Automating Games Proofs

Mehdi Mhalla
Mehdi.Mhalla@imag.fr
CNRS Université de Grenoble - LIG, B.P. 53 - 38041 Grenoble Cedex 09, France

Frédéric Prost
Frederic.Prost@imag.fr
Université de Grenoble - LIG, B.P. 53 - 38041 Grenoble Cedex 09, France

November 20, 2013

Abstract

1 Introduction

2 Formalization

In this section we precisely define our vocabulary and formalize some properties of board games. We think about chess, but our formalization can also be used, mutatis mutandis, for similar board games i.e. smaller chess variants (Gardner's Chess **??**, Los Alamos Chess, etc.), Shogi [], Xiangqi [], Go [].

- A **position** is the set of information that are sufficient to generate the **legal moves** allowed by the rules of the game at this point. For chess it consists in the knowledge of the piece placement, the side to move, the castling rights and the number of the column in which an en passant move is possible.
- A **game graph** is a graph in which vertices are positions and edges are legal moves transforming source positions into target positions. In the following we do not distinguish between a position and a node representing it, similarly we do not distinguish betwee a move and its representing edge. Nodes without outgoing edges are called **leafs** and have a **status** denoting their game-theoretic value. For chess possible status are **D**, **B**, **W**, which respectively mean that the game is a draw, a win for Black, a win for White. Vertices that are not leafs are called **nodes**. The colour of a vertex is the colour of the side to move in the position represented by the vertex.
- A vertex of a game graph is **saturated** there is an edge from this vertez for every possible legal move.
- The game graph **generated** by a position is the game graph obtained by freely saturating each vertex of the graph starting from the given position.
- A **White oracle** (resp. **Black oracle**) is a game graph in which each White (resp. Black) vertex is either a leaf or a node of degree 1, and each Black (resp. White) vertex is either a leaf or is saturated.
- A node of a White oracle (resp. Black oracle) is said to be a **friend node** if White (resp. Black) is to move, if it is Black (resp. White) to move the node is said to be a **foe node**.

We note $\mathcal{C}$ the game graph generated by the initial position of chess. All valid chess games are paths in $\mathcal{C}$ whose first vertex is the initial position. Note that our definitions do not take into account the threefold repetition rule, the draw by insufficent material, and the 50 move without capture or pawn moves rule. $\mathcal{C}$ is a graph and not a tree since a given position can be reached via different play (transpositions) and the same position may occur several times which translates into cycles.

The aim of our definitions is to be able to give a formal proof of the game-theoretical value of a position.

- A position is won for White (resp. Black) if there exists a White (resp. Black) oracle **O** such that each leaf of the oracle has the status **W**(resp **B**).
- A position is a draw if there are White and Black oracles from this position such that all the leafs of the White (resp. Black) oracle have a status that is either **W**(resp. **B**) or **D**.

One advantage of our definition is that one doesn't have to handle threefold repetitions or 50 move rules explicitely. Indeed, a White oracle for a draw always provides a White move for every reachable position from the starting position. Threefold repetitions and 50 moves rule are dealt with cycles in the graph.

An intuitive way to understand why such an oracle provides a draw is the fact that, by definition, the oracle contains all positions reachable from a given position. Moreover for each of these positions it gives a playable move. Since losing amounts to not being able to make a move, the oracle never loses.

Another point is that this definition can be tuned to any property. Instead of the theoretical value of a position any suitable logical formula could be chosen and applied to leafs. Consider for instance the formula: White can force checkmate with Black having only one Knight left. It is provable via a White oracle (without cycles) in which all leafs are positions in Which Black is checkmated with only one Knigth left. It is especially usefull from a technical point of view. Indeed when computing an oracle it is useful to stop the analysis beyond a given engine evaluation threshold, or when the number of pieces is such that the result can be determined by tablebases.

3 Practical data-structures for Oracles

In section **??** we have given a formal definition of oracles. However, from a practical point this definition contains too much redundancy and is not easy to handle.

We can distinguish three interesting different data structures related to a given oracle. Those data structures are determined by the intended use of the oracle.

1. Building the oracle: an oracle is built incrementally starting from an initial position and generating oracle moves for friend nodes and by saturating foe nodes by all legal moves. We call this data-structure an **oracle builder**, or builder for short.
2. Storing the oracle: in order to store the oracle we do not have to store the position at every node. Indeed the position of a node can be computed from the moves on the path from the initial position to the current position. Moreover, oracles have a very specific structure: foe nodes are saturated and friend nodes only have one descendant. Finally the number of legal moves is finite and can be totally ordered. Thus it is possible to completely store the oracle by storing the initial position and a spanning tree of the game graph by simply indicating the number of the move that has to be played for the friend side (since every foe moves have to be considered). We call this data-structure an **oracle reminder**, or reminder for short.
3. Playing along oracle lines: in order to use an oracle one only needs a huge table which tells him what to play depending on the position. It can be generated by compiling all positions, and oracle moves, of the friend nodes of the oracle. We call this structure an **oracle player**, or player for short.

