Mathematical Theory of Chess and Fairy Chess Games

Mohammed-Hounaïne EL-Hamiani-Khatat

June 2025

Independent Researcher
hounaine.hamiani@batencore.com
https://github.com/hounaine/baten_chess

Introduction

Chess, whether in its classical form or in countless fairy variants, shares a common mathematical foundation: the regulated movement of pieces on a finite, discrete board. Despite the diversity of rules and the proliferation of new pieces, no single framework has yet unified the following elements:

1. Modeling the board as a directed graph $G = (V, E)$ on an arbitrary $n \times p$ grid. 2. Defining each piece type p by a finite displacement set $\Delta_p \subset \mathbb{Z}^2$ and a path-clearance predicate. 3. Capturing advanced game conditions (check, checkmate, promotion, castling, en passant, pin) as modular logical guards. 4. Providing a declarative mini-DSL that, via simple YAML specifications, describes both Δ_p and the guard predicates for each piece and automatically generates a complete move validator.

We also explicitly note that any fairy-chess piece can be reduced to a pair $(\Delta_p, \mathrm{guards}_p)$, where $\Delta_p \subset \mathbb{Z}^2$ encodes its kinematics and guards_p is a set of logical predicates encoding its special rules.

This treatise delivers such a theory, combining rigorous formalism with practical extensibility.

A subsequent **Part II** (currently in preparation) will develop the theory of *dynamic control predicates*, *boundary-adapted directionality*, and *occupancy constraints* for both classical and fairy variants, thus completing our universal framework.

1 Part I: Modeling the Chessboard as a Graph

1.1 Vertices and Coordinates

Let $n \geq 1$. We define

$$V = \{(x, y) : x, y \in \{0, 1, \ldots, n-1\}\}.$$

Each square is labeled by (x, y) with $0 \leq x, y < n$. Hence $|V| = n^2$.

1.2 Displacement Sets Δ_p

Each piece p is associated with a finite set

$$\Delta_p = \{(dx_i, dy_i) \in \mathbb{Z}^2 : i = 1, \ldots, k_p\}.$$

Table 1 lists the classical Δ_p on an $n \times n$ board:

To define a *fairy piece*, simply choose any finite subset $\Delta_p \subset \mathbb{Z}^2$. For example, a "Dragon" can be

$$\Delta_{Dragon} = \Delta_{bishop} \cup \Delta_{knight} \cup \{(0, 1), (0, -1)\}.$$

Piece p	Δ_p (examples for 8×8; analog for any n)
Pawn (white)	$\{(0,1)\} \cup \{(\pm 1, 1)\}$ plus $(0,2)$ if on rank 1
Pawn (black)	$\{(0,-1)\} \cup \{(\pm 1, -1)\}$ plus $(0,-2)$ if on rank 6
Knight	$\{(\pm 1, \pm 2), (\pm 2, \pm 1)\}$
Bishop	$\{(\pm d, \pm d) : d = 1, \ldots, n-1\}$
Rook	$\{(d,0), (-d,0), (0,d), (0,-d) : d = 1, \ldots, n-1\}$
Queen	$\Delta_{queen} = \Delta_{bishop} \cup \Delta_{rook}$
King	$\{(\pm 1, 0), (0, \pm 1), (\pm 1, \pm 1)\}$

Table 1: Classical displacement sets Δ_p. For sliding pieces, $d = 1, \ldots, n-1$.

1.3 Definition of E_p and the Graph G_p

For each piece p, define the directed edge set

$$E_p = \{(u, v) \in V \times V : v - u \in \Delta_p\}.$$

Hence $G_p = (V, E_p)$ is the "movement graph" of piece p, ignoring any blocking.

1.4 Occupation Function

At any given board state, we maintain

$$\mathrm{Occ} : V \to \{\texttt{Empty}, \texttt{Friendly}, \texttt{Enemy}\}.$$

It indicates whether each square is empty, occupied by a friendly piece, or by an enemy.

2 Part II: Pure Kinematics of Piece Movements

We now define a helper and give pseudocode to test the legality of a single move $(u \to v)$ for piece p.

2.1 Straight-Line Path

Definition 1 (Straight-Line Path)**.** Given $u = (x_u, y_u)$ and $v = (x_v, y_v)$ such that $v - u \in \Delta_p$ and p is sliding, let $d = \|v - u\|_\infty$ and

$$P(u, v) = \{(x_u + t\,dx,\ y_u + t\,dy) : t = 1, \ldots, d-1\},$$

where $(dx, dy) = (v - u)/d$ is the unit direction.

