ABSTRACT

KRISHNAN, ABHIJEET. Interpretable Strategy Synthesis for Competitive Games. (Under the direction of Chris Martens and Arnav Jhala).

Competitive games admit a wide variety of player strategies and emergent, domain-specific concepts that are not obvious from an examination of their rules. Expert agents trained on these games demonstrate many useful strategies, but these are difficult for human players to understand and adopt. Algorithmically revealing these strategies could help players develop a better model for making decisions that lead to victories. In this document, I propose to investigate approaches to the problem of *interpretable strategy synthesis* i.e., the problem of finding algorithmic approaches to learning useful strategies for games that can be understood and applied by a human player. I describe how this problem may be approached using inductive logic programming for chess, and answer set programming for MicroRTS. My hypothesis is that different game domains will require domain-specific strategy representations in order to improve their in-game effectiveness and interpretability for a human player.

Interpretable Strategy Synthesis for Competitive Games

by
Abhijeet Krishnan

A dissertation proposal submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 Motivation

Esports (or e-sports, eSports) has emerged as a rapidly growing industry in recent years, and is projected to reach a market volume of $12.5B by 2030 (Research 2022). It has become a mainstream form of entertainment, attracting millions of viewers and players worldwide (Lynch 2017). As a result, it has also attracted the attention of researchers and analysts who seek to improve the performance of esports athletes, teams and organizations (Reitman et al. 2020).

Esports analytics is a relatively new field that utilizes statistical analysis and data mining techniques to gain insights into the performance of esports players, teams and games. Formally, Schubert et al. (2016) defines it as *the process of using esports related data, primarily behavioural telemetry but also other sources, to find meaningful patterns and trends in said data, and the communication of these patterns using visualization techniques to assist with decision-making processes.* There also exist commercial enterprises that provide esports analytics as a service for player improvement (Esports 2023; Mobalytics 2023).

One such novel source of esports-related data has come from recent advancements in reinforcement learning (RL). These have led to game-playing agents capable of competing with and even outperforming the best human experts at various games like chess (Silver et al. 2018), Go (Silver et al. 2016), Shogi (Li et al. 2020), Mahjong (Silver et al. 2018), *StarCraft II* (Vinyals

et al. 2019) and *Dota 2* (Berner et al. 2019). These agents do not simply take advantage of faster reaction and calculation abilities, but are actually employing new, better strategies that lead to more victories. Borrowing from Jeanette Wing's definition of computational thinking (Wing 2008), these agents appear to have better *abstractions* than human experts for the games they're trained to play.

Despite the existence of such agents in various competitive games, we still see human competition continue to thrive, with these agents leading to new ways of thinking and a re-evaluation of long-held beliefs about the game (Nelson 2019). These discoveries have, so far, involved manual or tool-assisted analysis of the games played by the agents (Sadler and Regan 2019; Zhou 2018). If the agents could themselves explain their strategies and decision-making to human players, I hypothesize that these explanations would help the human players improve.

Strategies in games are policies that tend to lead to favorable states for players. An example of a strategy in the board game chess is *center control* — having your own pieces occupy or threaten the center squares where they have maximum mobility and control over the rest of the board. Players learn to improve in two ways: 1) by identifying good strategies themselves with sufficient practice; and 2) by learning from the ones shared by more experienced players through an established vocabulary. The problem of *strategy synthesis* asks whether such strategies could be automatically identified by a learning algorithm such that they may be communicated to other players.

Strategies have been found, described, and shared by players across a wide variety of games. In the board game Go, the *ladder* is a basic capturing technique that new players must learn to identify and play out correctly (hun Cho 1997). In basketball, the *Triangle offense* is a well-known team strategy that was instrumental in leading the Chicago Bulls to several NBA championships (Rose 2012). In fighting games, a *frame trap* is when you stop attacking an opponent during a sequence of your attacks in order to bait them into starting a move and possibly countering them (Silman 2021). Finding useful strategies for playing a game—and an expressive vocabulary to describe them—can help players get better at a game.

Existing work in strategy synthesis for games relies on extensive domain knowledge found by human players for a game, and cannot be easily adapted to a new game. These works define game strategies as complete game-playing policies, while most human-found strategies typically describe short sequences of moves or features of the game state (see Chapter 2).

1.2 Thesis Statement

The research I propose is focused on the problem of *interpretable strategy synthesis* (ISS) for games and how it can be solved for a variety of game domains. I will formalize the problem

of ISS for games by providing clear definitions for relevant terms and defining metrics for success. I will approach the problem of ISS for two game domains that are qualitatively different from each other, namely chess and MicroRTS (Ontanon 2021). Finally, I will investigate the factors affecting the interpretability of a synthesized strategy by a human player. The proposed dissertation will answer the following research questions:

RQ1. How do we formally define the problem of ISS?

As detailed in Chapter 2, prior works have attempted to solve the problem of ISS, but have not labeled it as such. Subsequent works have explicitly mentioned the attempt to solve the problem of ISS, but have not situated it in a broader context. Providing a unified framework for ISS will help researchers compare and reuse work across multiple game domains. I answer this in Chapter 3.

RQ2. How do we approach the problem of ISS for the game of chess?

Chess has a long history as a competitive game and as a testbed for AI research. It has both a rich literature of competitive strategies and a thriving software ecosystem of chess-playing agents. This usefully positions it as a game to approach ISS in and compare any learned strategies with known ones. In the following RQs, I address how the problem of ISS may be approached for chess by defining a computational representation of a chess strategy and proposing a corresponding learning algorithm, namely inductive logic programming (ILP).

RQ2(a) Could we represent known chess tactics as a learnable, computational model of a chess-playing policy and develop metrics to show that they suggest better moves than a random baseline?

The framework for ISS proposed as part of **??** needs to be shown to be feasible in synthesizing useful strategies. In Sections §4.1.1, §4.1.2 and §4.1.3, I show how we may computationally model a chess strategy and translate known chess tactics into the model. I show that these tactics produce better moves on average compared to a random player, thus answering RQ2(a).

RQ2(b) Do the chess strategies learned using ILP outperform a random baseline in how closely their divergence scores approximate a beginner player?

While the feasibility of the framework proposed as part of **??** is to be evaluated in RQ2(a), it remains to be seen whether a suitable learning method can be developed to learn chess strategies from gameplay. In Section §4.1.4, I show how we can use inductive logic programming (ILP), a symbolic machine learning (ML) method, to serve as this learning algorithm.

RQ2(c) Do the chess strategies learned by an ILP system incorporating the changes of the new predicate vocabulary and precision/recall-based constraints produce moves better than those learned by an ILP system without these modifications?

The chess strategies learned by the system proposed as part of RQ2(b) do not produce moves comparable to a human beginner, let alone a strong chess-playing agent. In Section §4.1.5, I show how modifications to the learning algorithm can be made to improve the quality of the strategies it learns.

RQ3. How do we approach the problem of ISS for the game of MicroRTS?

MicroRTS is a small implementation of an RTS (real-time strategy) game, designed to perform AI research. In comparison to chess, it is real-time, partially observable, and durative. While there exist many hard-coded agent scripts to play the game, there is not as much literature on strategies that human players use. In the following RQs, I propose an approach to the problem of ISS for MicroRTS which uses a programmatic representation of a strategy derived from a specific context-free grammar, and an answer-set programming (ASP) based approach to learn it. I also propose to investigate the interpretability aspect of the synthesized strategies by running a user study to find which factors of the strategy model most affect interpretability.

RQ3(a) Does an ASP-based approach towards developing a synthesizer for the SynProS competition achieve the highest overall score across all test maps?

The SynProS competition (Moraes 2021a) provides a strategy model for MicroRTS in the form of a programmatic script derived from a context-free grammar (CFG). A synthesizer entered into the competition must generate a script for an input map in this derived from this CFG, which is then evaluated using a round-robin tournament run against the scripts synthesized by other synthesizers, as well as some baseline hard-coded scripts, and measuring their win rates. In Section §4.2.3, I propose investigating an approach that uses answer-set programming to design a synthesizer that can search for scripts that achieve a high win rate.

RQ3(b) Which features of a MicroRTS strategy model have a significant correlation with the interpretability of said strategy as measured by a user study?

While a lot of research has focused on the synthesis portion of ISS, there is a lack of work studying the interpretability of the synthesized strategies. I propose to run an exploratory user study that will use a set of pre-synthesized MicroRTS strategies that vary along various variables as detailed in Section §4.2.4, and ask participants to rate the interpretability of a strategy using a survey. Finding which factors most

correlate with interpretability will allow future researchers to develop methods that optimize for those factors when synthesizing strategies.

The work proposed in this dissertation will benefit the esports analytics industry by providing an additional, powerful tool to gain insight into player strategies. It will benefit explainable RL (XRL) research by contributing an additional technique to explain the behavior of RL agents. Finally, it will benefit research in intelligent tutoring systems by investigating the factors involved in the interpretability of an explanation by a human learner.

CHAPTER

2

RELATED WORK

2.1 Interpretable Strategy Synthesis

The problem of strategy synthesis can be framed in RL terms as that of learning a game-playing policy. However, most existing, state-of-the-art policies are much too complex for any human to memorize and simulate. For example, the default neural network (NN) model [1] provided with the chess-playing agent *Stockfish 15.1* consists of over 100,000 trainable parameters, which is impossible for a human chess player to keep in memory. An informal attempt at doing so with a much simpler model was also not successful (Deutsch 2017) The goal of interpretable strategy synthesis (ISS) is to learn game-playing policies that can be *interpreted* by humans. This could involve being able to simulate the model in order to play the game, or being able to read the learned policy. The overarching goal is for the synthesized strategies to provide some value to players in their own games.

The earliest works in ISS primarily learn rule-based agents. The rules are either selected from an authored list of rules, as in Spronck et al. (2004); Canaan et al. (2018), or derived from a representation like a context-free grammar, as in Mariño et al. (2021); de Freitas et al. (2018). Rule-based agents are primarily selected based on a qualitative appeal to their interpretability. None of these works attempt to empirically quantify the interpretability of their rule-based

[1] https://tests.stockfishchess.org/api/nn/nn-52471d67216a.nnue

agents, however.

Given the prevalence of rule-based strategy representations in previous ISS work, most learning methods involve searching over the space of rules to find a combination of rules that optimize the performance criteria (usually win rate, or score). These methods typically consist of derivative-free optimization techniques (Rios and Sahinidis 2013) like genetic algorithms, simulated annealing and local search.

The domains that have been tackled in prior ISS work have either been games with small action spaces, such as Blackjack (de Mesentier Silva et al. 2016), Nonograms (Butler et al. 2017) and Hanabi (Canaan et al. 2018), or simplified versions of more complex games such as MicroRTS (Mariño et al. 2021; Medeiros et al. 2022; Mariño and Toledo 2022). Spronck et al. (2004) tackles just the battle system in the commercial single-player game *Neverwinter Nights*.

Evaluation of strategies is primarily done by comparing their performance (measured using win rates, or scores) against hand-authored strategies. None of these works attempt to formally define what an "interpretable" strategy is, create metrics to measure the interpretability of a strategy, or attempt to evaluate the interpretability of a strategy with a user study. There are only appeals made to the interpretability of synthesized strategies due to the rule-based nature of the strategy models, which are readable by humans. Butler et al. (2017) compares synthesized strategies to known, existing strategies and measures how many of the known strategies were replicated by the synthesized set. Mariño et al. (2021); Mariño and Toledo (2022) provide a qualitative analysis of learned strategies.

2.1.1 Pattern Learning in Chess

Patterns have been used to suggest moves and guide playing strategies in middle-game positions (Berliner 1975; Pitrat 1977; Wilkins 1979) and endgames (Huberman 1968; Bramer 1977; Bratko 1982). (Levinson and Snyder 1991) used weighted patterns in their Morph system as an evaluation function to guide playing strategy. Very recent work has attempted to directly probe neural network engines to test for the presence of human concepts (McGrath et al. 2021). (Morales 1992) developed the PAL system to learn first-order patterns in chess using ILP.

2.2 Explainable Reinforcement Learning

Humans usually come up with strategies by observing what works in their own play, or studying the gameplay of other experts. Since human players can be treated as a game-playing policy, the problem of interpretable strategy synthesis is equivalent to that of *explaining* the policy of human players, or more generally, any game-playing agent.

Attempts to make RL agent policies amenable to human interpretation have been pursued in the field of explainable reinforcement learning (XRL). Puiutta and Veith (2020) provide a survey of recent XRL methods. An interpretability technique that has received some attention is that of training an inherently interpretable *surrogate model* which matches the performance of the original agent. Options for this surrogate model that have been investigated include decision trees (Bastani et al. 2018; Coppens et al. 2019; Sieusahai and Guzdial 2021) and programmatic policies (Verma et al. 2018; Trivedi et al. 2021).

A common theme in early XRL (and the more general field of explainable artificial intelligence) work is the imprecise definition of "interpretability" and a lack of metrics to quantify it. For example, Verma et al. (2018) evaluate the performance of their methods empirically but their appeal to their programmatic policy's interpretability rests on its representation as readable rules. A measurement of performance, but not interpretability, is repeated in other XAI works as well (Coppens et al. 2019; Liu et al. 2019; Shu et al. 2017; Hayes and Shah 2017).

This lack of precision in defining and measuring interpretability led to the work by Doshi-Velez and Kim (2017) which provides a definition of interpretability grounded in psychology, and a taxonomy for evaluating interpretability. Subsequent work has attempted to quantify the interpretability of proposed XAI techniques (Madumal et al. 2020; Slack et al. 2019; van der Waa et al. 2018) and studied the nature of interpretability itself (Kliegr et al. 2021; Lage et al. 2019).

CHAPTER

3

A FRAMEWORK FOR INTERPRETABLE STRATEGY SYNTHESIS

The problem of interpretable strategy synthesis (ISS) is first mentioned in Butler et al. (2017). It identifies it as a subset of automated game analysis, and implicitly solves it using a specific *strategy model* for the game of Nonograms. It evaluates the performance of the synthesized strategies, but doesn't make an attempt to evaluate their interpretability. As detailed in Chapter 2, prior works have attempted to solve the problem of ISS, but have not labeled it as such. Subsequent works have explicitly mentioned the attempt to solve the problem of ISS, but have not situated it in a broader context. In my thesis, I will attempt to formally define the problem of ISS by abstracting common elements and providing clear definitions, thereby answering **??**. In Section §4.1.3, I will also demonstrate a proof-of-concept for how this formalization can help solve the ISS problem for a hitherto untackled domain, which will answer RQ2(a).

3.1 Problem Definition

Let us be given a game environment $\mathcal{G}$ modeled as a finite, episodic Markov decision process $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}, \mathcal{R}, \gamma \rangle$, where —

- $\mathcal{S}$ is the finite set of legal game states reachable from the start state s_0,
- $\mathcal{A}(s)$ is the set of legal actions that can be taken in a state $s \in \mathcal{S}$,
- $\mathcal{P}$ is the state transition function that models the game's dynamics,
- $\mathcal{R}$ is the reward function describing the game's win conditions, and
- γ is a discount factor between $[0, 1]$

We define a *strategy* σ for $\mathcal{G}$ at timestep t as a probability distribution over the available actions in a state, for a *subset* of states in the state space. The states for which σ is defined are termed as the states for which the strategy is *applicable*, and is given by $\mathrm{A}_\sigma \subseteq \mathcal{S}$.

$$\sigma(a|s) \doteq \mathbb{P}[A_t = a | S_t = s], \forall s \in \mathrm{A}_\sigma, a \in \mathcal{A}(s) \tag{3.1}$$

The definition of a strategy in equation (3.1) borrows from the notion of a *policy* as used in RL (Sutton and Barto 2018) as well as game-theoretic notions of a strategy (Boros et al. 2012). Intuitively, a strategy describes a pattern or feature of a game state that comes up often in regular gameplay and that tends to influence the actions taken in states with that feature. A strategy is not a general policy and cannot be used to sample an action in states outside the set of applicable states.

