Regular Language Inference for Learning Rules of Simplified Boardgames

Jakub Kowalski, Andrzej Kisielewicz

University of Wrocław, Poland

CIG
15.08.2018
Simplified Boardgames
Simplified Boardgames (Y. Björnsson; 2012)

- Turn based; two players; zero-sum games;
- Rectangular board; fixed initial position; max one piece per square;
- One piece movement per turn;
- *Capturing* only at destination square;
- Winning conditions:
  - reaching a *goal* square using a certain piece,
  - captured some number of opponent’s pieces;
- *Draw* occurs when the preset maximum game length is reached.

Set of piece’s move rules

Regular language over an alphabet $\Sigma$ containing triplets ($\Delta x, \Delta y, on$):

- $\Delta x, \Delta y$ are relative column/row distances;
- $on \in \{e, p, w\}$ describes the content of the destination square:
  - $e$ – empty square,
  - $p$ – square occupied by an opponent piece,
  - $w$ – square occupied by an own piece.
- ♙d3-a3: (−1, 0, e)(−1, 0, e)(−1, 0, e)
- ♙d3-f5: (1, 1, e)(1, 1, p)
- ♖d5-f6: (2, 1, e)
- ♙d3-f3: (1, 0, e)(1, 0, w)

1 Deep Blue vs Garry Kasparov, 1997, Game 6, Move 19 (last)
Simplified Chess example

---BOARD---
8 8
rnbqkbnr
pppppppp
...........
...........
...........
PPPPPPPP
RNBQKBNR

---GOALS---
200 &
@P 0 7, 1 7, 2 7, 3 7, 4 7, 5 7, 6 7, 7 7 &
@p 0 0, 1 0, 2 0, 3 0, 4 0, 5 0, 6 0, 7 0 &
#K 1 &
#k 1 &

---PIECES---
P (0,1,e) + (-1,1,p) + (1,1,p) +
(0,1,e)(0,5,e)(0,-4,e) + ... &
N (2,1,e) + (2,-1,e) + (-2,1,e) + (-2,-1,e) + (1,2,e)...
Q (0,1,e)^[^*] + (0,1,e)^[^*](0,1,p) + (1,1,e)^[^*] + ... &
Let $\Sigma^*$ be the set of all possible words over the alphabet $\Sigma$.

For DFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, where $Q$ is the set of states, $q_0$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \rightarrow Q$ the transition function (which, in our study, is usually a partial function),

by $L(A)$ we denote the language accepted by $A$. 
Learning by Observing
Goal

Learn game rules given records of previously played matches. In Simplified Boardgames it turns to learn each piece's DFA.

Proposed scenarios

- *single move known*
  the observer sees only the performed move (real-world)

- *all legal moves*
  for every position all legal moves are listed (GGP-like)

Motivation

- Produce fast alternative to move-generation mechanism in GGP domain
Proposed algorithm

\[\textbf{Algorithm} \text{ LearnDFA(Piecetype} \ p_t, \ \text{TrainingData} \ t_d)\]

1. \(\text{dfa} \leftarrow \text{constructPTA}(p_t, t_d)\)
2. \textbf{if} not \(\text{consistent}(p_t, t_d)\) \textbf{then}
3. \hspace{1em} \text{return} \ null
4. \textbf{end if}
5. \(\text{dfa}_\text{min} \leftarrow \text{minimizeDFA}(\text{dfa})\)
6. \(n \leftarrow 0\)
7. \(Q.\text{insert} (\text{dfa}_\text{min})\)
8. \textbf{while} not \(Q.\text{empty}()\) and \(n < \text{MaxExpansions}\) \textbf{do}
9. \hspace{1em} \text{dfa} \leftarrow Q.\text{pop}()
10. \hspace{1em} \textbf{if} |\text{dfa}| < |\text{dfa}_\text{min}| \textbf{then}
11. \hspace{2em} \text{dfa}_\text{min} \leftarrow \text{dfa}
12. \hspace{1em} \textbf{end if}
13. \hspace{1em} \text{statepairs} \leftarrow \text{generalizeCandidates}(\text{dfa}, K)\)
14. \hspace{1em} \textbf{for} all \((s, s') \in \text{statepairs}\) \textbf{do}
15. \hspace{2em} \text{dfa}' \leftarrow \text{NFAtoDFA}(\text{collapse}(\text{dfa}, s, s'))
16. \hspace{2em} \textbf{if} \text{consistent}(p_t, \text{dfa}', t_d) \textbf{then}
17. \hspace{3em} \text{dfa}' \leftarrow \text{minimizeDFA} (\text{dfa}')
18. \hspace{3em} Q.\text{insert} (\text{dfa}')\)
19. \hspace{2em} \textbf{end if}
20. \hspace{1em} \textbf{end for}
21. \hspace{1em} n \leftarrow n + 1
22. \textbf{end while}
23. \textbf{return} \ \text{dfa}_\text{min}
Consistency checking