2.2 Move Validation Pseudocode

```
function isLegal(p, u, v, Occ):
    if Occ[v] == Friendly: return False
    delta = (v.x-u.x, v.y-u.y)
    if delta not in Delta[p]: return False
    if p in {Knight, King}: return True
    if p==Pawn:
        dx, dy = delta
        if dy==1 and dx==0 and Occ[v]==Empty: return True
        if dy==2 and dx==0 and onStart(u) and Occ[v]==Empty and Occ[(u.x,u.y+1)]==Empty: retu
        if abs(dx)==1 and dy==1 and Occ[v]==Enemy: return True
        return False
```

```
stepX, stepY = sign(delta.x), sign(delta.y)
d = max(abs(delta.x), abs(delta.y))
for t in range(1,d):
    w = (u.x+t*stepX, u.y+t*stepY)
    if Occ[w] != Empty: return False
return True
```

3 Part III: Unification of Advanced Guard Rules

Let B be a finite $n \times p$ chessboard, modeled as the directed graph

$$G = (V, E), \quad V = \{(x, y) \mid 1 \le x \le n,\ 1 \le y \le p\}, \quad E = \bigcup_{p \in \mathcal{P}} E_p,$$

where for each piece type p,

$$E_p = \{\, (u, v) \in V \times V \mid v - u \in \Delta_p \text{ and } \text{path\_clear}(u, v)\},$$

and $\Delta_p \subset \mathbb{Z}^2$ is the set of elementary displacement vectors for p.

A candidate move $(u \to v)$ must first satisfy this *pure kinematic* test. We then apply the following *guards*—each a predicate—that filter out illegal moves:

1. Color Guard

$$\text{ColorGuard}(p, u, v, b) \ \equiv\ \text{color}(p) = \text{turn}(b).$$

2. Pin Guard

Let b' be the board after symbolically moving piece p from u to v, and let k be the king's square of color(p). Then

$$\text{PinGuard}(p, u, v, b) \ \equiv\ \neg\big(\exists\, w : (w \to k) \in E_{p'}\big).$$

3. Check Guard

$$\text{CheckGuard}(p, u, v, b) \ \equiv\ \neg\big(\text{is\_in\_check}(b', \text{color}(p))\big).$$

4. Castling Guard

Applicable only if $\Delta_{\text{King}} \ni (\pm 2, 0)$. Let flag be the appropriate castling–right indicator and $d = (v - u)/2$. Then

$$\text{CastlingGuard}(u, v, b) \ \equiv\ (v - u) \in \Delta_{\text{King}} \wedge \text{castling\_rights[flag]} = \text{True} \wedge \forall\, w \in \{u + s \cdot d \mid s = 1, 2\} :\ (w \notin b.\text{pie}$$

5. Promotion Guard

For pawns P:

$$\text{PromotionGuard}(p, u, v, b) \ \equiv\ p = \text{P} \ \wedge\ (v_y = 1 \text{ or } v_y = p).$$

6. En Passant Guard

Let ep be the stored en passant target and last_move $= (u', v')$. Then

$$\text{EnPassantGuard}(p, u, v, b) \ \equiv\ p = \text{P} \ \wedge\ v = \text{ep} \ \wedge\ v' - u' = 2\Delta_{\text{P}}.$$

A move $(u \to v)$ is *legal* if and only if it passes all guards:

$$\text{is\_legal}(p, u, v, b) \ \equiv\ (u \to v) \in E_p \ \wedge \bigwedge_{G \in \{\text{Color,Pin,Check,Castling,Promotion,EP}\}} G(p, u, v, b).$$

Theorems

Soundness. Every move accepted by this pipeline is legal under the official rules.

Completeness. Every legal move is accepted by the pipeline, provided Δ_p and guards are correctly defined.

4 Part IV: Mini-DSL for Move Validator Generation

We now present a formal specification language (DSL) that declares both kinematic vectors Δ_p and guard predicates for any piece p. This DSL uses a YAML-based syntax and is parameterized by arbitrary board dimensions $n \times p$. The generator script `generate_dsl.py` processes these specifications and produces a complete Python move validator.