A *strategy model* (M) is a computational program that can be run to realize equation (3.1). A strategy is an instance of a strategy model. The strategy model serves as a *template* for a strategy that a learning algorithm must output. In practice, it is tightly coupled to the learning algorithm and could incorporate domain-specific knowledge to improve its performance or interpretability. Examples of strategy models used in literature are if-else rules or programmatic scripts derived from a domain-specific language.

I propose the notion of an *interpretability measure* ($\mathcal{I}$), which is a function that returns a score for how interpretable a particular strategy σ is. Since the interpretability measure is intended to be an objective function which must be optimized for the ISS problem, it ought to be computational in nature. Obtaining human-oriented measures of interpretability might be prohibitively expensive if used for training, and ought to be used only for evaluating the learned strategy models. Most current research in ISS does not define an interpretability measure, nor do they conduct an empirical evaluation of the interpretability of the learned strategy models (see Table 3.1).

The ISS problem $\langle \mathcal{G}, \mathcal{R}, M, \mathcal{I} \rangle$ is then the problem of finding a strategy model σ^* that simultaneously optimizes both reward and interpretability. This is described in Equation (3.2).

$$\sigma^* \doteq \arg\max_{\sigma} \mathcal{R}(\sigma) I(\sigma) \tag{3.2}$$

Paper	**Domain**	**Model**	**Algorithm**	$\mathcal{R}$	$\mathcal{I}$
Spronck et al. (2004)	Neverwinter Nights	authored rules	dynamic scripting	win rate	–
de Mesentier Silva et al. (2016)	Blackjack	if-else rules	several search algorithms	score	–
Butler et al. (2017)	Nonograms	condition-action rule	SMT solver	coverage	–
Canaan et al. (2018)	Hanabi	authored rules	genetic algorithm	score	–
de Freitas et al. (2018)	Mario AI Framework	CFG-based DSL script	genetic algorithm	level score	–
Mariño et al. (2021)	MicroRTS	CFG-based DSL script	local search	win rate	–
Krishnan and Martens (2022a)	Chess	FO logic rules	ILP	coverage, divergence	–
Mariño and Toledo (2022)	MicroRTS	CFG-based DSL script	genetic algorithm	win rate	–
Medeiros et al. (2022)	MicroRTS, Can't Stop	CFG-based DSL script	SA, UCT	win rate	–

Table 3.1: List of works in ISS and their classification according to the proposed ISS framework.

CHAPTER

4

ALGORITHMS FOR INTERPRETABLE STRATEGY SYNTHESIS

I propose to investigate various algorithms to solve the ISS problem for a variety of game domains. Primarily, I will restrict the scope of my investigation to two specific domains - 1) chess and 2) MicroRTS. These have been chosen for the large quantity of existing research and literature available on both of them, and the qualitative differences between the domains. For example, chess is a turn-based, fully-observable game, whereas MicroRTS is a real-time, partially observable game.

4.1 Chess

The earliest known precursor to the game of chess was a game called *chaturanga* first played in India in the 7th century CE. Its modern form took shape in Spain in the 15th century (Averbakh 2012). This long history has led to a wealth of study and analysis of the game over time, which has only been accelerated with the advent of chess-playing computers (chess engines). This is useful because it provides a rich set of existing player strategies to compare against.

Chess' extensive history and appeal as a test of intelligence has led it to be called the

drosophila[1] of artificial intelligence (McCarthy 1990). It has been a mainstay of AI research from the invention of the digital computer (Claude 1950) to the neural network revolution (Silver et al. 2018). There has been a lot of research investigating the idea of using patterns to guide a computer to play chess is not new. Patterns have been used to suggest moves and guide playing strategies in middle-game positions (Berliner 1975; Pitrat 1977; Wilkins 1979) and endgames (Huberman 1968; Bramer 1977; Bratko 1982). Levinson and Snyder (1991) used weighted patterns in their Morph system as an evaluation function to guide playing strategy. Very recent work has attempted to directly probe neural network engines to test for the presence of human concepts (McGrath et al. 2021). Morales (1992) developed the PAL (Patterns and Learning) system to learn first-order patterns in chess using ILP (inductive logic programming).

For the game of chess, I present a model for a human-readable chess strategy, inspired by documented chess tactics, for playing chess (Krishnan and Martens 2022b). I used this to model known chess tactics from literature. Using the novel metrics of coverage and divergence, I show that these tactics produced better moves than a random baseline. This answers RQ2(a). I next present a system that uses a modified ILP system to optimize these metrics for learning these tactics from gameplay data (Krishnan and Martens 2022a). This system can learn chess tactics from positions drawn from play traces of human vs. human games. It requires no more domain knowledge than the base rules of chess. I show that the learned tactics generalize well to real-world chess positions and produce moves that outperform a random player. This answers RQ2(b). Finally, I improve upon this system by introducing new constraints to reduce the search space, and a new predicate vocabulary. Using an ablation study, I show that these changes significantly ($p < 0.01$) reduce the difference in strength between the moves suggested by the tactics, and the ground truth (Krishnan et al. 2023). This answers RQ2(c).

4.1.1 Strategy Model for Chess

Tactics in chess are maneuvers that take advantage of short-term opportunities (Seirawan 2005, Chapter 1). They are a move or series of moves that bring about an immediate advantage for the player. They are identified by *motifs*—positional patterns that indicate whether a tactic might exist in the given position. An example of a tactic is the *fork*, where a piece simultaneously attacks two opponent pieces at once (see Figure 4.1).

Our strategy model for chess uses the concept of a *chess tactic*. Formally, we define our strategy model for chess (hereafter referred to as a tactic) as a first-order logic rule expressed in Prolog using a particular predicate vocabulary. As seen in Figure 4.2, the rule head of the tactic consists of the variables `Position`, `From` and `To`. The input state is described by `Position`,

[1] fruit fly; easily bred and thus extensively used in genetics research
[2] https://lichess.org/training/4pEpj

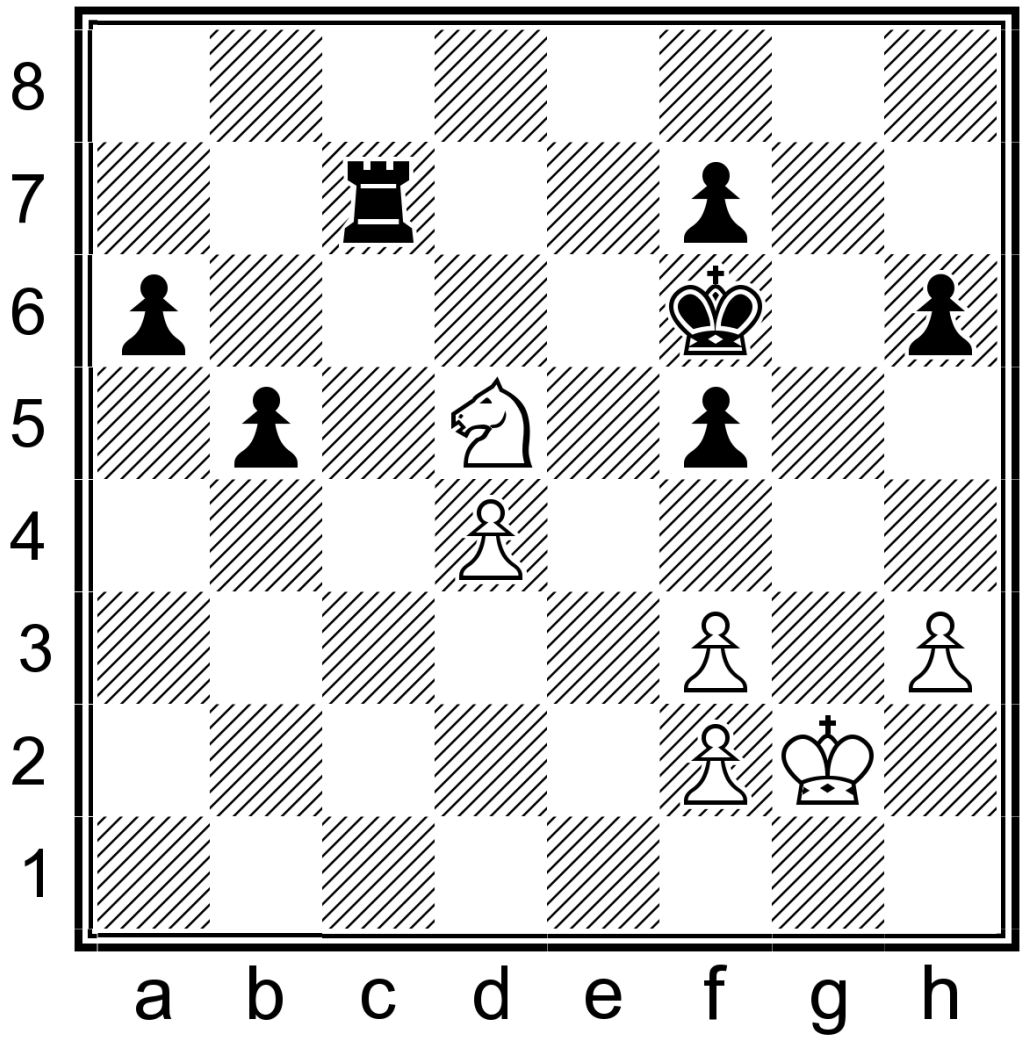

Figure 4.1: An example of a fork from a Lichess puzzle[2]. The white knight on d5 simultaneously checks the opponent king on f6 and attacks the rook on c7.

which is also expressed in first-order logic using our predicate vocabulary. `From` and `To` describe the output action, namely, the move which begins from the square `From` and ends on the square `To`. This replicates the long algebraic notation for moves used in the Universal Chess Interface (UCI) protocol, an open communication protocol allowing chess engines to interact with user interfaces (Kahlen 2004).

```
tactic(Position, From, To) ←
        feature_1(···),
        feature_2(···),
        ⋮
        feature_n(···)
```

Figure 4.2: Our tactic model expressed in Prolog pseudocode. Every `feature_i` clause is a rule defined in the predicate vocabulary.

A single tactic, or even a set of tactics, does not represent complete policy for playing chess. This is because we might encounter a position for which no tactic matches. In this case, our model cannot make a move. There might also be positions to which multiple tactics apply, in which case an arbitration process for selecting a single move among the various suggestions is not obvious.

If the tactic is provided as a query to the Prolog interpreter with a grounded instance of `Position` and non-ground move variables, it will attempt to *unify* the latter with ground moves (Sterling and Shapiro 1994). The variable binding(s) (i.e., moves) it finds are treated as the action distribution defined on the state, with every found move being equi-probable, and every other move accorded a probability of 0.

Our usage of first-order logic to model chess tactics is motivated by the following reasons —

1. Chess tactics are an important concept that human players use to think about chess (Szabo 1984) and are useful in chess education (Gobet and Jansen 2006).

2. First-order logic has been extensively used to model chess patterns (see Section §2.1.1).

3. Logic rules are commonly acknowledged to be interpretable and have a long history of research (Zhang et al. 2021).

4.1.2 Metrics to Evaluate Chess Strategy Models

A computational representation of a chess strategy (tactic) is not enough; we need a computational evaluation metric to guide any potential learning system towards "good" tactics and away from "bad" tactics. Characteristics of good tactics are — 1) they produce strong moves, i.e., they produce moves that tend to lead to winning game states, and 2) they are encountered relatively frequently in the states reached during typical gameplay. To realize these goals, we introduce two metrics — *coverage* and *divergence*.

4.1.2.1 Coverage

A tactic σ's coverage for a set of positions P is calculated as —

$$P_{\mathrm{A}} \doteq P \cap \mathrm{A}_{\sigma} \tag{4.1}$$

$$\mathrm{Coverage}(\sigma, P) \doteq \frac{|P_{\mathrm{A}}|}{|P|} \tag{4.2}$$

Coverage is the fraction of positions in a given set to which the tactic is *applicable*. If P is representative of positions encountered in games, coverage is a useful measure of how likely it is that the tactic can be used to make moves in games. A low coverage value indicates that the tactic is *situational*, whereas a high coverage value indicates that the tactic is general.

4.1.2.2 Divergence

In order to find good moves, chess engines (chess-playing agents) must have some way of evaluating a board position to produce a numerical score. A good move is one which results in the maximum increase of this evaluation. In RL terms, a chess engine must have a good state-value function V_{π}.

Classical chess engines used a collection of hand-crafted heuristics to evaluate a position (Bijl and Tiet 2021). Modern neural-network based engines use a combination of heuristics and neural network models trained on millions of positions to produce evaluations. These evaluations are standardized to be reported in *centipawns* (Cp), or roughly $1/100^{\mathrm{ths}}$ of the value of a pawn (Guid and Bratko 2017). Positive values indicate white is favoured, and negative values indicate black is favoured. A position evaluated to be +1 roughly corresponds to an advantage of a single pawn for white. If a sequence of moves leading to a forced checkmate is found (the condition for winning in chess), then that is reported instead.

To measure the quality of moves suggested by a tactic, we extend a metric previously used to analyse world chess champions (Guid and Bratko 2006, 2011; Romero 2019). A chess engine

E usually provides a position evaluation function $v_E(s)$. From this, we can obtain a move evaluation function $q_E(s,a)$ as follows —

$$q_E(s,a) = \sum_{s',r} \mathcal{P}(s',r|s,a)[r + v_E(s')] \tag{4.3}$$

$$= v_E(s'), s' \text{ is non-terminal} \tag{4.4}$$

Equation equation (4.4) follows from equation (4.3) since rewards in chess are 0 for non-terminal states, $\gamma = 1$, and chess rules are deterministic. A chess engine could evaluate a position to be a 'Mate in X' rather than a numerical score. In this case, we assign an arbitrary large value to the evaluation.

Given two moves a_1, a_2 made in a position s, we can calculate their difference $d_E(s,a_1,a_2)$ as —

$$d_E(s,a_1,a_2) \doteq \mid q_E(s,a_1) - q_E(s,a_2) \mid \tag{4.5}$$

We can now define the *divergence* of a tactic from a set of examples P as the average difference between the moves suggested by the tactic and the ground truth move —

$$\text{Divergence}_E(\sigma(\cdot), P) \doteq$$
$$\frac{1}{|P_A|} \sum_{(s,a_1)\in P_A} \sum_{a_2 \in \mathcal{A}(s)} \sigma(a_2|s) d_E(s,a_1,a_2) \tag{4.6}$$

Divergence is also reported in units of centipawns (Cp). Divergence of a tactic from the ground truth is low and close to 0 when its suggestions are similar in engine evaluation to the ground truth moves, and takes on large values when it differs significantly. Divergence is a useful metric to measure how closely a tactic approximates a reference policy.

We do not introduce a measure $\mathcal{I}$ for the interpretability of a chess tactic. This is a useful area for future work.

4.1.3 Evaluation of Known Chess Tactics

In a proof-of-concept study of the proposed ISS framework, I show how the symbolic strategy model (tactics) for chess as described in Section §4.1.1 can be learned using this framework. I use patterns learned by an existing inductive logic programming (ILP) system called PAL (Patterns and Learning) (Morales 1992) to derive these tactics. I then present an evaluation

of a set of tactics obtained from PAL against a random baseline using the metrics defined in Section §4.1.2. In showing that the learned tactics resemble a weak chess-playing program, but are unable to outperform a strong one, I answer RQ2(a).

4.1.3.1 Background

4.1.3.1.1 PAL System The PAL (Patterns and Learning) system was introduced in Morales (1992). It attempts to use ILP to synthesize patterns for chess play, which are expressed using a subset of Horn clause logic. It contributes a predicate vocabulary for expressing these patterns and chess positions as Horn clauses. The pattern-learning problem is framed as an ILP problem, for which a heuristically-constrained version of the *rlgg* (relative least general generalization) algorithm is used to induce plausible hypotheses. Patterns learned can be *static* and not involve any piece movement, or be *dynamic* and describe multi-move tactics. We expand upon how the PAL system formally defines and synthesizes these chess patterns.