Consistent DFA should generate all moves known to be legal and no moves known to be illegal.

Algorithm consistent(Piecetype \( pt \), DFA \( dfa \), TrainingData \( td \))

1: for all \( \{ \text{pos} \in td \} \) do
2:     for all \( \{ \text{sq} \in \text{pos.board} \mid \text{pieceType}(\text{sq}) = pt \} \) do
3:         \( \text{movesDFA} \leftarrow \text{generateMoves}(\text{dfa}, \text{pos}, \text{sq}) \)
4:         if \( \text{pos.moves}(\text{sq}) \nsubseteq \text{movesDFA} \) then
5:             return False
6:         end if
7:     if \( \text{pos.moveListing} = \text{all} \) then
8:         if \( \text{movesDFA} \supseteq \text{pos.moves}(\text{sq}) \) then return False end if
9:     else \{ \text{pos.moveListing} = \text{some} \} \}
10: for all \( \{ \text{pmp} \in \text{movesDFA} \setminus \text{U} \} \) do
11:     if \( \text{\#}(\text{pmp}) \nsubseteq \text{U} \) then
12:         return False
13:     end if
14: end for
15: end if
16: end for
17: end for
18: return True
Regular Language Inference
Regular Language Inference

**Definition**

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 is the size of the minimal DFA consistent with these sets.

**Algorithms**

- Proven to be NP-Hard (Gold, 1978)
- TB/Gold (Trakhtenbrot and Barzdin, 1973 / Gold, 1978)
- RPNI – Regular Positive and Negative Inference (Oncina and Garcia, 1992 / Lang, 1992)
- RPNI guarantees that the obtained DFA is consistent, and is equivalent to the target DFA if some special conditions are met for $S_+, S_-$. 
Algorithm RPNI($S_+ \subseteq \Sigma^*, S_- \subseteq \Sigma^*$)

1: $A = \langle Q, \Sigma, \delta, q_0, F \rangle \leftarrow PTA(S_+)$
2: $blue \leftarrow \{\delta(q_0, a) : a \in \Sigma\}$
3: $red \leftarrow \{q_0\}$
4: while $blue \neq \emptyset$ do
5:     choose $q \in blue$
6:     if $\exists p \in red. L(MergeAndFold(A, p, q)) \cap S_- = \emptyset$ then
7:         $A \leftarrow MergeAndFold(A, p, q)$
8:     else
9:         $red \leftarrow red \cup \{q\}$
10:    end if
11: $blue \leftarrow \{\delta(q, a) : q \in red, a \in \Sigma\} \setminus red$
12: end while
13: return $A$
Boardgame Rules Inference
Problem Statement and Model

Moves partitioning

Given data obtained by observing movements of a piece \( p \), there exist the partition of \( \Sigma^* \) (alphabet of all possible moves) 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.

