Bark: An Attributed Graph Grammar Chess Engine

Davide Marincione

May 25, 2024

1 Introduction

Chess is a fairly complex game, with a relatively wide set of rules and a small but non-trivial branching factor (unlike Go, which has very simple rules but huge branching factor). This makes it a good candidate for projects such as Bark. Unlike traditional chess engines, Bark is not programmed in any language: it is instead a set of rules that can be applied to a graph, which is then used to play chess. This makes it a very complex and unwieldy project, as it requires a lot of work to define the rules and the graph structure. In this report, I will describe the rules and the graph structure used in Bark, how their mechanisms differ from a usual chess engine, and I will discuss the challenges and limitations of the project.

A small disclaimer Since this report is dealing with attributed graph grammars, it is full of words like "node" and "edge"; to avoid repetition, I will use the formatting **node** and ***edge*** to decrease the verbosity.

2 Representing the board

Chess cannot be played without a chessboard, therefore the first step in creating Bark was to define a graph structure that could represent the board.

How it is done Fortuitously, a chessboard is composed of 64 squares and, because of that, it can be conveniently be represented as a set of 64-bit integers, where each bit gives the value of a square with respect to specific information of the board. Because of this, a chessboard is usually represented with twelve 64-bit integers, one for each piece type and color. Not only is this representation very compact, but it also allows for very fast operations, as bitwise "magic bitboard" operations are at the core of every modern chess engine.

How Bark does it Bark, however, does not use this representation. Instead, the board is composed of 64 **square** and 46 **slide_group** nodes. Each **square** is connected to its neighbors via a ***adj*** and to its **slide_group**s via a ***part_of***. Neighbors are defined only to be those squares with a common edge, and **slide_group**s are used to represent the groups of squares reachable in a single move by sliding pieces (bishop, rook or queen). Because of this, **slide_group**is actually an abstract class from which **bishop_like_group** and **rook_like_group** inherit.