4.1.3.1.1.1 Pattern Formalism A pattern in PAL is formally defined as a non-recursive Horn clause of the form

$$\text{Head} \rightarrow D_1, D_2, \cdots, D_n, F_1, F_2, \cdots, F_m$$

where,

- Head is the head of the pattern definition
- The D_i are "input" predicates used to describe the position and represent pieces involved in the pattern
- The F_j are instances of definitions which are either provided as background knowledge or learned by PAL, and represent the conditions (relations between pieces and places) to be satisfied by the pattern.

An example of a *checking move* pattern, where a move that puts the opponent king in check is suggested, is reproduced from the paper in Figure 4.3. A key predicate is `make_move`, which determines whether a pattern is static or dynamic. The `contents` predicates are used to describe the position on the board. The remaining predicate definitions are provided as background knowledge.

```
can_check(S1,P1,(X1,Y1),S2,king,(X2,Y2),(X3,Y3),Pos1) ←
        contents(S1,P1,(X1,Y1),Pos1),
        contents(S2,king,(X2,Y2),Pos1),
        other_side(S1,S2),
        ¬ in_check(S2,(X2,Y2),P1,(X1,Y1),Pos1),
        make_move(S1,P1,(X1,Y1),(X3,Y3),Pos1,Pos2),
        in_check(S2,(X2,Y2),P1,(X3,Y3),Pos2).
```

Figure 4.3: PAL rule for the `can_check` pattern. A piece (P1) can check the opponent's King after moving to (X3,Y3).

4.1.3.1.1.2 Pattern Synthesis The input to the PAL generalization algorithm is a set of pattern definitions (both predefined and learned) along with a description of a chess position (as ground unit clauses). The algorithm extends Buntine's method for constructing the *rlgg* of two clauses to multiple clauses. It uses the following constraints and heuristics to limit hypothesis size and increase the algorithm's generalisation steps -

- Disallowing variables in the head or body of a rule which are not *connected* to a literal i.e., not equal to a variable of that literal
- Labeling constants occurring in the ground literals of a rule body to make patterns piece-invariant
- Restricting the legal moves from a position to only be those which introduce a new predicate name or remove an existing predicate name

PAL uses an automatic example generator to manually guide the generalization algorithm towards learning desired concepts. Given an example of the target concept, the generator *perturbs* the example to create a new example for which a classification label must be provided. To restrict the example space searched, the automatic example generator attempts to generate examples which specialize the current hypothesis in case of a prior positive example, or generalize it in case of a prior negative example. We refer interested readers to the original thesis for further details.

4.1.3.2 Methodology

4.1.3.2.1 Implementation using PAL We use the PAL system to synthesize tactics. We select seven patterns that PAL was shown to learn, and modify them to output a move suggestion. These patterns and their verbal definitions are listed in Table 4.1. All patterns learned other than `pin` are 1-ply *dynamic* patterns, which means they include a single `make_move` predicate

in the rule body looking ahead one move. We modify these patterns to introduce a `suggestion` predicate with the same variables as `make_move`. For `pin`, which is a static pattern as learned by PAL, we convert it into a dynamic pattern as shown in Figure 4.4 and introduce the `suggestion` predicate in the same way.

```
pin(S1,P1,(X1,Y1),S2,king,(X2,Y2),S2,P3,(X3,Y3),(X4,Y4),Pos1) ←
    sliding piece(P1,(X1,Y1),Pos1),
    make_move(S1,P1,(X1,Y1),(X4,Y4),Pos1,Pos2)
    sliding piece(P1,(X4,Y4),Pos2),
    stale(S2,P3,(X3,Y3),Pos2),
    threat(S1,P1,(X4,Y4),S2,P3,(X3,Y3),Pos2),
    in_line(S2,king,(X2,Y2),S2,P3,(X3,Y3),S1,P1,(X4,Y4),Pos2).
```

Figure 4.4: Modified PAL rule for the `pin` pattern to convert it into a tactic

Pattern	Definition
`can_threat`	A piece (P1) can threaten another piece (P2) after making a move to (X3,Y3)
`can_fork`	A piece (P1) can produce a fork to the opponent's King and piece (P3) after making a move to (X4,Y4)
`can_check`	A piece (P1) can check the opponent's King after a moving to (X3,Y3)
`discovered_check`	A check by piece (P2) can be "discovered" after moving another piece (P1) to (X4,Y4)
`discovered_threat`	A piece (P1) can threaten an opponent's piece (P3) after moving another piece (P2) to (X4,Y4)
`skewer`	A King in check by a piece (P1) "exposes" another piece (P3) when it is moved out of check to (X4,Y4)
`pin`	A piece (P3) cannot move because it will produce a check on its own side by piece (P1)

Table 4.1: Patterns learned by the PAL system that are used to create tactics

4.1.3.3 Evaluation

We wish to investigate whether the synthesized tactics tend to suggest good moves to play. We do this by measuring coverage and divergence for each of our tactics over a set of positions using both a strong and a weak reference engine. For our strong reference engine, we use *Stockfish*

14, the winner of the TCEC 2020 Championship (Haworth and Hernandez 2021). For our weak reference engine, we use Maia Chess (McIlroy-Young et al. 2020), a chess engine trained to produce human-like moves. We use the `maia1` model, which is targeted toward 1100 ELO (a measure of relative playing strength, and roughly equal to a beginner). We limit the search depth to 1-ply for both Stockfish 14 and Maia 1100 to resemble our tactics. As a baseline, we use a random tactic which is applicable to all positions and produces a random legal move in the position. We limit the number of suggestions from a tactic to 3, and assume the output order of tactic suggestions as the intended ranked order. For ease of implementation, we manually translated the tactics from Prolog definitions to Python functions. We use 5000 games from the January 2013 archive of standard rated games played on lichess.com (lichess.org 2021). For each game, we generate positions by iterating through the move list, making the move, and adding the resulting position to the evaluation set. In total, we generate 325,830 positions.

4.1.3.4 Results and Analysis

We summarize the results of our evaluation in table 4.2.

From the high coverage values obtained, we conclude that tactics like `can_threat` and `discovered_threat` are too general, whereas tactics like `discovered_check` are too specific. Tactics like `can_check`, `can_fork` and `skewer` strike a balance between these extremes.

From the divergence metrics calculated using Maia-1100 (our weak engine), we see that most of our tactics have lower divergence scores than our random baseline, indicating that they tend to produce moves which are evaluated somewhat similarly to a weak engine's best moves. For Stockfish 14, however, all our tactics have higher divergence scores than random, indicating that they do not tend to produce moves similar to a strong engine. Thus, we qualitatively conclude that our tactics resemble that of a beginner chess player.

Tactic	Coverage	Divergence	
		SF14	Maia
`can_threat`	0.96	378.94	9.22
`can_check`	0.45	549.19	4.02
`can_fork`	0.32	676.45	4.67
`discovered_check`	≈0	338.55	18.64
`discovered_threat`	0.96	375.97	1.19
`skewer`	0.22	748.4	5.41
`pin`	0.79	526.45	4.9
`random`	**1**	**328.09**	**8.28**

Table 4.2: Coverage and DCG for each tactic

4.1.3.5 Conclusion and Future Work

We have described a symbolic sub-policy model for chess inspired by the pattern-action model of chess tactics. We have used patterns learned by an ILP system to construct these tactics. We have contributed a metric for measuring the divergence of these tactics to a reference chess-playing agent. We evaluated a set of tactics learned by a chess pattern learning system using our metric to find that they resembled a weak engine, but were not similar to a strong one.

We use patterns learned by PAL to obtain our tactics. However, PAL uses manual labeling of generated examples to learn specific concepts, and requires additional effort to convert the learned patterns into tactics for our model. We aim to investigate the automatic learning of tactics from a dataset of chess positions and move suggestions using ILP as implemented by modern systems like Popper (Cropper and Morel 2021). Future work could investigate alternate ILP algorithms to use our divergence metric as a loss function to optimize tactics for (Evans and Grefenstette 2018).

Our tactic model is loosely inspired by how chess tactics are learned and practiced. However, our tactics are limited to looking 1-*ply* in the future i.e., they can only recognize the presence of a matching pattern in the immediately next position. Many chess tactics suggest *combinations* of moves, a series of moves where the matching pattern shows up only in a particular sequence (see Figure 4.5). Extending the tactic model to express and recognize such combinations will be a useful avenue for future work. We also wish to investigate the expression of longer-term plans from chess literature like centre control and pawn structure using tactics.

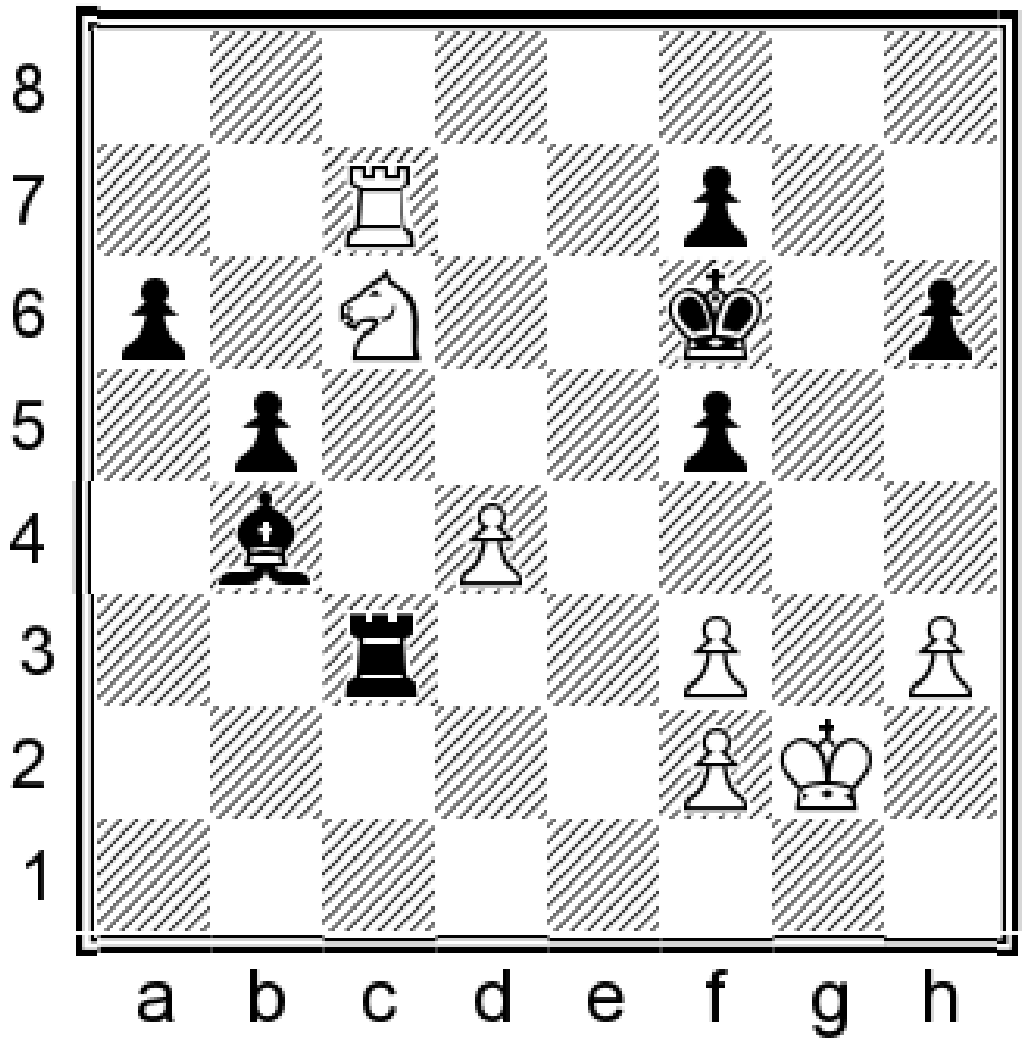

Figure 4.5: An example of the limitations of our 1-ply `can_fork` tactic. White has no immediate forking move here, leading to the tactic not matching. However, if they play **1. Nxb4**, then Black's best response is **1. ... Rxc7** which allows a fork with **2. Nd5+** leading to the capture of the rook.

Our appeal to the interpretability of these tactics rests on similar claims made regarding the interpretability of rule-based strategies. Future work will involve rigorously testing these assumptions with user studies using evidence-based measures of interpretability (Lage et al. 2019; Kliegr et al. 2021). Specifically, we wish to investigate the ease of learning and applying these tactics in real games played by human players.

4.1.4 Learning Tactics using Inductive Logic Programming

In this work, I present a system that integrates a modified inductive logic programming (ILP) engine with coverage and divergence metrics for learning tactics. My system can extrapolate chess tactics from positions drawn from play traces of human vs. human games. It requires no more domain knowledge than the base rules of chess. I show that the learned tactics generalize well to real-world chess positions and produce moves that outperform a random player. I discuss some limitations of our approach and conclude with directions for future work. In so doing, I answer RQ2(b).

4.1.4.1 Background

4.1.4.1.1 Inductive Logic Programming Inductive logic programming (ILP) is a form of symbolic machine learning that aims to induce a hypothesis (a set of logical rules) that generalizes given training examples. It can learn human-readable hypotheses from smaller amounts of data than neural network-based models (Cropper and Dumančić 2020).

An ILP problem is specified by three sets of Horn clauses —

- B, the background knowledge,
- E^+, the set of positive examples of the concept, and
- E^-, the set of negative examples of the concept.

The ILP problem is to induce a hypothesis $H \in \mathcal{H}$ (an appropriately chosen hypothesis space) that, in combination with the background knowledge, entails all the positive examples and none of the negative examples. Formally, this can be written as —

$$\forall e \in E^+, H \cup B \models e \text{ (i.e., } H \text{ is } \textit{complete)}$$
$$\forall e \in E^-, H \cup B \not\models e \text{ (i.e., } H \text{ is } \textit{consistent)}$$

To make the ILP problem more concrete, we reproduce an example from the paper by Cropper and Dumančić (2022).

Example 1. *E^+ and E^- contain positive and negative examples of the target* `last` *relation respectively. B contains background knowledge, i.e., clauses that might be useful in inducing a hypothesis for* `last`.

$$E^+ = \left\{ \begin{array}{l} \texttt{last([m,a,c,h,i,n,e], e).} \\ \texttt{last([l,e,a,r,n,i,n,g],g).} \\ \texttt{last([a,l,g,o,r,i,t,h,m],m).} \end{array} \right\}$$

$$E^- = \left\{ \begin{array}{l} \texttt{last([m,a,c,h,i,n,e], m).} \\ \texttt{last([m,a,c,h,i,n,e], c).} \\ \texttt{last([l,e,a,r,n,i,n,g],x).} \\ \texttt{last([l,e,a,r,n,i,n,g],i).} \end{array} \right\}$$

$$B = \left\{ \begin{array}{l} \texttt{empty(A) :- ...} \\ \texttt{head(A,B) :- ...} \\ \texttt{tail(A,B) :- ...} \end{array} \right\}$$

From this information, we could induce a hypothesis for `last` *as —*

$$H = \left\{ \begin{array}{l} \texttt{last(A,B) :- head(A,B),tail(A,C),empty(C).} \\ \texttt{last(A,B) :- tail(A,C),last(C,B).} \end{array} \right\}$$

4.1.4.1.2 Popper Popper is an ILP system that implements an approach called *learning from failures* (Cropper and Morel 2021). It operates in three stages: generate, test and constrain. Given an ILP problem, it generates a candidate solution, tests it against examples in the training set, and formulates constraints based on the outcome of failed examples to cut down the solution space. These three stages are repeated until a solution is found. Popper also allows for specifying hypothesis biases to further constrain the solution space based on domain knowledge of the problem being solved.

Popper uses two general types of constraints – *generalization* and *specialization*. Assume an ILP problem $\langle E^+, E^-, B \rangle$ and a generated hypothesis H. If H entails a negative example, then H is too general and so we can prune generalisations of it. Similarly, if H does not entail a positive example, it is too specific, and we can prune specialisations of it. The notions of the generalisation and specialisation of a hypothesis are defined in terms of *clausal subsumption*, and we refer readers to the original paper for formal definitions of these terms. Popper supports providing a *language bias* to influence the hypothesis space.

