *single move known* scenario of learning, and comparable although more erroneous results in the GGP-like *all legal moves known* scenario. Moreover, it requires much less time for learning than

TABLE II
EXPERIMENT RESULTS FOR THE *single move known* SCENARIO.

Consistency Check	Correct size (%)			Errors (%)		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	**73.7**	70.6	**73.7**	**5.4**	19.8	9.6
Fractional$_{0.6}$	**88.7**	61.5	87.3	6.8	40.4	**6.1**
Fractional$_{0.7}$	**89.9**	65.7	89.8	4.9	37.6	**2.8**
Fractional$_{0.8}$	**88.0**	64.6	**88.0**	4.5	34.0	**4.4**
Fractional$_{0.9}$	**73.3**	71.5	**73.3**	3.7	14.4	**3.4**
Fast	**68.5**	**68.5**	**68.5**	**0**	**0**	**0**

TABLE III
LEARNING TIMES (IN SECONDS).

Consistency Check	*all legal moves*			*single move known*		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	68.2	16.4	**4.1**	1598.6	454.1	**82.4**
Fractional$_{0.5}$	51.8	9.4	**2.2**			
Fractional$_{0.6}$	50.8	9.3	**2.1**	5.4	4.4	**0.2**
Fractional$_{0.7}$	50.8	9.2	**2.0**	5.6	3.3	**0.2**
Fractional$_{0.8}$	49.4	9.5	**1.8**	5.7	3.0	**0.2**
Fractional$_{0.9}$	42.8	10.4	**1.5**	5.7	2.6	**0.2**
Fast	4.0	4.6	**0.4**	4.6	5.3	**0.4**

the other tested algorithms. Experiments show that in practice gathered observations data are incomplete, which justifies the need for a learning function based on some approximation method. For this reason, we have also proposed consistency check functions, determining the acceptance of the proposed DFA's on the basis of the fraction of unsafe moves. These are *FractionalCheck*$_\alpha$ and *FastCheck* (which is a special case of *FractionalCheck* with $\alpha = 1$). As Tables I and II show, we can select an α value giving us better results then Björnsson's consistency check, which has been our reference point.

Nevertheless, all the tested methods are efficient in comparison to game reasoners used in GGP. This supports the thesis that the problem of General Game Playing should be solved by detecting subclasses with more effective, domain-based algorithms applicable [13]. As the boardgames are one of the most common classes of games used in GGP, the effort in this direction is relevant for the domain, and has possible practical applications in the improvement of GGP systems [4].

ACKNOWLEDGMENTS

This work was supported in part by the National Science Centre, Poland, under the project numbers 2017/25/B/ST6/01920 (Jakub Kowalski) and 2015/17/B/ST6/01893 (Andrzej Kisielewicz).