- 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 \( S_? \).
Consider moves of the white knight: \((-1, 2, e) \in S_+\) because it is listed as one of the legal moves; \((-1, 2, e)(-1, 2, p) \in S_-\) because this move can be performed on current position (capturing black knight) but is not listed as a legal one; \((-1, 2, e)^2 \in S_\emptyset\) because in given position we cannot decide its correctness; \((-1, 2, e)^4 \in S_0\) because move length exceeds the board size.

```
 r.k....r
 p..nb.p.
 ..b....p
 .p.n.p..
 ..PP....
 ...Q.NB.
 .P...PPP
 R.....K.
 43
 P  1  1 (0,1,e)
 P  1  1 (0,1,e)(0,5,e)(0,-4,e)
 N  5  2 (2,1,e)
 N  5  2 (-2,-1,e)
 N  5  2 (1,2,e)
 N  5  2 (-1,2,e)
 N  5  2 (-1,-2,e)
 Q  3  2 (1,1,e)
 Q  3  2 (1,1,e)(1,1,p)
 ...```
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 $aw \in S_0 \land wa \in S_0$.

There exist a procedure $O : \Sigma^* \to \{T, F\}$ 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.)

**Observation**

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

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

which estimates the total number of moves that can be made on such a board.
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 $L(\mathcal{A}) \cap S_- = \emptyset$.
- Whether $L(\mathcal{A}) \cap S_0$ will be empty or not, in practice do not influence the results generated by $\mathcal{A}$.
- $L(\mathcal{A})$ could contain some words from $S_-$:
  - allows to simplify the language definition and minimize produced DFA,
  - unsafe if e.g., making illegal move means instant loss
Consider the *limited rider* piece, i.e., a piece that moves in one direction for a given limited distance (e.g., Short Rook, Cloud Eagle).

**Theorem**

Let $\mathcal{D}$ be a training data in the all legal moves scenario, such that for a limited rider $Q$ all its legal moves were observed, but any extension of its movements (i.e. one-step longer rides) are in $\mathcal{S}$. If $Q$ is a limited rider in a game $\mathcal{G}$, and a training data $\mathcal{D}$, then the DFA returned by the Björnsson's Algorithm generates illegal moves.

**Theorem**

Consider single move known scenario and a game $\mathcal{G}$ containing a limited rider $Q$ and another piece $R$ with unlimited ride, i.e. satisfying $\{a_i^j, a_i^{j-1}b_i\} \subseteq L_R \cup S_0$ 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 $R$. Then, the DFA returned by the Björnsson's Algorithm generates illegal moves.
Existing algorithms complexity analysis

Observation

Let $S_+$ be the language accepted by the piece's prefix tree acceptor $pt$, and $C(k, td)$ 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 $\text{LearnDFA}(pt, td)$ may be bounded by

$$O(||S_+||(2||S_+|| + C(||S_+||, td))).$$

Observation

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

$$O((||S_+|| + C(||S_+||, td)||S_+||^2).$$
Efficient Consistency Checking
Fast consistency check

Assumption

\[ S_+ \subseteq L \text{ and } L \cap (S_- \cup S_?) = \emptyset, \]  

(1)

Algorithm

\begin{algorithm}
\begin{algorithmic}[1]
\Function{FastCheck}{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^* )}
\ForAll{a \in \Sigma}
\If{\exists y, y'. \delta(x, a) = y \land \delta'(x', a) = y'}
\If{F(y) \neq F'(y')}
\Return{\text{False}}
\EndIf
\EndIf
\EndFor
\If{\Ø(wa)}
\Continue
\EndIf
\If{F(\delta(x, a))}
\Return{\text{False}}
\EndIf
\If{\neg\text{FastCheck}(A, \delta(x, a), T, null, wa)}
\Return{\text{False}}
\EndIf
\Return{True}
\EndFunction
\end{algorithmic}
\end{algorithm}

Complexity

Linear in \(|L(A) \cap (S_+ \cup S_?)|.|
**Fractional Acceptance Consistency Check**

**Assumption**

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.