4.1.4.2 Methodology

In this section, describe the problem of learning chess tactics as an instance of strategy synthesis for chess. We describe our formal model for chess tactics using first-order logic, and describe our method for learning them using ILP. Finally, we describe the metrics that we use to measure a tactic's utility as part of our learning method.

4.1.4.2.1 ILP for Tactic Learning Given the definitions of a strategy and a chess strategy model, we formalize the problem of learning a chess tactic model as an ILP problem $\langle E^+, E^-, B \rangle$. E^+ is a set of $\langle$position, move$\rangle$ tuples where the move made in the position is drawn from a target policy. E^- is a set of $\langle$position, move$\rangle$ tuples where the move made in the position is *not* drawn from the target policy. B is the predicate vocabulary used to express positions, moves and chess tactics. We want to learn a hypothesis H that maximizes the number of examples entailed in E^+ and minimizes the number of examples entailed in E^-. The hypothesis H is our chess strategy ς.

Since we might not be able to find a single hypothesis that meets our objectives, we instead learn a collection of tactics and evaluate their usefulness using the metrics of *coverage* and *divergence*.

Given our tactic model being a first-order logic rule, we model the problem of learning it as an ILP problem. We use training examples drawn from real games played by humans online. Each training example is a $\langle$position, move$\rangle$ tuple, where position is the board state converted to first-order logic using a hand-engineered predicate vocabulary, and move is the move made in that position, also represented in first-order logic. We define this predicate vocabulary in the background knowledge, along with predicates representing the board state and relationships between squares. Our choice of predicates is motivated by the ability to use them to express tactics from chess literature, like the pin or the fork. We also introduce a foreign predicate implemented externally to represent a legal move. See Figure 4.6 for an expression of the *fork* tactic from chess literature in this model. Our background knowledge design borrows from that of the PAL system (Morales 1992). We refer readers to the Appendix Cfor the complete list of predicates in our background knowledge.

Given this formulation of the tactic-learning problem as an ILP problem, we select Popper as the ILP system to learn tactics with. Popper searches for the hypothesis that maximizes the F1 score when evaluated against the examples. However, we wish to learn *multiple* tactics that might not cover the entire example set. Our proposed system supports learning multiple tactics. We do so by modifying the *generate* and *constrain* stages of the Popper ILP system in the following ways —

- *generate*: modified to only generate tactics that produce legal moves
- *constrain*: prevent further *specializations* of a tactic that does not match any position in the training set from being generated

```
fork(Position,From,To) ←
    make_move(From, To, Position, NewPosition),
    attacks(To, Square1, NewPosition),
    attacks(To, Square2, NewPosition),
    different_pos(Square1, Square2),
```

Figure 4.6: An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary. The first `attacks` clause states that the piece at `To` attacks the opposing piece at `Square1` in the current position.

4.1.4.3 Evaluation

In this section, we describe the procedure used to evaluate the tactics produced by our learning system. We describe our testing and training datasets, how we generate the tactics used for evaluation, and the metrics we use to measure the performance of the learned tactics. Since we do not have a reference set of existing tactics expressed in our predicate vocabulary to compare against, we choose to compare against the following baseline tactics —

- **random move tactic**: makes a random legal move in a given position, and is applicable to all positions
- **ground move tactic**: replicates the move made in the ground truth example, and is applicable to all positions
- **engine move tactic**: makes the best move suggested by an engine, and is applicable to all positions. We use two engines - Stockfish 14 and Maia-1600.

Hypothesis 1. *The average divergence of the tactics learned by the system is lower than that of a tactic that makes random moves.*

4.1.4.3.1 Dataset To source our ⟨position, move⟩ training examples, we use a collection of games played by human players on the Internet chess server Lichess[3]. Specifically, we use the

[3]https://lichess.org/

January 2013 archive of standard (played with regular chess rules) rated (users stand to gain or lose rating points based on the outcome) games played on the website (lichess.org 2021). The archive consists of 121,332 games, with players of ELO rating 1601 ± 289 (new players start at 1500 (Lichess 2021)). To generate N examples, we randomly sample N games and select a single random position from that game, along with the move made in that position. We select positions beginning from move 12 (Guid and Bratko 2006; Romero 2019), and exclude games which did not end normally or by time forfeit. Using this procedure, we generated a dataset of 100 examples. We found that using more training examples did not produce new tactics since the hypothesis space was exhausted. For testing, we use the February 2013 archive of 123,961 games and ELO rating 1595 ± 298 to generate 10 testing examples.

4.1.4.3.2 Training Using the bias settings provided by Popper, we limit the size of the learnable hypotheses to a maximum of one clause, five variables and five body literals. Empirically, we find that this strikes a good balance between learning time and quality of learned tactics. We run our proposed method until no more solutions are found. We obtain a list T of 837 tactics.

4.1.4.3.3 Performance Metrics To measure the performance of our tactics on the test data, we use the metrics of *accuracy*, the percentage of moves suggested by a tactic that matches the move in the test data, and *coverage* and *divergence*, which are defined in Equations (4.2) and (4.6) respectively. To provide the evaluation function for calculating divergence, we use the engines Maia-1600 and Stockfish 14. The Maia-1600 engine has been trained on games played by players of ELO rating between $[1600-1699]$ and has been shown to resemble human moves (McIlroy-Young et al. 2020). Stockfish 14 is the winner of the TCEC 2020 Championship (Haworth and Hernandez 2021) and is an extremely strong engine. We limit the search depth of both engines to 1-ply. A single move made by a player is 1-ply. This choice follows from the capabilities of our tactic model and the architecture of the Maia family of chess engines in that they do not conduct tree search.

4.1.4.4 Results

Based on the data obtained, we report the results for each of accuracy, coverage and divergence as a histogram with 20 buckets.

From the coverage values in Figure 4.9, we see that all the tactics learned by our system cover 30% - 60% of the test set. We conclude that the tactics learned by our system are moderately likely to be applicable to positions that arise in real games.

From the accuracy histogram in Figure 4.10, we see the tactics learned by our system are only marginally more accurate at predicting moves in the test set than the random baseline.

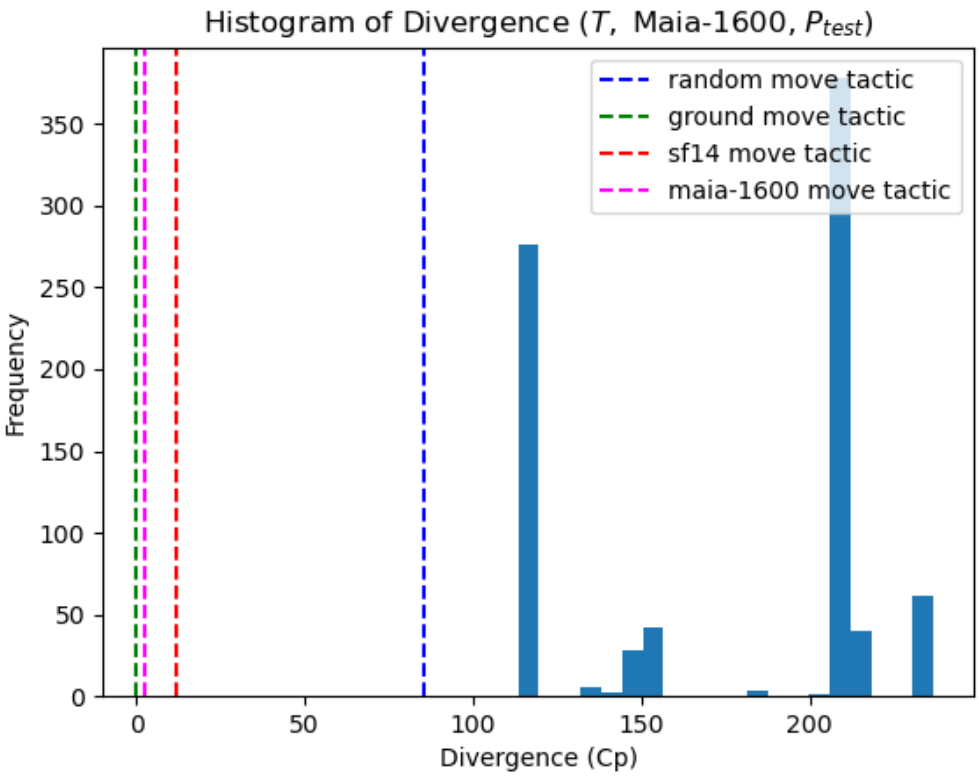

Figure 4.7: Divergence histogram for T evaluated using Maia-1600

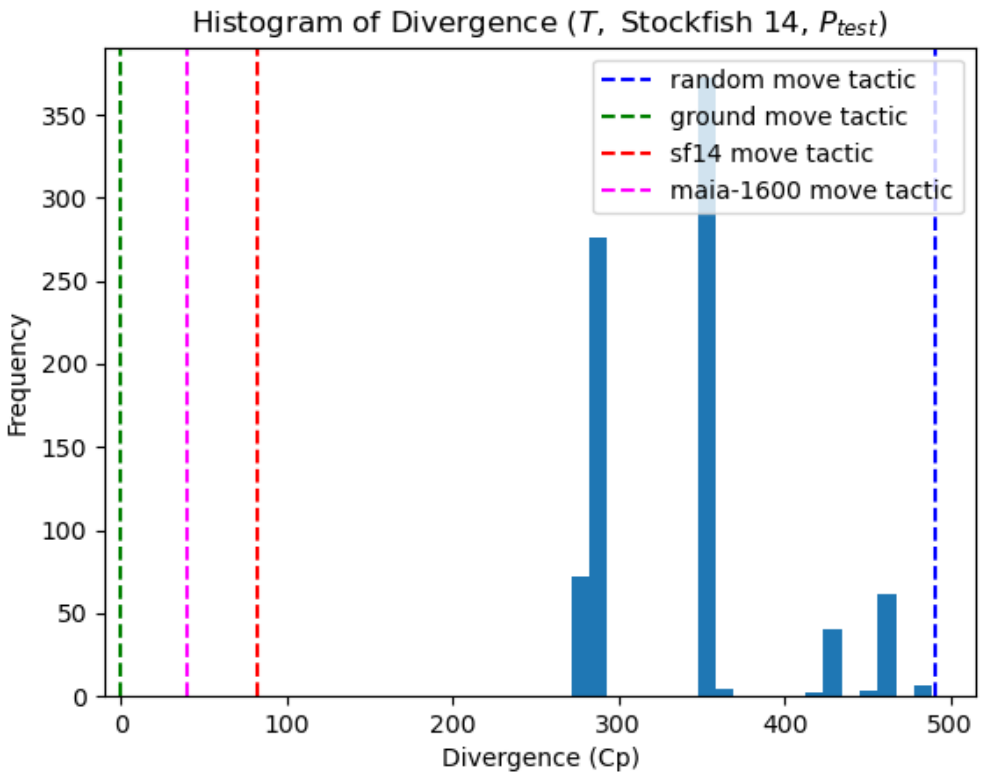

Figure 4.8: Divergence histogram for T evaluated using Stockfish 14

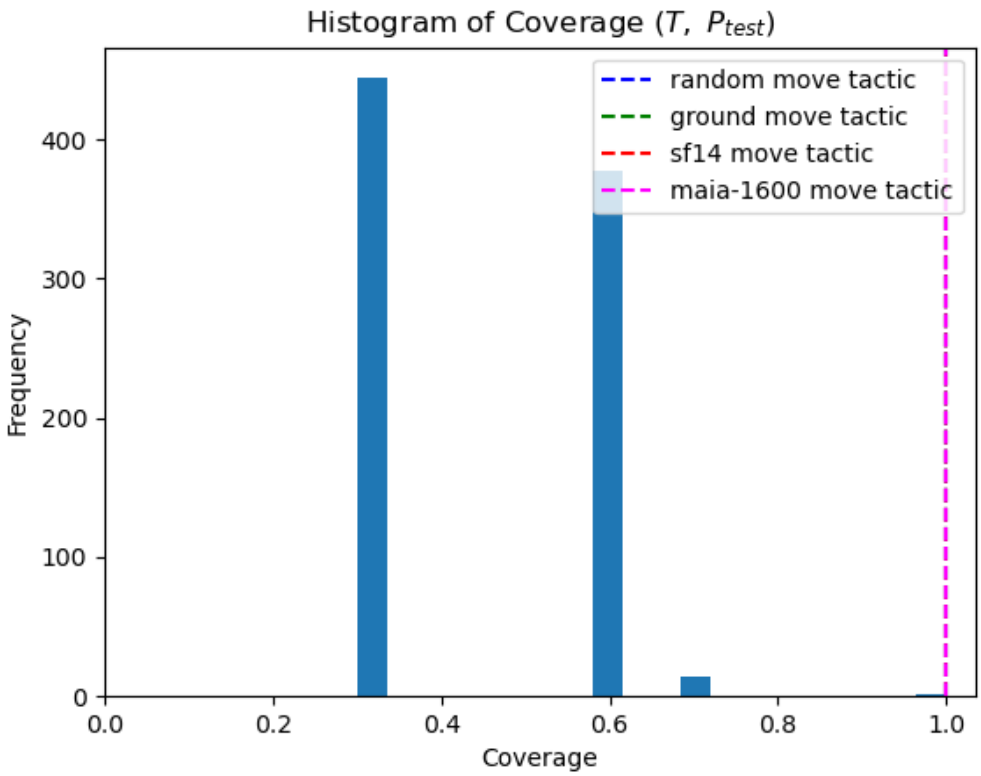

Figure 4.9: Coverage histogram for T

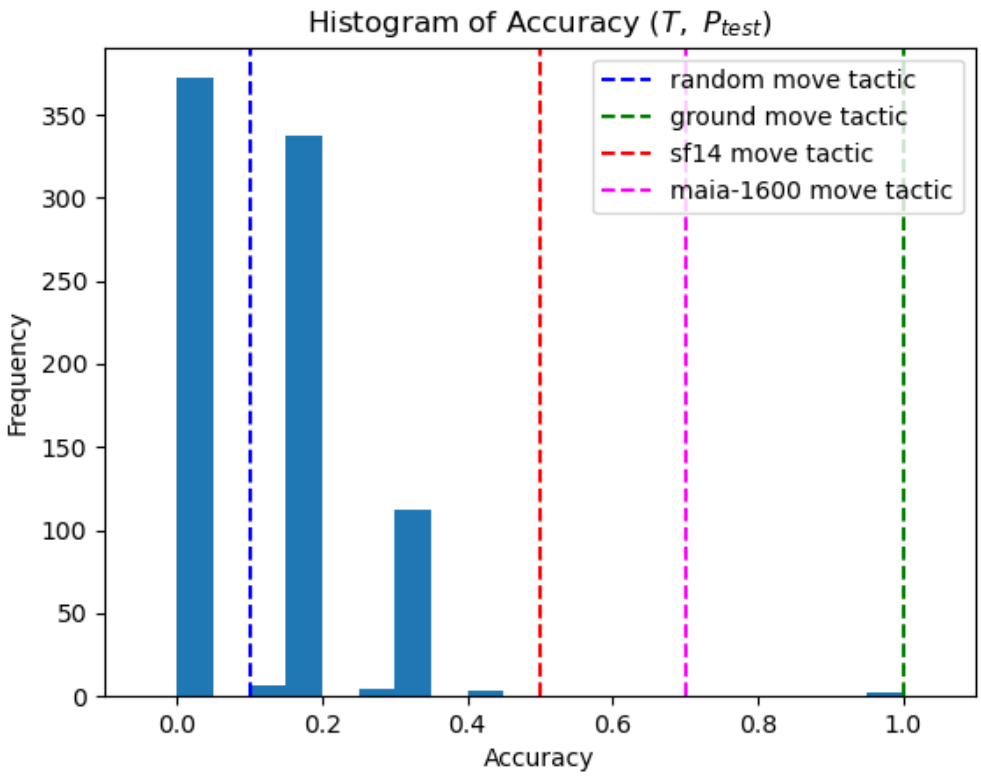

Figure 4.10: Accuracy histogram for T

Overall, the highest accuracy of a learned tactic in T is 42% compared to the 10% accuracy of the random baseline.