REFERENCES

[1] M. Genesereth and M. Thielscher, *General Game Playing*. Morgan & Claypool, 2014.
[2] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, pp. 62–72, 2005.
[3] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General Game Playing: Game Description Language Specification," Stanford Logic Group, Tech. Rep., 2006.
[4] Y. Björnsson, "Learning Rules of Simplified Boardgames by Observing," in *ECAI*, ser. FAIA, 2012, vol. 242, pp. 175–180.
[5] J. Kowalski and A. Kisielewicz, "Towards a Real-time Game Description Language," in *ICAART*, vol. 2, 2016, pp. 494–499.
[6] T. Schaul, "A video game description language for model-based or interactive learning," in *IEEE CIG*, 2013, pp. 1–8.
[7] M. Thielscher, "A General Game Description Language for Incomplete Information Games," in *AAAI*, 2010, pp. 994–999.
[8] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General Video Game AI: Competition, Challenges and Opportunities," in *AAAI*, 2016, pp. 4335–4337.
[9] M. Świechowski, H. Park, J. Mańdziuk, and K. Kim, "Recent Advances in General Game Playing," *The Scientific World Journal*, vol. 2015, 2015.
[10] B. Pell, "METAGAME in Symmetric Chess-Like Games," in *Heuristic Programming in Artificial Intelligence: The Third Computer Olympiad.*, 1992.
[11] J. Pitrat, "Realization of a general game-playing program," in *IFIP Congress*, 1968, pp. 1570–1574.
[12] C. de la Higuera, *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
[13] J. Kowalski and A. Kisielewicz, "Testing General Game Players Against a Simplified Boardgames Player Using Temporal-difference Learning," in *IEEE Congress on Evolutionary Computation*, 2015, pp. 1466–1473.
[14] J. Kowalski, J. Sutowicz, and M. Szykuła, "Simplified Boardgames," 2016, arXiv:1606.02645 [cs.AI].
[15] P. Gregory, Y. Björnsson, and S. Schiffel, "The GRL System: Learning Board Game Rules With Piece-Move Interactions," in *IJCAI Workshop on General Intelligence in Game-Playing Agents*, 2015, pp. 55–62.
[16] S. Cresswell and P. Gregory, "Generalised Domain Model Acquisition from Action Traces," in *International Conference on Automated Planning and Scheduling*, 2011, pp. 42–49.
[17] S. N. Cresswell, T. McCluskey, and M. M. West, "Acquisition of Object-Centred Domain Models from Planning Examples," in *International Conference on Automated Planning and Scheduling*, 2009, pp. 338–341.
[18] S. Muggleton, A. Paes, V. Santos Costa, and G. Zaverucha, "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples," in *Inductive Logic Programming*, ser. LNCS, vol. 5989, 2010, pp. 123–130.
[19] Ł. Kaiser, "Learning Games from Videos Guided by Descriptive Complexity," in *AAAI*, 2012, pp. 963–969.
[20] J. Kirk and J. E. Laird, "Interactive task learning for simple games," *Advances in Cognitive Systems*, vol. 3, pp. 11–28, 2014.
[21] E. M. Gold, "Complexity of automaton identification from given data," *Information and Control*, vol. 37, no. 3, pp. 302–320, 1978.
[22] L. Pitt and M. K. Warmuth, "The Minimum Consistent DFA Problem Cannot Be Approximated Within Any Polynomial," *Journal of the Association for Computing Machinery*, vol. 40, no. 1, pp. 95–142, 1993.
[23] B. A. Trakhtenbrot and Y. M. Barzdin, *Finite automata: Behavior and Synthesis*. American Elsevier Publishing Company, 1973.
[24] J. Oncina and P. Garcia, "Identifying Regular Languages In Polynomial Time," in *Pattern Recognition and Image Analysis*, 1992, pp. 49–61.
[25] K. Lang, "Random DFA's can be Approximately Learned from Sparse Uniform Examples," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 45–52.
[26] P. García, A. Cano, and J. Ruiz, "A Comparative Study of Two Algorithms for Automata Identification," in *Grammatical Inference: Algorithms and Applications*, ser. LNCS, 2000, vol. 1891, pp. 115–126.
[27] P. Dupont, "Incremental regular inference," in *Grammatical Interference: Learning Syntax from Sentences*, ser. LNCS, 1996, vol. 1147, pp. 222–237.
[28] A. Cano, J. Ruiz, and P. García, "Inferring Subclasses of Regular Languages Faster Using RPNI and Forbidden Configurations," in *Grammatical Inference: Algorithms and Applications*, ser. LNCS, 2002, vol. 2484, pp. 28–36.
[29] R. Parekh and V. Honavar, "Learning DFA from Simple Examples," *Machine Learning*, vol. 44, no. 1, pp. 9–35, 2001.
[30] Wikipedia. (2017, January) Fairy chess piece — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Fairy_chess_piece.
[31] J. Kowalski, "General Game Description Languages," Ph.D. dissertation, University of Wrocław, 2016.
[32] (2016) The Chess Variant Pages. http://www.chessvariants.org/.
[33] J. Kowalski and M. Szykuła, "Procedural Content Generation for GDL Descriptions of Simplified Boardgames," 2015, arXiv:1108.1494 [cs.AI].
[34] ——, "Evolving Chesslike Games Using Relative Algorithm Performance Profiles," in *Applications of Evolutionary Computation*, ser. LNCS, 2016, vol. 9597, pp. 574–589.