**FractionalCheck\(_\alpha\)**

If it produces language \( L \), then at most \((1 - \alpha)|L \setminus S_0|\) generated words belong to \( S_? \).

**Description**

- Instead of returning *False* in *FastCheck* in case of finding an accepted move from \( S_? \), the algorithm tracks the number of such moves.
- Function returns *False* if their number exceeds the threshold.
- In the *all legal moves* scenario, browses through all the observed positions and checks subsets between observed and DFA-generated moves.
Spine Compaction Algorithm
RPNI and Björnsson’s Algorithm are general purpose methods.
Let’s do something more domain-specific.

Typical case

Goal
Fast learning of probable piece movements
performing only specialized state merges
minimize the number of required consistency checks
Successfully deal with not-so-typical cases
Spines selection

- PTA’s usually have a form of multiple *spines* with similar subtrees attached.
- We will search for a *vertebra* words, i.e., the words that can be used as a period of some spines in PTA.
- Vertebra are proper if the corresponding states it points out have the same acceptance.
- States along the spine (distant of vertebra length) will be the promising candidates for a merge.

Main procedure

1. Find all spines in given PTA (iterating for possible periods $k$)
2. Analyze spines in proper order and candidates for merge (pairs if states)
Spines selection

Example

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

Time complexity

Assuming that we traverse only trees, the time complexity of spine selection is \(\Theta(|Q|)\).
Spine compaction

- If we have two corresponding states in a spine, we can safely merge them iff. subtrees rooted in them are the same (except vertebra outgoing edge).
- This is too costly to check, so instead we only require first level equality.
- Moreover, we allow subtrees further from the root to be smaller (assuming sparse data).

Sketch

- *Pair selection* procedure starts with a spine root $r$ as a candidate for merging with $\delta(r, w)$.
- We traverse through the spine comparing corresponding subtrees and replace candidate for merging if we meet a larger corresponding subtree.
- The complexity of the procedure is $\Theta(|v| \cdot d)$, assuming $d$ is the maximal degree of a node in a spine.
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.
Main algorithm

- We use constant indicating maximal allowed length of cycle
- We search for all spines in the initial PTA (excluding the root)
- Pairs selection procedure is filtering spines to find the proper candidates to merge
- We try to merge every pair and check consistency – starting with shorter vertebra first and the longest spines
- Pairs of states are merged using the same deterministic merge procedure as in RPNI
- Sets of pairs resulting in inconsistent automata are stored to skip repetitions.
- As a experience-based heuristic we prevent creating multiloops.
Observation

Let $S_+$ be the language accepted by the piece’s PTA $pt$, and $C(k, td)$ 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 Spine Compaction algorithm by $O(||S_+||^2)$. Then, the complexity of the Spine Compaction procedure is

$$O((||S_+||^2 + C(||S_+||, td))||S_+||^2).$$
Experiments and Empirical Evaluation
Experiments

Games

We used 8 fairy chess games, containing altogether 41 pieces
- chess, Los Alamos, Tamerlane, Breakthrough Checkers
- 3 PCG games (including Legacy of Ibis)
- Special game containing limited rider and with a setup forcing the requirements for the theorems.

Test data

- based on plays between random agents
- learning performed for the first player only