From the divergence histogram for T calculated using Maia-1600 (Figure 4.7), we see that Maia-1600 evaluates the learned tactics as having *greater* divergence from ground that the random baseline. However, from Figure 4.8, we see that the stronger Stockfish 14 evaluates the same tactics as having *lower* divergence from ground. We can conclude that the learned tactics are better approximating Stockfish 14's policy as compared to Maia-1600's policy.

4.1.4.4.1 Qualitative Analysis We present three tactics learned by our system to further analyze qualitatively.

4.1.4.4.1.1 Low Divergence The tactic in Figure 4.11 was chosen from those having the least divergence from ground as measured by Stockfish 14. It can be interpreted as follows – if one of your pieces is attacking the opponent's piece (`attacks`(From, Square1, Position)), move it instead to a square which puts it in line with two of your opponent's pieces (as seen in the predicate `behind`(To, Square1, Square2, Position)). To the author's knowledge, this does not represent an existing tactical idea in chess.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  attacks(From,Square1,Position),
  behind(To,Square1,Square2,Position),
  different_pos(Square1,Square2).
```

Figure 4.11: A learned tactic with low divergence from ground as evaluated by Stockfish 14.

4.1.4.4.1.2 Highest Accuracy The tactic in Figure 4.12 was chosen from that having the highest accuracy. It can be interpreted as follows - if one of your pieces is attacking your opponent's piece (`attacks`(From, Square1, Position)), move it instead to a different square. This somewhat resembles the idea of a tactical retreat in chess. It can be thought of as a more general tactic than that in Figure 4.17.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  attacks(From,Square1,Position),
  different_pos(From,To),
  different_pos(From,Square1).
```

Figure 4.12: A learned tactic with the highest accuracy.

4.1.4.4.1.3 Variable Reshuffling The tactic in Figure 4.13 represents a possibility in the tactic hypothesis space that is merely a permutation of the variables in the rule. It is difficult to interpret this in terms of chess, and is very likely meaningless. It points to opportunities to restrict the hypothesis space using language biases to prevent such tactics from being generated.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  behind(From,To,Square1,Position),
  behind(From,Square2,Square3,Position),
  behind(From,Square3,Square2,Position),
  behind(From,Square2,To,Position).
```

Figure 4.13: A learned tactic with meaningless variable reshuffling.

4.1.4.5 Discussion

Since our system uses ILP as its learning method, it inherits many benefits and challenges associated with ILP. Our learning system requires carefully selected language biases in the form of background knowledge in order to learn efficiently (Cropper and Dumančić 2020). Designing predicates suitable to game-like domains will make searching for better tactics more efficient. The learned tactics are also interpretable (Muggleton et al. 2018), and can be added back into the background knowledge to allow lifelong learning (Cropper and Tourret 2020). However, work needs to be done in this area in the context of games.

Our tactics are able to beat a random baseline in producing moves similar to a beginner player. However, they still have high divergence from player moves, have low accuracy and do

not suggest the same moves, indicating that they do not yet adequately represent a human beginner player. Improving the learning algorithm to penalize tactics which do not match the ground truth moves could lead to better results.

Given that our tactics serve as an interpretable model of human players, they could also be used to interpret chess engines, and serve as a general technique for explainable AI. However, further work is required to learn tactics of playing strength comparable to top engines.

discuss applicability to other domains In order to apply our work to another game domain, one would need to engineer a set of Prolog-based features rich enough to express a single-move policy. Due to its discrete, grid-like and turn-based nature, chess lends itself to representation using Prolog more easily than other games. Thus, it was easy to write features to express chess tactics since there existed a wealth of tactics from extant literature. For other game domains, it would be helpful to review existing strategies developed by the player community and design features to express them in Prolog. The features involved in expressing known tactics will hopefully prove general enough to learn effective unknown tactics as well. For new games which do not have an extensive knowledge base of known tactics, it would be necessary to design features based on the rules of the game. The combination of rules would become the features used to learn new tactics.

talk about ILP limitations where a compact, generalizing hypothesis may not exist - affects interpretability The interpretability of first-order logic rules is not a given, and particularly long rules involving predicates with many variables cannot be considered "interpretable" by any measure. The use of an interpretability score which models how interpretable a human player would find a tactic can be used to optimize tactics for both divergence and interpretability. Further work is needed to identify tactic features that contribute to interpretability and design a computational model for calculating an interpretability score.

a position-move pair (p,m) come from human-played games yet its evaluation E(p,m) comes from an AI chess engine. What if these two sources are poorly aligned? What if the humans played differently from how the AI chess engine would play? We use a reference engine to provide the evaluation scores for a given move. If the reference engine and the target policy are not aligned, it would mean that moves are considered to be similar as per the divergence metric are perhaps not actually similar. In the Results section, we saw that Maia-1600 gave poorer divergence scores for the tactic moves as compared to Stockfish 14. It is possible that Stockfish 14, with its greater playing strength, sees less of a difference between the ground and tactic moves since they are both equally bad, or can be equally good with superhuman play. It is necessary to investigate further the effects of reference engine mis-alignment with the ground policy that we are trying to approximate with tactics.

4.1.4.6 Conclusion

We have presented the problem of learning chess tactics in the form of first-order logic rules from examples. We have presented a learning method that uses ILP to learn these tactics. We have evaluated this system and shown that the tactics learned cover a large portion of the test set. Using the metrics of divergence, we showed that they can approximate a human beginner better than a random baseline. We discussed limitations of our method and concluded that while it is promising, further work is required to learn tactics of acceptable playing strength.

4.1.5 Improving Tactic Learning using Precision and Recall

In an extension to the work in Section §4.1.4, I present an improvement to the strategy learning approach for chess using ILP. I present a new vocabulary for chess tactics that implements the full range of basic chess rules as first-order logic predicates. I formulate new constraints for the learning system based on precision and recall to help reduce the hypothesis space of chess tactics. Using an ablation study, I show that this approach is able to learn rules that are correct, provide sufficient coverage, and minimize divergence from expert recommendations. I discuss some limitations of our approach and conclude with directions for future work [4]. I thus answer RQ2(c).

4.1.5.1 Methodology

In this section, I describe the additional constraints provided to the learning system based on precision and recall. I detail a proof of why the recall constraint is correct. I provide a description of our new predicate vocabulary for describing chess tactics.

4.1.5.1.1 Constraints Based on Precision and Recall We propose new constraints to limit the size of the hypothesis space during generation based on precision and recall. We formally prove that a recall constraint will always improve recall of the learned hypotheses. The $\preceq$ operator in our proofs indicates *theory subsumption* as defined in Cropper and Morel (2021). $H_1 \preceq H_2$ indicates that all the examples entailed by hypothesis H_1 are also entailed by H_2.

Definition 1 (Precision constraint)**.** *A precision constraint prunes the specializations of a hypothesis if its precision on a set of examples is less than some pre-defined lower limit.*

A precision constraint cannot be shown to prune hypotheses with less precision than the current one. We add it to investigate its effect on the learned tactics.

[4] The code release for this work is available at https://github.com/AbhijeetKrishnan/interpretable-chess-tactics

Definition 2 (Recall constraint)**.** *A recall constraint prunes specializations of a hypothesis if its recall on a set of examples is less than some pre-defined lower limit.*

Theorem 1. *Given hypotheses $H_1, H_2 \in \mathbb{H}$ with $H_1 \preceq H_2$ and having recall values of r_1 and r_2 on a training set respectively, then $r_1 \leq r_2$.*

Proof.

$$\text{Recall} = \frac{tp}{tp + fn} \tag{4.7}$$

$$H_1 \preceq H_2 \tag{4.8}$$

$$r_1 = \frac{tp_1}{tp_1 + fn_1} \tag{4.9}$$

$$r_2 = \frac{tp_2}{tp_2 + fn_2} \tag{4.10}$$

By the definition of theory subsumption, we have

$$tp_1 \leq tp_2, \text{ and} \tag{4.11}$$

$$fp_1 \leq fp_2 \tag{4.12}$$

Since the number of positive examples is constant for the same population,

$$\therefore r_1 \leq r_2 \tag{4.13}$$

□

4.1.5.1.2 Predicate Vocabulary We contribute a new predicate vocabulary for expressing chess tactics written in Prolog. Our new predicate vocabulary allows us to express tactics involving more situational rules of chess, such as en passant and pawn promotion. It improves the efficiency of tactic unification by removing the need for relying on a foreign predicate for checking legal moves. Figure 4.14 shows an interpretation of the *fork* expressed using our predicate vocabulary. Our predicate vocabulary consists of 25 predicates in total. These do not include the supporting rules required to construct these predicates since those are not allowed to be used in our tactic model. A listing of all predicates used in our vocabulary along with their descriptions can be found in Appendix C.

```
fork(Position,Move) ←
        legal_move(Position,Move),
        move(Move,_,To,_),
        make_move(Position,Move,NewPosition),
        can_capture(NewPosition,To,ForkSquare1),
        can_capture(NewPosition,To,ForkSquare2),
        different(ForkSquare1,ForkSquare2).
```

Figure 4.14: An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary.

4.1.5.2 Evaluation

In this section, we describe the procedure used to evaluate the improvements made to our tactic learning system.

Hypothesis 2. *The average divergence of the tactics learned by the system including the precision and recall-based constraints and the new predicate vocabulary is lower than the average divergence of the tactics learned by the old system.*

We use the same experimental setup for learning tactics as in our previous work. We perform an ablation study to test the effect of the new predicate vocabulary, precision constraint and recall constraint on the tactics learned. We learn tactics for the following systems —

- T_{old}: the system from our previous work
- T_{none}: the new system described in this paper, with the new predicate vocabulary but without the precision and recall constraints
- T_{prec}: the new system with the new predicate vocabulary and only the precision constraint (with a lower limit of 0.1)
- T_{rec}: the new system with the new predicate vocabulary and only the recall constraint (with a lower limit of 0.1)
- T_{new}: the new system (with a lower limit of precision and recall of 0.1)

To refute the hypothesis, we perform a one-sided Welch's t-test using the divergence values of the learned tactics.

4.1.5.2.1 Dataset We use the same procedure as in Section §4.1.4.3.1 to source our train and test data. For positive examples, we select a single random position from each game and the move made in that position. For negative examples, we augment the dataset with the same position and the top-3 choices of Maia-1600 that do not include the ground truth move. Using this procedure, we generated a dataset of 1297 training examples, of which 488 were positive and the rest negative, and 100 testing examples.

4.1.5.2.2 Training We use the same training hyperparameters as in Section §4.1.4.3.2.

4.1.5.2.3 Performance Metrics We use the same performance metrics as in Section §4.1.4.3.3. We also compare them against the following baseline tactics —

- **random move tactic**: makes a random legal move in a given position, and is applicable to all positions
- **ground move tactic**: replicates the move made in the ground truth example, and is applicable to all positions
- **engine move tactic**: makes the best move suggested by an engine, and is applicable to all positions. We use two engines – Stockfish 14 and Maia-1600.

4.1.5.3 Results and Analysis

	Size	**Accuracy**	**Coverage**	**Maia-1600**		**Stockfish 14**	
				Divergence	p	**Divergence**	p
T_{old}	837	0.12 ± 0.13	0.45 ± 0.16	174.48 ± 48.69	-	337.56 ± 57.98	-
T_{none}	141	0.34 ± 0.24	0.26 ± 0.35	103.28 ± 63.67	< 0.01	268.45 ± 131.67	< 0.01
T_{prec}	511	0.32 ± 0.23	0.21 ± 0.29	276.59 ± 147.13	1.0	276.59 ± 147.13	< 0.01
T_{rec}	387	0.10 ± 0.12	0.52 ± 0.41	143.27 ± 161.61	< 0.01	467.78 ± 138.69	1.0
T_{new}	284	0.11 ± 0.13	0.54 ± 0.41	143.57 ± 167.53	< 0.01	473.31 ± 151.90	1.0

Table 4.3: Summary statistics for the tactics learned by each system in the ablation study.

Based on the data obtained, we report boxplots for the tactics obtained from each of our systems. The summary statistics of the tactics learned by each system is shown in table 4.3.

Based on the results of the one-sided Welch's t-test shown in Table 4.3, we see that every system other than T_{prec} demonstrates a significant ($p < 0.01$) improvement in divergence

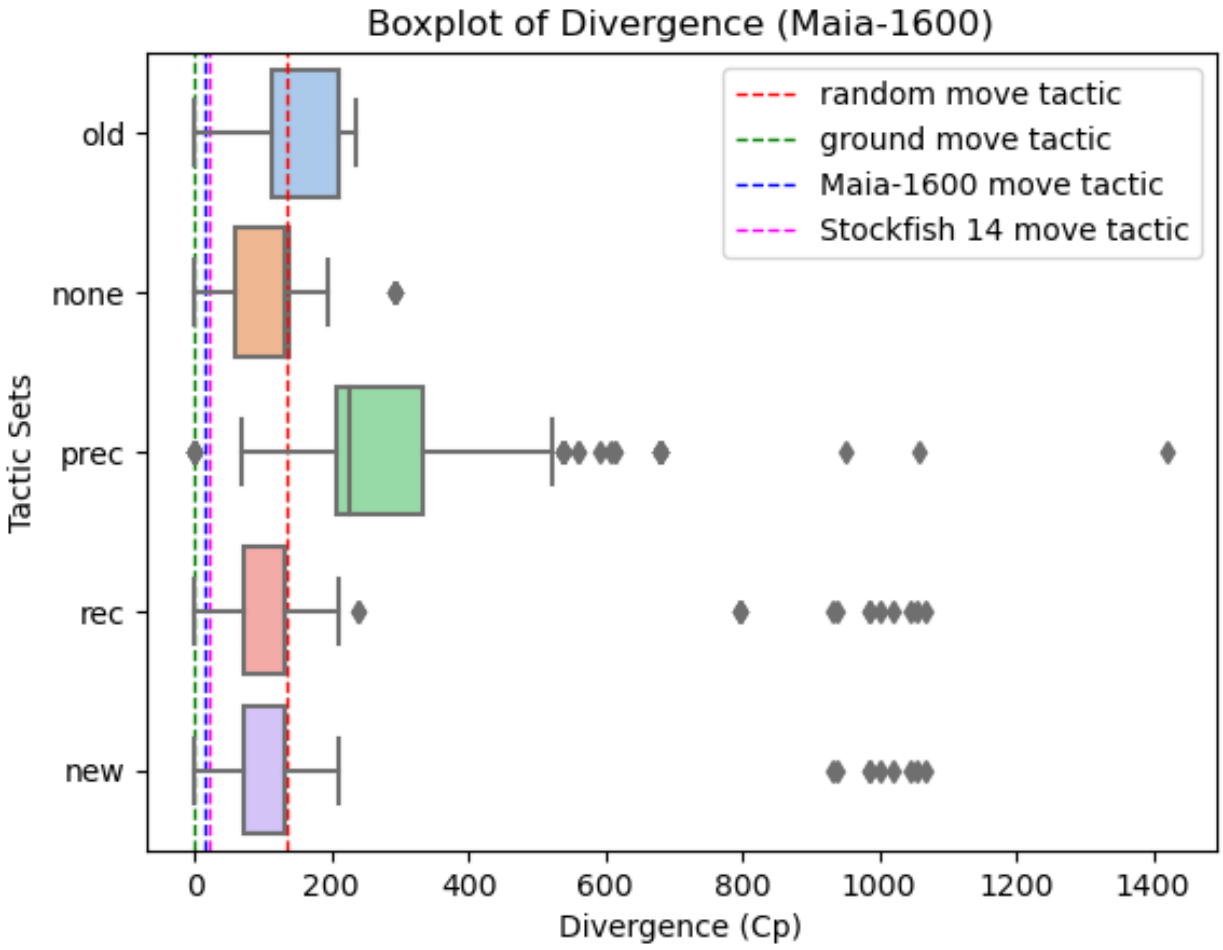

Figure 4.15: Boxplot of tactic divergence (evaluated using Maia-1600) for each system

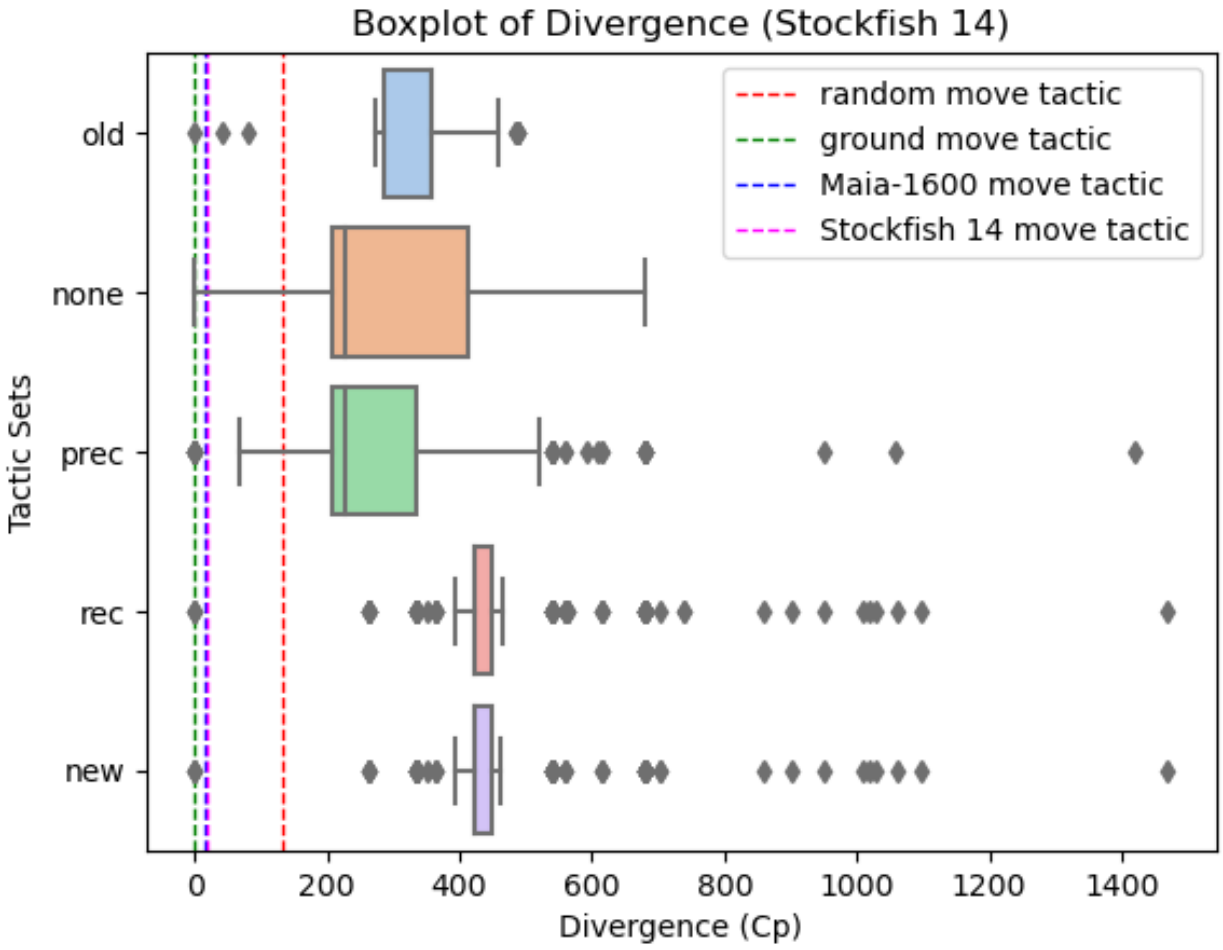

Figure 4.16: Boxplot of tactic divergence (evaluated using Stockfish 14) for each system

over the old system when calculated using Maia-1600. When divergence is calculated using Stockfish 14, only T_{none} and T_{prec} demonstrate a significant improvement. We conclude that the new predicate vocabulary is effective at improving divergence universally, whereas the recall constraint is effective at improving divergence when calculated using a weaker engine.

4.1.5.3.1 Qualitative Analysis We present two tactics learned by our system to further analyze qualitatively. These were learned by T_{new}.

4.1.5.3.1.1 Low Divergence The tactic in Figure 4.17 was chosen from those having the least divergence from ground as measured by Maia-1600. It can be interpreted as follows - make the move that allows you to castle in the subsequent position. Castling early is a common piece of advice given to beginner players.