4.1 DSL Grammar (EBNF)

The grammar for a single piece specification is:

```
piece_spec   ::= "piece:" WS PIECE_NAME '\n'
                 "displacement:" WS "[" displacement_list "]" '\n'
                 "slide:" WS SLIDE_FLAG '\n'
                 "guards:" '\n' guard_list

displacement_list ::= vector ("," WS vector)*
vector            ::= "[" INT "," INT "]"
SLIDE_FLAG        ::= true | false
guard_list        ::= ("- " GUARD_NAME ":" guard_rule '\n')+

PIECE_NAME        ::= /[A-Za-z_]+/
GUARD_NAME        ::= /[A-Za-z_]+/
guard_rule        ::= /.+/
```

4.2 Semantic Mapping

Define a mapping Φ from a DSL spec S_p to the pair $(\Delta_p, \mathcal{G}_p)$:

$$\Phi(S_p) = (\Delta_p, \mathcal{G}_p),$$

where:

- Δ_p is the finite set of integer vectors parsed from `displacement`.
- $\mathcal{G}_p$ is the set of guard predicates parsed from `guards`, each compiled into a Boolean function on (u, v, b).

4.3 Completeness Theorem

Theorem 1. For any finite $\Delta' \subset \mathbb{Z}^2$ and any finite set of guard predicates $\mathcal{G}'$, there exists a DSL specification S_p such that $\Phi(S_p) = (\Delta', \mathcal{G}')$.

Sketch. List each vector of Δ' in the `displacement` section. Encode each predicate of $\mathcal{G}'$ as a line under `guards`. The DSL grammar imposes no restriction on the predicate syntax, so every `guard_rule` can be represented. Hence Φ is onto.

$\square$

5 Part V: Extensions and Generalizations

In this section, we explore how the graph-theoretic framework and guard rules can be extended beyond standard and fairy-chess on an 8×8 board, to more general settings:

1. **Arbitrary Board Dimensions.** Extend the vertex set to

$$V = \{(x, y) \mid 1 \leq x \leq n,\ 1 \leq y \leq p\},$$

   and adjust all displacement sets Δ_p accordingly. The move-validation cost remains $O(\max(n, p))$.

2. **Weighted and Multi-Graph Models.** Assign a weight function $w : E \to \mathbb{R}$ to edges so that each move $(u \to v)$ carries an evaluation metric (e.g. control value). This yields a directed weighted graph $G_w = (V, E, w)$ for advanced AI path scoring.

3. **Boundary Directional Constraints.** For nodes on the board border or corners, define a local adjustment

$$\Delta_p(u) \ \subset\ \Delta_p,$$

   removing unavailable directions. This formalizes the reduction of directionality at edges and corners.

4. **Higher-Dimensional Boards.** Generalize to d-dimensional boards, e.g.

$$V = \{(x_1, \ldots, x_d) \mid 1 \leq x_i \leq n_i\},$$

   with $\Delta_p \subset \mathbb{Z}^d$. Guard rules extend verbatim.

5. **Dynamic Rulesets.** Allow Δ_p and guard predicates to depend on time or game state, modeling time-dependent or conditional rules (e.g. unlocking new moves after certain events).

6. **Phase-Dependent Movesets.** Integrate with the phase-automaton DSL (Section IV), enabling distinct

$$\Delta_p^{(\text{phase})} \quad \text{and} \quad \{\text{guards}\}^{(\text{phase})}$$

   per game phase (opening, middlegame, endgame).

6 Part VI: Conclusion and Future Perspectives

In this treatise, we have developed a unified, graph-theoretic framework for validating both classical and fairy-chess moves on an arbitrary $n \times p$ board. By distinguishing pure kinematics (the displacement sets Δ_p and path-clearance predicates) from advanced guard rules (color, pin, check, castling, promotion, en passant), we achieve both rigor and extensibility. A mini-DSL further automates the generation of move validation code directly from declarative specifications.

Key contributions:

- A formal graph model $G = (V, E)$ that supports $O(\max(n, p))$ move-check complexity.
- A modular guard pipeline for advanced rules ensuring soundness and completeness.
- A declarative DSL for piece definitions and guard configurations, enabling rapid prototyping of new variants.
- Extensions to arbitrary board shapes, weighted and higher-dimensional graphs, and dynamic or phase-based rulesets.

Future directions:

- Mechanical verification of the validator using proof assistants (Coq, Isabelle).
- Integration of weighted graph metrics for AI-driven evaluation of control and mobility.
- Support for multi-graph and stochastic move rules in probabilistic or AI-driven games.
- Exploration of 3D and non-Euclidean board geometries under the same formalism.

This theory lays the mathematical groundwork for a truly universal chess engine—*Fibochess*—and for designers of fairy variants seeking a robust, extensible platform.