- turnlimit set to 80
- all legal moves scenario:
  20 datasets per game, each containing 50 games
- single move scenario:
  20 datasets per game, each containing 1000 games
Results for the *all legal moves* scenario

<table>
<thead>
<tr>
<th>Consistency Check</th>
<th>Correct size (%)</th>
<th></th>
<th>Errors (%)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bjö.</td>
<td>RPNI</td>
<td>SC.</td>
<td>Bjö.</td>
</tr>
<tr>
<td><em>Björnsson</em></td>
<td>87.6</td>
<td>91.5</td>
<td>87.3</td>
<td>4.9</td>
</tr>
<tr>
<td><em>Fractional</em>&lt;sub&gt;0.5&lt;/sub&gt;</td>
<td>87.6</td>
<td>89.9</td>
<td>84.9</td>
<td>4.9</td>
</tr>
<tr>
<td><em>Fractional</em>&lt;sub&gt;0.6&lt;/sub&gt;</td>
<td>89.0</td>
<td>89.8</td>
<td>86.6</td>
<td>2.4</td>
</tr>
<tr>
<td><em>Fractional</em>&lt;sub&gt;0.7&lt;/sub&gt;</td>
<td>87.1</td>
<td>87.1</td>
<td>87.1</td>
<td>2.4</td>
</tr>
<tr>
<td><em>Fractional</em>&lt;sub&gt;0.8&lt;/sub&gt;</td>
<td>85.6</td>
<td>85.5</td>
<td>85.6</td>
<td>2.4</td>
</tr>
<tr>
<td><em>Fractional</em>&lt;sub&gt;0.9&lt;/sub&gt;</td>
<td>73.7</td>
<td>73.5</td>
<td>73.7</td>
<td>2.4</td>
</tr>
<tr>
<td><em>Fast</em></td>
<td>69.3</td>
<td>69.3</td>
<td>69.3</td>
<td>0</td>
</tr>
</tbody>
</table>

**Correct size definition**

Generated automata has a *correct size* if it has the same size as the optimal piece’s recognizing DFA.
Results for the *single move known* scenario

<table>
<thead>
<tr>
<th>Consistency Check</th>
<th>Correct size (%)</th>
<th>Errors (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bjö.</td>
<td>RPN</td>
</tr>
<tr>
<td><strong>Björnsson</strong></td>
<td>73.7</td>
<td>70.6</td>
</tr>
<tr>
<td><strong>Fractional</strong>&lt;sub&gt;0.6&lt;/sub&gt;</td>
<td>88.7</td>
<td>61.5</td>
</tr>
<tr>
<td><strong>Fractional</strong>&lt;sub&gt;0.7&lt;/sub&gt;</td>
<td>89.9</td>
<td>65.7</td>
</tr>
<tr>
<td><strong>Fractional</strong>&lt;sub&gt;0.8&lt;/sub&gt;</td>
<td>88.0</td>
<td>64.6</td>
</tr>
<tr>
<td><strong>Fractional</strong>&lt;sub&gt;0.9&lt;/sub&gt;</td>
<td>73.3</td>
<td>71.5</td>
</tr>
<tr>
<td><strong>Fast</strong></td>
<td>68.5</td>
<td>68.5</td>
</tr>
</tbody>
</table>

**Error definition**

By an *error*, we mean a non-empty intersection with $S_-$ of the piece’s true language.

---

Jakub Kowalski, Andrzej Kisielewicz

Regular Language Inference for Learning Rules of Simplified Boardgames
### Learning times (in seconds)

<table>
<thead>
<tr>
<th>Consistency Check</th>
<th>all legal moves</th>
<th>single move known</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bjö.</td>
<td>RPNI</td>
</tr>
<tr>
<td>Björnsson</td>
<td>68.2</td>
<td>16.4</td>
</tr>
<tr>
<td>Fractional(_{0.5})</td>
<td>51.8</td>
<td>9.4</td>
</tr>
<tr>
<td>Fractional(_{0.6})</td>
<td>50.8</td>
<td>9.3</td>
</tr>
<tr>
<td>Fractional(_{0.7})</td>
<td>50.8</td>
<td>9.2</td>
</tr>
<tr>
<td>Fractional(_{0.8})</td>
<td>49.4</td>
<td>9.5</td>
</tr>
<tr>
<td>Fractional(_{0.9})</td>
<td>42.8</td>
<td>10.4</td>
</tr>
<tr>
<td>Fast</td>
<td>4.0</td>
<td>4.6</td>
</tr>
</tbody>
</table>
Conclusions

**SpineCompaction** learning algorithm

- provides most accurate results in the *single move known* scenario,
- comparable although more erroneous results in the GGP-like *all legal moves known* scenario,
- requires significantly much less time for learning.

Experiments show that in practice gathered observations data are incomplete, which justifies the need for a learning function based on some approximation method.

**FractionalCheck**$_\alpha$ and **FastCheck** consistency checking methods

- we can select an $\alpha$ value giving better results than the reference algorithm.
- they are also much faster.
Thank you