```
f(Board,Move) ←
  legal_move(Board,Move),
  make_move(Board,Move,NewBoard),
  castling_move(NewBoard,NewMove).
```

Figure 4.17: A learned tactic with low divergence from ground as evaluated by Maia-1600.

4.1.5.3.1.2 Highest Accuracy The tactic in Figure Figure 4.18 was chosen from those having the highest accuracy. It can be interpreted as making a move that puts your opponent in check. This tactic does not have high accuracy when evaluated using the stronger Stockfish 14 engine, which reveals a tendency for lower-rated players to tend to make checking moves when available.

```
f(Position,From,To) ←
  legal_move(Board,Move),
  make_move(Board,Move,NewBoard),
  color(Color),
  in_check(NewBoard,Color,CheckingPiece).
```

Figure 4.18: A learned tactic with the highest accuracy.

4.1.5.4 Discussion and Future Work

The contribution of the new predicate vocabulary and precision/recall-based constraints helps our learning system generate better tactics that more closely approximate the ground moves made. Adding constraints based directly on divergence could help the system search for better tactics.

Our system is able to learn tactics that are evaluated favorably by a weak chess engine. Thus, the tactics seem to be effective at low levels of play. However, further work needs to be done to learn tactics that are evaluated favorably by a stronger chess engine. This would help these tactics be effective even at high levels of play.

4.1.5.4.1 Beyond Chess The contribution of a new predicate vocabulary resulted in measurable improvement to the learned tactics for Chess. To replicate this in another game domain, one would need to engineer a set of Prolog-based features rich enough to express a single-move policy. Due to its discrete, grid-like and turn-based nature, chess lends itself to representation using Prolog more easily than other games. For other game domains, it would be helpful to review existing strategies developed by the player community and design features to express them in Prolog. For new games which do not have an extensive knowledge base of known tactics, it would be necessary to design features based on the rules of the game.

4.1.5.4.2 Interpretability The interpretability of first-order logic rules is not a given, and particularly long rules involving predicates with many variables may not be considered "interpretable" to a general audience. The use of an interpretability score which models how interpretable a human player would find a tactic can be used to optimize tactics for both divergence and interpretability. Further work is needed to identify tactic features that contribute to interpretability and design a computational model for calculating an interpretability score.

4.1.5.4.3 Evaluation We use a reference engine to provide the evaluation scores for a given move. If the reference engine and the target policy are not aligned, it would mean that moves are considered to be similar as per the divergence metric may not actually similar. In the Results section, we saw that Stockfish 14 gave poorer divergence scores for the tactic moves as compared to Maia-1600. It is possible that Stockfish 14, with its greater playing strength, sees more of a difference between the ground and tactic moves. It is necessary to investigate further the effects of reference engine misalignment with the ground policy that we are trying to approximate with tactics.

4.1.5.5 Conclusion

We have presented the problem of learning chess tactics in the form of first-order logic rules from examples. We have introduced two significant improvements to a previous tactic-learning system in the form of a) a new predicate vocabulary, and b) constraints based on precision and recall. We have presented a novel evaluation method for these improvements using an ablation study to measure their effects on the divergence of the learned tactics from a target policy of beginner human players. We showed that the new predicate vocabulary and recall-based constraint contributed significantly to reducing the divergence, whereas the precision-based constraint did not. We discussed limitations of the improvements made and suggested future improvements in the form of new divergence-based constraints.

4.2 MicroRTS

MicroRTS is a small implementation of a real-time strategy (RTS) game, designed to perform AI research (Ontanon 2021). RTS games can be viewed as simplified military simulations, where several players struggle over resources scattered over a 2D terrain by setting up an economy, building armies, and guiding them into battle in real-time (Buro 2003). Their roots date back to 1989, with the launch of the game *Herzog Zwei* (Sega Enterprises Ltd.) on the Sega Genesis (Geryk 2001). They are now among the most popular esport genres, and top tournaments regularly feature prize pools upwards of $1M USD (Lieberman 2022).

Figure 4.19: A screenshot from the game 0 A.D., showing typical RTS interface elements such as a resource overview (top left), a map of the game world (bottom left), and a description of the selected unit (bottom center).

In comparison to chess, MicroRTS is real-time, and partially observable. A description of its game state is shown in Figure 4.20. It has an active research community around it with robust software implementations in Java (Ontanon 2021) and Python (Huang et al. 2021). A MicroRTS AI competition has been organized since 2017 with the goal of producing an agent that has the highest win rate (Ontañón et al. 2018). Due to a lack of a significant player base, there are not as many documented strategies that human players use for MicroRTS, with most hard-coded agent scripts written to imitate strategies found in other RTS games. However, there is a growing body of research that attempts to synthesize programmatic strategies for MicroRTS (Mariño et al. 2021; Medeiros et al. 2022). This effort was consolidated into the Synthesis of Programmatic Strategies (SynProS) track for the MicroRTS AI Competition (Moraes 2021a). Section §4.2.1 details the strategy model for a MicroRTS strategy that is used in the competition, that I will adopt for the proposed work in MicroRTS.

In my proposed work, I will investigate an approach to synthesize programmatic strategies for MicroRTS that follows from existing work done in the domain, and leverages my prior experience with logic programming. I detail this approach in Section §4.2.3 and expect it to answer RQ3(a). Due to a dearth of research investigating the interpretability of the synthesized programmatic strategies, I propose to investigate the factors affecting their interpretability by conducting a user study. I detail the design of this proposed study in Section §4.2.4, and expect

it to answer RQ3(b).

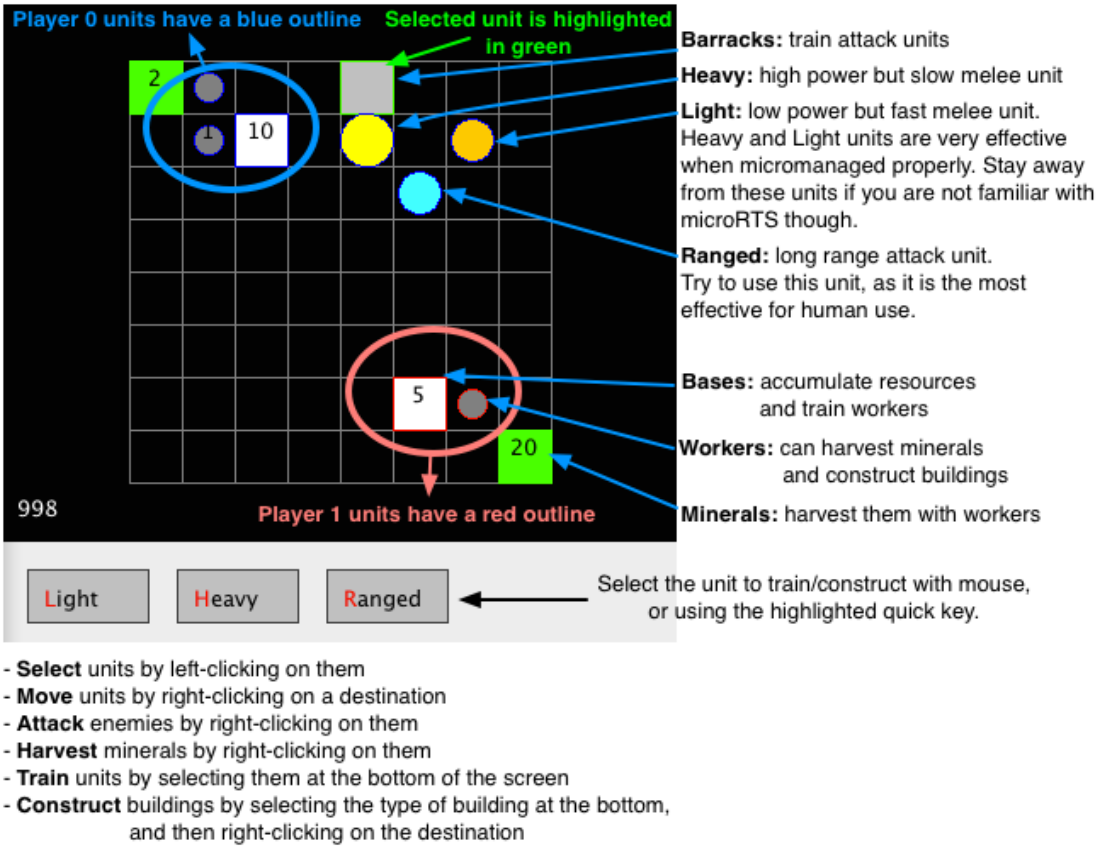

Figure 4.20: A description of the MicroRTS game state taken from the project's README on GitHub.

4.2.1 MicroRTS Strategy Model

Mariño et al. (2021) introduced a domain-specific language (DSL) for MicroRTS scripts. A program in this DSL represents a controller for a MicroRTS agent. The ISS problem is then to synthesize a program in this DSL that meets the performance and interpretability measures defined for the domain.

The DSL can be described as a context-free grammar (CFG), as shown in Figure 4.21, where S_1 is the start symbol. The DSL accepts programs with branching and loops, and it implements a set of functions that assign actions to units in the game, such as "attack the closest enemy", which are denoted by $c_i, i \in [1, n]$. The DSL also implements Boolean functions such as "do I control 2 ranged units?", which are denoted by $b_j, j \in [1, m]$. A detailed description of the

functions implemented in the DSL can be found in MicroRTS's Wiki page (Moraes 2021b). A sample script in this DSL is shown in Figure 4.22.

$$
\begin{aligned}
&S_1 \rightarrow C\ S_1 \mid S_2\ S_1 \mid S_3\ S_1 \mid \epsilon \\
&S_2 \rightarrow \text{if } (S_5) \text{ then } \{C\} \mid \text{if } (S_5) \text{ then } \{C\} \text{ else } \{C\} \\
&S_3 \rightarrow \text{for (each unit } u) \{S_4\} \\
&S_4 \rightarrow C\ S_4 \mid S_2\ S_4 \mid \epsilon \\
&S_5 \rightarrow \text{not } B \mid B \\
&B \rightarrow b_1 \mid b_2 \mid \cdots \mid b_m \\
&C \rightarrow c_1\ C \mid c_2\ C \mid \cdots \mid c_n\ C \mid c_1 \mid c_2 \mid \cdots \mid c_n \mid \epsilon
\end{aligned}
$$

Figure 4.21: The production rules of a context-free grammar describing the DSL for MicroRTS.