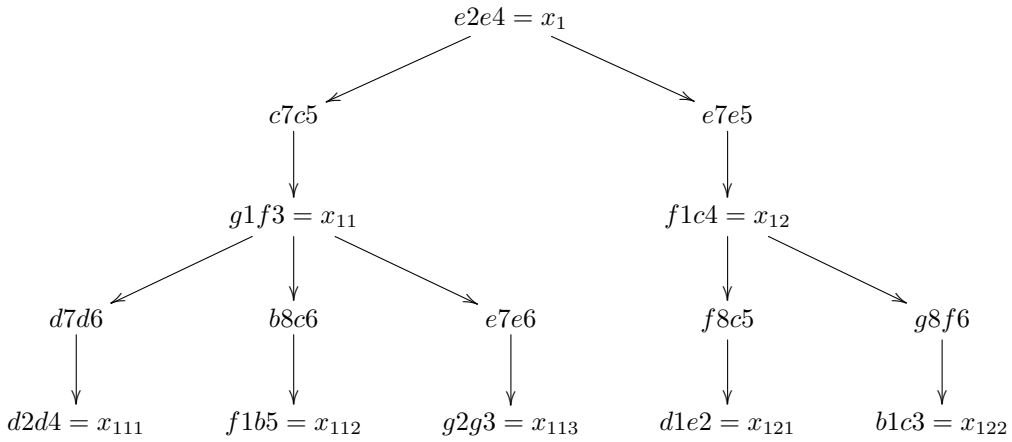
3.1 Oracle builder

An oracle is built incrementally, we discuss an algorithm for it in section **??**, here we just consider the data-structure. The general idea of oracle building is to start from a position and to incrementally compute oracle moves for all possible answer of the opponent. The fact that it is incremental implies that there exists leafs (nodes without descendance) which doesn't have a clear game-theoretic status.

3.2 Oracle reminder

The aim of the reminder data-structure is to be the most concise possible while still containing enough information to compute oracle builder and oracle player from it. It relies heavily on the fact that oracles have a particular shape and that legal moves can be enumerated and sorted. The rough idea is to consider a spanning tree of the oracle starting from the initial position and only recording friend moves. Leafs are anotated with null moves (move number 0) and the degree of a node is completely determined by the number of legal moves.

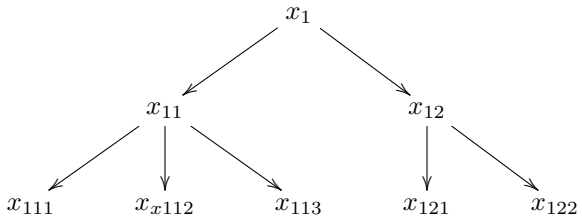
Suppose for instance that we have a White oracle for draw from the initial position of chess that looks like this:



The content of friend nodes is made by a move and the number of this move among legal moves. For instance $e2e4 = x_1$ means that **1 e4** is the x_1-th move from the initial position with relation to the ordering chosen on legal moves.

For the sake of simplicity we suppose that there are only two legal Black moves after White's first move.

Actually this tree contains redundant information in the sense that since it is a White oracle then all Black's move have to be considered. Those moves can be automatically generated and don't need to be recorded. Therefore it coul'd be represented in a more comapct way with the following tree:



The prefix list of this tree can be written as the simple following string "$x_1x_{12}x^*_{111}x^*_{112}x^*_{113}x_{13}x^*_{131}x^*_{132}$" where the $*$ annotation denotes leafs. Since the number of legal moves can be computed, the original tree can be rebuilt only using this information only. Indeed the number of childs for each node is equal to the number of legal moves of the position obtained by playing all moves from the root to a node.

From a practical point of view, if we make the plausible assumption that there are no positions in the oracle with more than 128 legal moves one single byte is enough to encode two nodes of the oracle. The first bit of the byte indicates whether the node is a leaf or not and the 7 remaining bits are used to give the number of the legal move chosen.

3.3 Oracle player

The aim of the oracle player is to provide a data-structure allowing to play perfectly, i.e. providing a move that does not change the theoretical value of the position. One only has to record the set of all reachable positions of the oracle for the friend side and to give the associated move to tackle this problem.

This table is simply built from the original position and the remainder. Starting from the initial one has to play the oracle move and then compute and play all legal moves from this position. For all this positions we associate the oracle move and go on like this towards the leafs. Transpositions are automatically dealt since the remainder is a spanning tree of the oracle.

4 Oraclefinder: an algorithm for chess oracle generation

In this section we give an algorithm, Oraclefinder, to compute an oracle in order to prove that the theoretical value of a position is a draw or a win. Basically, to prove that a position is a draw we need to build a draw oracle for the side to move and as many draw oracles as there are legal moves from the initial position for the opponent. Typically to prove that the game of chess is draw requires a White oracle for draw from the starting position and twenty Black oracles for draw for each of the first twenty legal moves. To prove that a position is a win requires only one oracle having all its leaves with a win status.

Contrary to usual chess exploration engines, the main point of Oraclefinder is not to find the best move, from a chess point of view, but to find the move that will generate the smallest oracle. It is a meta-algorithm that uses chess engines only to find a list of interesting moves (basically an interesting move is a move that do not change the theoretical value of a position).

For instance, a proof certificate of the draw for a position where there are only two Kings (suppose that we do not consider this situation as an immediate draw because of the material insufficiency) on the board could theoretically have 7,800 nodes. Indeed, there are 3900 legal positions to multiply by 2 because of the side to move. But this number can be droped to 216 if the side trying to draw decides to move his King back and forth on two squares only. Once every position have been explored the Oracle is complete, indeed any move will generate a position already in the oracle.

5 Results

6 conclusion