```
for(u):
    if not HaveUnitsStrongest(Heavy,u):
        harvest(1,u)
for(u):
    if not HaveQtdEnemiesbyType(Heavy,9):
        attack(Worker,closest,u)
    if not HaveUnitsStrongest(Ranged,u):
        moveaway(Worker,u)
        train(Worker,4,Right)
```

Figure 4.22: A sample script in the MicroRTS DSL, lightly formatted to resemble Python.

4.2.2 Metrics to Evaluate MicroRTS Strategy Models

Following the SynProS competition, I propose to use the *win rate* of a MicroRTS strategy on a map as a measure of its performance. It is computed by entering a strategy in a round-robin tourament with the baseline strategies used in the MicroRTS AI competition, namely

RandomBiased, POWorkerRush, POLightRush and NaiveMCTS. The win rate is a non-zero fractional value between 0 and 1.

As the interpretability measure, I propose to use the notion of the *non-interpretability penalty* defined for the SynProS competition. It approximates the interpretability of a strategy using the size of the program encoding the strategy in terms of the number of instructions, i.e., for-loops, if statements, boolean expressions and commands. It is computed as in Equation (4.15).

$$I^{-1}(\sigma) = \left\{ \begin{array}{ll} 0, & \text{for } 0 \leq |\sigma| \leq 10 \\ 0.01|\sigma| - 0.1, & \text{for } 10 < |\sigma| < 100 \\ 1, & \text{for } 100 \leq |\sigma| \end{array} \right\} \tag{4.14}$$

$$I(\sigma) = 1 - 0.4 I^{-1}(\sigma) \tag{4.15}$$

I^{-1} is the non-interpretability penalty defined in the SynProS competition, I is the interpretability measure introduced in Chapter 3, σ is a MicroRTS strategy represented using the model in Section §4.2.1, and $|\sigma|$ is the number of instructions in the strategy.

4.2.3 MicroRTS Strategy Synthesis using Answer Set Programming

I propose to investigate an approach to the problem of ISS for MicroRTS that is based on answer set programming (ASP). I detail this proposed approach in Section §4.2.3.2, and describe how I will evaluate it in Section §4.2.3.3. From this investigation, I expect to answer RQ3(a).

4.2.3.1 Background

Include program synthesis as a related field here?

4.2.3.1.1 Answer Set Programming Answer Set Programming (ASP) is a declarative programming paradigm based on the semantics of *stable models* (Gelfond and Lifschitz 1988). ASP programs consist of a set of rules which allow facts to be *derived* from them. The language of ASP (Gebser et al. 2015) introduces powerful modeling constructs which allow using rules to rapidly specify a design space, and restrict it using integrity constraints. ASP has been used in the context of games to generate levels (Smith and Mateas 2011; Smith et al. 2012). Figure 4.23 is a reproduction of Figure 8.7 from Shaker et al. (2016) that lists and ASP program which can generate maze-like levels with the desired number of game objects.

```
#const width=10.

param("width",width).

dim(1..width).

tile((X,Y)) :- dim(X), dim(Y).

adj((X1,Y1),(X2,Y2)) :- tile((X1,Y1)), tile((X2,Y2)), \\
    #abs(X1-X2)+#abs(Y1-Y2) == 1.

start((1,1)). finish((width,width)).

% tiles have at most one named sprite
0 { sprite(T,wall;gem;altar) } 1 :- tile(T).

% there is exactly one altar and one gem in the whole level
:- not 1 { sprite(T,altar) } 1. :- not 1 { sprite(T,gem) } 1.
```

Figure 4.23: An ASP program which can generate maze-like levels with integrity constraints that specify the number of game objects.

4.2.3.1.2 Draco: Formalizing Visualization Design Knowledge as Constraints Draco is a system introduced by Moritz et al. (2018) that uses ASP-based constraints to model data visualization knowledge. Data visualizations were modeled using a visualization-specification language called Vega-Lite (Satyanarayan et al. 2016). Using visualization preference data gathered from user studies, a linear regression model was built to predict user preference scores given the representation of the visualization in Vega-Lite. The design space of possible visualizations was modeled in ASP, and the regression function was translated to ASP to generate new visualizations that tried to optimize the regression function, thus maximizing their predicted appeal.

4.2.3.2 Methodology

I propose replicating the approach taken by the Draco system to synthesize interpretable programmatic strategies for MicroRTS. I will first approach the problem of predicting a strategy's win rate by assembling a dataset of strategies and their win rates, and training a simple linear regression model on it. The strategies will be synthesized using existing synthesizers due to Mariño et al. (2021) and Medeiros et al. (2022). Then, I will model the space of MicroRTS strategies in ASP. This will be relatively straightforward since it is already defined as a CFG. I

will then engineer features of each strategy, such as program size, number of for-loops and so on, and use ASP to optimize the linear regression function in order to synthesize MicroRTS strategies that have a high win rate.

4.2.3.3 Evaluation

The evaluation of the learned strategies will be conducted in a similar manner to the framework described by the SynProS competition. In addition to the baseline agents used, I will also compare the ASP-based strategy synthesizer to existing strategy synthesizers, as well as to top-performing bots in the MicroRTS AI competition, namely COAC, Mayari, MentalSeal, UmSbot, Droplet, Rojo and Alet. My hypothesis is that the scripts learned by the Draco-based system will achieve a higher win rate on the test maps compared to the other synthesizers.

I will also measure the interpretability of the synthesized strategies using the interpretability measure used in the SynProS competition, and conduct a qualitative analysis of them.

4.2.4 Factors Affecting MicroRTS Strategy Interpretability

While a lot of research has focused on the synthesis portion of ISS, there is a lack of work studying the interpretability of the synthesized strategies. I propose to run an exploratory user study that will use a set of pre-synthesized MicroRTS strategies that vary along various program factors as detailed in Section §4.2.3.2, and ask participants to rate the interpretability of a strategy using a survey. Finding which factors most correlate with interpretability will allow future researchers to develop methods that optimize for those factors when synthesizing strategies. I describe the study design which will allow me to measure perceived interpretability in Section §4.2.4.2. I expect the results of this investigation to answer RQ3(b).

4.2.4.1 Background

4.2.4.1.1 Interpretability in Explainable AI Artificial intelligence has become increasingly ubiquitous in our daily lives, performing a wide range of functions from suggesting corrections to our writing, to populating our social media feed with new posts, and even deciding whether our resume for a job application should move on to the next stage. Given the weight of the decisions they are now entrusted with, there is a growing realization that we, as humans, should have some insight into how these algorithms make the decisions they do. This is further complicated by the sheer complexity of modern artificial intelligence models, with the latest neural networks boasting upwards of a million parameters.

The field of explainable artificial intelligence (XAI) attempts to investigate methods and techniques to peek inside the black box of modern artificial intelligence models. Adadi and

Berrada (2018) provide one of the first comprehensive surveys on the field of XAI. Tjoa and Guan (2020) survey XAI work from the lens medical research. Zhang et al. (2021) provide a recent survey on interpretability techniques specifically for neural network-based models.

As mentioned in Chapter 2, there has been a similar lack of precision in the definition and measurement of "interpretability" as it pertains to explaining the decisions made by machine learning models.

4.2.4.2 Methodology

Borrowing from the taxonomy presented in Doshi-Velez and Kim (2017), I propose to use a *human-grounded* evaluation involving a simplified task to quantify the interpretability of a MicroRTS strategy. Specifically, I will use a forward simulation/prediction task where I present subjects with a strategy, a game state, and four options for future states, and ask them to respond with which future state they think will be reached upon executing the given strategy from the given game state. The strategies for this study will be synthesized using the ASP model of the strategy space described in Section §4.2.3.2. I will provide additional constraints to the generation system in order to obtain strategies that vary uniformly across the variables to be studies, such as program size, number of for-loops etc. The current game state will be obtained at random from a gameplay trace of the strategy executing in a game against a baseline opponent. The three incorrect options for future states will be generated by executing a random strategy from the current game state.

4.2.4.3 Evaluation

After completing the study, I will obtain a dataset that contains details of the participants, the strategies, the task used (i.e., the current game state and options for future game states), and the provided and correct responses. I will train a decision tree model on the binary classification task of predicting whether a strategy will be correctly simulated. I will measure the importance of a strategy feature by going through all the splits for which the feature was used and measuring how much it has reduced the variance or Gini index compared to the parent node (Molnar 2018).

CHAPTER

5

PROPOSED RESEARCH

In this chapter, I will detail my plans for completing the research work proposed in this document which serve to answer the RQs listed in Section §1.2. In the case of completed work, I will cite the paper detailing that work. In the case of proposed work, I will describe a plan to complete the various components of that research necessary for answering the underlying RQ, and submitting to a suitable, peer-reviewed venue. I will also provide an estimated timeline for these research milestones and PhD degree milestones.

5.1 Research Plan

RQ1. How do we formally define the problem of ISS?

The details for my proposed definition and framework for the problem of ISS are provided in Chapter 3. This work was published in Krishnan and Martens (2022b) at the Explainable Agency in Artificial Intelligence Workshop held during the 36th AAAI Conference on Artificial Intelligence.

RQ2. How do we approach the problem of ISS for the game of chess?

This broader RQ is answered by the more specific RQs below.

RQ2(a) Could we represent known chess tactics as a learnable, computational model of a chess-playing policy and develop metrics to show that they suggest better moves than a random baseline?

In Sections §4.1.1, §4.1.2 and §4.1.3, I show how we may utilize the framework defined in Chapter 3 to computationally model a chess strategy and translate known chess tactics into the model. I show that these tactics produce better moves on average compared to a random player, thus answering RQ2(a) and contributing towards RQ2. This work was published in Krishnan and Martens (2022b) at the Explainable Agency in Artificial Intelligence Workshop held during the 36th AAAI Conference on Artificial Intelligence.

RQ2(b) Do the chess strategies learned using ILP outperform a random baseline in how closely their divergence scores approximate a beginner player?

In Section §4.1.4, I show how we can use inductive logic programming (ILP), a symbolic machine learning (ML) method, to serve as a learning algorithm for chess strategies. This work was published in Krishnan and Martens (2022a) at the Workshop on Artificial Intelligence for Strategy Games (SG) and Esports Analytics (EA) held during the 18th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.

RQ2(c) Is the average divergence of the strategies learned by an ILP system incorporating the changes of the new predicate vocabulary and precision/recall-based constraints lower than of that learned by an ILP system without these modifications?

In Section §4.1.5, I show how modifications to the learning algorithm can be made to improve the quality of the strategies it learns. This work has been submitted to a peer-reviewed conference and is currently under review.

RQ3. How do we approach the problem of ISS for the game of MicroRTS?

This broader RQ is answered by the more specific RQs below.

RQ3(a) Does an answer-set programming based approach towards developing a synthesizer for the SynProS competition achieve the highest overall score across all test maps?

In Section §4.2.3, I detail an approach that uses answer-set programming to design a synthesizer that can search for scripts that achieve a high win rate. The key components of this approach are —

- **ML Framework to predict the win rate of a strategy**: This component involves designing a linear regression model to predict the win rate of a particular MicroRTS strategy given its representation as a set of program features. I will

assemble a dataset of MicroRTS strategies using existing synthesizers as detailed in Section §4.2.3.2. Their win rate will be measured using the evaluation procedure described in Section §4.2.3.3, but using only baseline agents as used in the MicroRTS AI competition.

- **Modeling the space of strategies using ASP**: This component involves writing an ASP program to model the space of MicroRTS strategies. I will develop this by drawing on prior experience modeling chess tactics in Prolog (see Section §4.1.5.1), and developing a level generator in ASP[cite].
- **Constraint Optimization**: This component involves translating the linear regression model into an optimization constraint that will be used in the ASP program for MicroRTS strategies. This should allow the generation of MicroRTS strategies that try to maximize win rate.
- **Evaluation, Paper Writing and Submission**: I plan to submit this work in the research track at the Advances in Computer Games 2023 conference (ACG 2023).

RQ3(b) Which features of a MicroRTS strategy model have significant correlation with the interpretability of said strategy as measured by a user study?

The design for this study is described in Section §4.2.4. Its key components are:

- **Obtaining strategies**: I will collect the strategies that users will interpret by using both existing synthesizers (see Section §4.2.3.2) and the ASP model of a MicroRTS strategy I will develop as part of RQ3(a).
- **Desigining the interpretability task**: My proposed design of the interpretability task requires obtaining game states from a MicroRTS environment by running various agents in it. I will use the MicroRTS Gym framework (Ontañón et al. 2018) to collect the data for this task.

Following this, I will design my study and submit it to the IRB for approval. Then, I will analyze participant responses and write it up as a paper. The planned venue for publication is the as yet unannounced 6th International Workshop on EXplainable and TRAnsparent AI and Multi-Agent Systems (EXTRAAMAS 2024) whose deadline is expected to be in March 2024.

5.2 Proposed Timeline

I present a semester-wise breakdown of the important milestones before my planned thesis defense and graduation.

- Spring 2022
  - **??** Framework (EAAI-22) - *Completed*
  - RQ2(a) Study (EAAI-22) - *Completed*
- Fall 2022
  - RQ2(b) Study (AIIDE-22 SG Workshop) - *Completed*
- Spring 2023
  - RQ2(c) Study (under review) - *Completed*
  - RQ3(a) Dataset Assembly - *In progress*
  - RQ3(a) ASP Modeling
- Summer 2023
  - RQ3(a) Study (ACG-23)
  - RQ3(b) Obtaining Strategies
- Fall 2023
  - RQ3(b) Task Design
  - RQ3(b) Study Approval
- Spring 2024
  - RQ3(b) Analysis (EXTRAAMAS '24)
  - Complete dissertation (early Spring '24)
  - Defend thesis (Mid-late Spring '24)
  - Graduate (May '24)

CHAPTER

6

CONCLUSION

In this document, I have outlined my thesis work, including work I have already performed, as well as what is yet to be done. My goal for this dissertation is to investigate approaches to the problem of interpretable strategy synthesis (ISS) for multiple game domains in order to better understand how to design effective strategy models for games, and which algorithms are most successful to learn them. The knowledge generated from this work will inform existing research and industry in esports analytics that try to generate insights from player data. It will also help the player base uncover new strategies which can be easily communicated and shared.

6.1 Summary

My research for this dissertation is interdisciplinary and draws from XAI research, cognitive modeling and game AI. I will investigate designs for game strategy models that can be best understood by players. I will also investigate and implement algorithms to automatically learn these strategies from gameplay data.

6.2 Expected Contributions

The work proposed for this dissertation has stemmed from research I have completed on the design of interpretable strategy models for the games of chess and MicroRTS, and algorithms to learn them. The expected contributions of this dissertation are as follows —

- *A formal definition of the problem of interpretable strategy synthesis and a framework to describe its components.* While there has been prior work in attempting to solve the problem of ISS, there has yet to be a unified, formal description of the problem in order to help researchers situate their work and reuse models and algorithms developed for other games. This formalization will help provide a common vocabulary for researchers working on the ISS problem.

- *An approach towards solving the ISS problem for chess by using a chess tactic-inspired strategy model, and a learning algorithm based on ILP.* While there has been a lot of research on pattern-learning in chess, none of them attempt to make their learning methods work for games other than chess. The tactic model and ILP-based learning method is available as publically-available software implementations for other researchers to use and apply to different games.

- *An approach towards solving the ISS problem for MicroRTS using ASP and a measurement of the critical factors affecting the interpretability of its strategy model.* While prior work has developed a strategy model for MicroRTS and algorithms to learn it, there has been no attempt to study how interpretable the learned strategies are, or how we might make them better for the purpose of player understanding. This work will provide a software implementation of an ASP-based learning method to derive MicroRTS strategies, as well as a dataset of participant ratings of strategy interpretability which will inform future work in synthesizing interpretable MicroRTS strategies. It will also inform work in XRL that attempts to produce script-based explanations.

BIBLIOGRAPHY

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

Averbakh, Y. (2012). *A history of chess: from Chaturanga to the present day*. SCB Distributors.

Bastani, O., Pu, Y., and Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. *arXiv preprint arXiv:1805.08328*.

Berliner, H. J. (1975). A representation and some mechanisms for a problem solving chess program. Technical report, Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Bijl, P. and Tiet, A. (2021). *Exploring modern chess engine architectures*. PhD thesis, Victoria University, Melbourne.

Boros, E., Elbassioni, K., Gurvich, V., and Makino, K. (2012). On nash equilibria and improvement cycles in pure positional strategies for chess-like and backgammon-like n-person games. *Discrete Mathematics*, 312(4):772–788.

Bramer, M. A. (1977). *Representation of Knowledge for Chess Endgames Towards a Self-Improving System*. PhD thesis, Open University (United Kingdom).

Bratko, I. (1982). Knowledge-based problem-solving in al3. *Machine intelligence*, 10:73–100.

Buntine, W. (1988). Generalized subsumption and its applications to induction and redundancy. *Artificial intelligence*, 36(2):149–176.

Buro, M. (2003). Real-time strategy games: A new ai research challenge. In *IJCAI*, volume 2003, pages 1534–1535.

Butler, E., Torlak, E., and Popović, Z. (2017). Synthesizing interpretable strategies for solving puzzle games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pages 1–10.

Canaan, R., Shen, H., Torrado, R., Togelius, J., Nealen, A., and Menzel, S. (2018). Evolving agents for the hanabi 2018 cig competition. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

Claude, E. S. (1950). Programming a computer for playing chess. *Philosophical Magazine, Ser*, 7(41):314.

Coppens, Y., Efthymiadis, K., Lenaerts, T., Nowé, A., Miller, T., Weber, R., and Magazzeni, D. (2019). Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence*, pages 1–6.

Cropper, A. and Dumančić, S. (2020). Inductive logic programming at 30: a new introduction. *arXiv preprint arXiv:2008.07912*.

Cropper, A. and Dumančić, S. (2022). Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research*, 74:765–850.

Cropper, A. and Morel, R. (2021). Learning programs by learning from failures. *Machine Learning*, 110(4):801–856.

Cropper, A. and Tourret, S. (2020). Logical reduction of metarules. *Machine Learning*, 109(7):1323–1369.

de Freitas, J. M., de Souza, F. R., and Bernardino, H. S. (2018). Evolving controllers for mario ai using grammar-based genetic programming. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

de Mesentier Silva, F., Isaksen, A., Togelius, J., and Nealen, A. (2016). Generating heuristics for novice players. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

Deutsch, M. 2017. My month-long quest to become a chess master from scratch. Medium. https://medium.com/@maxdeutsch/my-month-long-quest-to-become-a-chess-master-from-scratch-51ff8003d3f2 (Accessed: 2023-03-21).

Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning.

Esports, S. 2023. Shadow | game-changing esports analytics - shadow. https://shadow.gg (Accessed: 2023-03-21).

Evans, R. and Grefenstette, E. (2018). Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64.

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., and Schaub, T. (2015). Abstract gringo. *Theory and Practice of Logic Programming*, 15(4-5):449–463.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA.

Geryk, B. 2001. A history of real-time strategy games. Gamespot. Archived from the original on April 27, 2011., https://web.archive.org/web/20110427052656/http://gamespot.com/gamespot/features/all/real_time/ (Accessed: 2023-03-26).

Gobet, F. and Jansen, P. J. (2006). Training in chess: A scientific approach. *Education and chess*.

Guid, M. and Bratko, I. (2006). Computer analysis of world chess champions. *ICGA journal*, 29(2):65–73.

Guid, M. and Bratko, I. (2011). Using heuristic-search based engines for estimating human skill at chess. *ICGA journal*, 34(2):71–81.

Guid, M. and Bratko, I. (2017). Influence of search depth on position evaluation. In *Advances in computer games*, pages 115–126. Springer.

Haworth, G. and Hernandez, N. (2021). The 20th top chess engine championship, tcec20. *J. Int. Comput. Games Assoc.*, 43(1):62–73.

Hayes, B. and Shah, J. A. (2017). Improving robot controller transparency through autonomous policy explanation. In *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, pages 303–312.

Huang, S., Ontañón, S., Bamford, C., and Grela, L. (2021). Gym-μrts: toward affordable full game real-time strategy games research with deep reinforcement learning. In *2021 IEEE Conference on Games (CoG)*, pages 1–8. IEEE.

Huberman, B. J. (1968). *A program to play chess end games*. PhD thesis, Department of Computer Science, Stanford University.

hun Cho, C. (1997). *GO: A complete introduction to the game*. Kiseido.

Kahlen, S.-M. (2004). Uci protocol. Accessed: 2022-09-21.

Kliegr, T., Bahník, Š., and Fürnkranz, J. (2021). A review of possible effects of cognitive biases on interpretation of rule-based machine learning models. *Artificial Intelligence*, page 103458.

Krishnan, A. and Martens, C. (2022a). Synthesizing interpretable chess tactics from player games. In *Proceedings of the Workshop on Artificial Intelligence for Strategy Games (SG) and Esports Analytics (EA), 18th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. American Association for Artificial Intelligence.

Krishnan, A. and Martens, C. (2022b). Towards the automatic synthesis of interpretable chess tactics. In *Proceedings of the Explainable Agency in Artificial Intelligence Workshop, 36th AAAI Conference on Artificial Intelligence*, pages 91–97. American Association of Artificial Intelligence.

Krishnan, A., Martens, C., and Jhala, A. (2023). Improving strategy synthesis for chess using precision and recall. [Manuscript submitted for publication].

Lage, I., Chen, E., He, J., Narayanan, M., Kim, B., Gershman, S. J., and Doshi-Velez, F. (2019). Human evaluation of models built for interpretability. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 7, pages 59–67.

Levinson, R. and Snyder, R. (1991). Adaptive pattern-oriented chess. In *Machine Learning Proceedings 1991*, pages 85–89. Elsevier.

Li, J., Koyamada, S., Ye, Q., Liu, G., Wang, C., Yang, R., Zhao, L., Qin, T., Liu, T.-Y., and Hon, H.-W. (2020). Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*.

Lichess (2021). Chess rating systems. Accessed: 2022-09-23.

lichess.org (2021). lichess.org open database. https://database.lichess.org/. Accessed: 2021-10-27.

Lieberman, J. 2022. Analysing the biggest esports prize pools of 2022. Esports Insider. https://esportsinsider.com/2022/12/biggest-esports-prize-pools-2022 (Accessed: 2023-03-28).

Liu, G., Schulte, O., Zhu, W., and Li, Q. (2019). Toward interpretable deep reinforcement learning with linear model u-trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 414–429. Springer.

Lynch, J. 2017. As nfl ratings drop, a new internet study says young men like watching esports more than traditional sports. Business Insider. https://www.businessinsider.com/nfl-ratings-drop-study-young-men-watch-esports-more-than-traditional-sports-2017-9 (Accessed: 2023-03-21).

Madumal, P., Miller, T., Sonenberg, L., and Vetere, F. (2020). Explainable reinforcement learning through a causal lens. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2493–2500.

Mariño, J. R., Moraes, R. O., Oliveira, T. C., Toledo, C., and Lelis, L. H. (2021). Programmatic strategies for real-time strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 381–389.

Mariño, J. R. and Toledo, C. F. (2022). Evolving interpretable strategies for zero-sum games. *Applied Soft Computing*, 122:108860.

McCarthy, J. (1990). Chess as the drosophila of ai. In *Computers, chess, and cognition*, pages 227–237. Springer.

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Hassabis, D., Kim, B., Paquet, U., and Kramnik, V. (2021). Acquisition of chess knowledge in alphazero. *arXiv:2111.09259 [cs, stat]*. arXiv: 2111.09259.

McIlroy-Young, R., Sen, S., Kleinberg, J., and Anderson, A. (2020). Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 25th ACM SIGKDD international conference on Knowledge discovery and data mining*.

Medeiros, L. C., Aleixo, D. S., and Lelis, L. H. (2022). What can we learn even from the weakest? learning sketches for programmatic strategies. *arXiv preprint arXiv:2203.11912*.

Mobalytics. 2023. Mobalytics - the all-in-one companion for every gamer. https://mobalytics.gg/ (Accessed: 2023-03-21).

Molnar, C. (2018). A guide for making black box models explainable. *URL: https://christophm. github. io/interpretable-ml-book*, page 3.

Moraes, R. 2021a. Synpros - synthesis of programmatic strategies. https://rubensolv.github.io/synpros-microrts/ (Accessed: 2023-03-26).

Moraes, R. 2021b. Synthesis of programmatic strategies framework synpros · farama-foundation/microrts wiki · github. https://github.com/Farama-Foundation/MicroRTS/wiki/Synthesis-of-Programmatic-Strategies-Framework---SynProS (Accessed: 2023-03-26).

Morales, E. (1992). *First order induction of patterns in Chess*. PhD thesis, PhD thesis, The Turing Institute-University of Strathclyde.

Moritz, D., Wang, C., Nelson, G. L., Lin, H., Smith, A. M., Howe, B., and Heer, J. (2018). Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics*, 25(1):438–448.

Muggleton, S. H., Schmid, U., Zeller, C., Tamaddoni-Nezhad, A., and Besold, T. (2018). Ultra-strong machine learning: comprehensibility of programs learned with ilp. *Machine Learning*, 107(7):1119–1140.

Nelson, P. H. (2019). When magnus met alphazero. *New In Chess*, 2019(8):2–10.

Ontanon, S. (2021). The combinatorial multi-armed bandit problem and its application to real-time strategy games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9(1):58–64.

Ontañón, S., Barriga, N. A., Silva, C. R., Moraes, R. O., and Lelis, L. H. (2018). The first microrts artificial intelligence competition. *AI Magazine*, 39(1):75–83.

Pitrat, J. (1977). A chess combination program which uses plans. *Artificial Intelligence*, 8(3):275–321.

Puiutta, E. and Veith, E. M. (2020). Explainable reinforcement learning: A survey. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 77–95. Springer.

Reitman, J. G., Anderson-Coto, M. J., Wu, M., Lee, J. S., and Steinkuehler, C. (2020). Esports research: A literature review. *Games and Culture*, 15(1):32–50.

Research, G. V. 2022. Esports market size, share & trends analysis report by revenue source (sponsorship, advertising, merchandise & tickets, media rights), by region, and segment forecasts, 2022 - 2030. Grand View Research. https://www.grandviewresearch.com/industry-analysis/esports-market (Accessed: 2023-03-21).

Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56:1247–1293.

Romero, O. (2019). Computer analysis of world chess championship players. *ICSEA 2019*, page 212.

Rose, L. (2012). *Winning basketball fundamentals*. Human Kinetics.

Sadler, M. and Regan, N. (2019). Game changer. *AlphaZero's Groundbreaking Chess Strategies and the Promise of AI. Alkmaar. The Netherlands. New in Chess*.

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., and Heer, J. (2016). Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350.

Schubert, M., Drachen, A., and Mahlmann, T. (2016). Esports analytics through encounter detection. In *Proceedings of the MIT Sloan Sports Analytics Conference*, volume 1, page 2016. MIT Sloan Boston, MA.

Seirawan, Y. (2005). *Winning chess tactics*. Everyman Chess.

Shaker, N., Togelius, J., Nelson, M. J., Nelson, M. J., and Smith, A. M. (2016). Asp with applications to mazes and levels. *Procedural Content Generation in Games*, pages 143–157.

Shu, T., Xiong, C., and Socher, R. (2017). Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*.

Sieusahai, A. and Guzdial, M. (2021). Explaining deep reinforcement learning agents in the atari domain through a surrogate model. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 17, pages 82–90.

Silman, J. 2021. Fighting games explained: What is frame data and how to use it to your advantage? Playstation Competition Center. https://compete.playstation.com/en-us/all/articles/fighting-games-explained-what-is-frame-data-and-how-to-use-it-to-your-advantage (Accessed: 2023-03-24).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Slack, D., Friedler, S. A., Scheidegger, C., and Roy, C. D. (2019). Assessing the local interpretability of machine learning models. *arXiv preprint arXiv:1902.03501*.

Smith, A. M., Andersen, E., Mateas, M., and Popović, Z. (2012). A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 156–163.

Smith, A. M. and Mateas, M. (2011). Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):187–200.

Spronck, P., Sprinkhuizen-Kuyper, I., and Postma, E. (2004). Online adaptation of game opponent ai with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):45–53.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog: Advanced Programming Techniques*, pages 87–90. MIT Press, 2nd edition.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szabo, A. (1984). *Computer chess tactics and strategy*. PhD thesis, University of British Columbia.

Tjoa, E. and Guan, C. (2020). A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21.

Trivedi, D., Zhang, J., Sun, S.-H., and Lim, J. J. (2021). Learning to synthesize programs as interpretable and generalizable policies. *Advances in Neural Information Processing Systems*, 34.

van der Waa, J., van Diggelen, J., Bosch, K. v. d., and Neerincx, M. (2018). Contrastive explanations for reinforcement learning in terms of expected consequences. *arXiv preprint arXiv:1807.08706*.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. (2018). Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wilkins, D. E. (1979). *Using patterns and plans to solve problems and control search*. Stanford University.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725.

Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K. (2021). A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742.

Zhou, Y. (2018). Rethinking opening strategy: Alphago's impact on pro play. *CreateSpace*, 1(36):212.

APPENDICES

APPENDIX

A

ACRONYMS

A summary of all acronyms is documented in Table A.1. Need to update this.

Table A.1: A summary of acronyms used in alphabetical order.

Acronym	Abbreviation
Electronic sports	Esports
Reinforcement Learning	RL
Interpretable Strategy Synthesis	ISS
Real-time Strategy	RTS
Research Question	RQ
Inductive Logic Programming	ILP
Explainable Reinforcement Learning	XRL
Neural Network	NN
relative least general generalization	rlgg
Patterns and Learning	PAL
Top Chess Engine Championship	TCEC

APPENDIX

B

VARIABLES

A summary of all variables is documented in Table B.1. Need to update this.

Table B.1: A summary of variables and their abbreviations used in this document in order of their use.

Variable	Abbreviation
Game environment	$\mathcal{G}$
State space	$\mathcal{S}$
Action space	$\mathcal{A}$
State transition function	$\mathcal{P}$
Reward function	$\mathcal{R}$
Discount factor	γ
Strategy	σ
State	s
Timestep	t
Applicable states	A_σ
Interpretability measure	I

APPENDIX

C

BACKGROUND KNOWLEDGE DEFINITIONS

This appendix has all the background knowledge definitions used by our system. These do not include any helper rules used to define the predicates in the background knowledge since those are not used to define tactics.

- `legal_move(Board,Move)`: Defines legal moves of chess pieces.
- `in_check(Board,Side,Square)`: Defines whether a particular side is in check by a piece.
- `pawn_capture(Board,Move)`: Defines a pawn capture move.
- `castling_move(Board,Move)`: Defines valid castling move.
- `make_move(Board,Move,NewBoard)`: Performs the actual movement of a piece by changing the state appropriately.
- `can_capture(Board,Sq1,Sq2)`: Defines whether a piece on square `Sq1` can capture another on square `Sq2`.

- `is_capture(Board,Move)`: Defines valid capture move.
- `en_passant(Board,Square)`: Defines the en passant square on the current board, if any.
- `queenside_castle(Board,Side)`: Defines the queenside castling rights of the side.
- `kingside_castle(Board,Side)`: Defines the kingside castling rights of the side.
- `turn(Board,Side)`: Defines who's turn it is to make a move on the current board.
- `is_attacked(Board,Square,SquareSet)`: Defines the squares that are being attacked by a piece.
- `can_attack(Board,Sq1,Sq2)`: Defines an attack being carried out by a piece on one square, on another square.
- `is_empty(Board,SquareSet)`: Defines a set of empty squares on the board.
- `xrays(Board,Behind,Middle,Front)`: Defines a piece that x-rays another piece.
- `valid_piece_at(Board,PieceType,Col,Sq)`: Defines the existence of a piece and its attributes at a given square.
- `remove_piece_at(Board,Square,NewBoard)`: Removes a piece from the given square.
- `move(Move,From,To,PromoList)`: Unifies the list representation of a move with its component parts, for e.g., `move([a7,a8,q],a7,a8,[q])`.
- `other_color(Color,OtherColor)`: Defines a relationship between the two opposing colors on the board.
- `color(Color)`: Defines a valid color.
- `sliding(PieceType)`: Defines a piece that moves in a straight line.
- `piece_type(PieceType)`: Defines a valid piece type.
- `piece(Piece)`: Defines a valid piece.
- `sq_between_non_incl(Sq1,Sq2,Sq2)`: Square Sq3 is in the same straight line defined by squares Sq1 and Sq2 (non-inclusive).
- `different(Sq1,Sq2)`: Squares Sq1 and Sq2 are different.