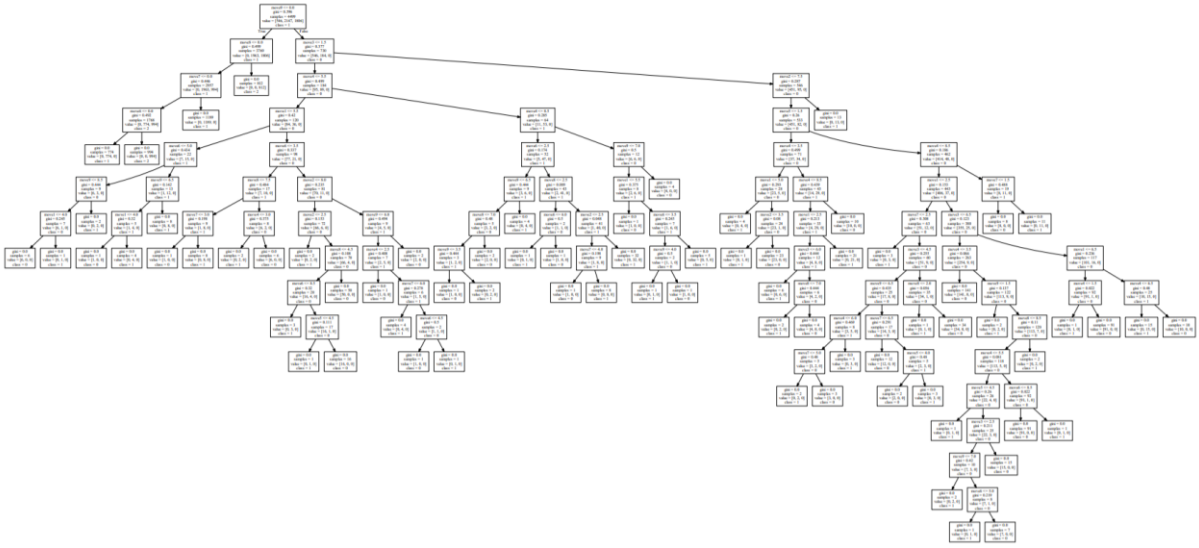
Utrecht University

# Learning board game rules by observing game play, a comparison of symbolic and non-symbolic AI.



**Bachelor Thesis Artificial Intelligence**

Seth Teekens
5550068
7.5 ECTS

Supervisor: Mel Chekol
Second reader: John Jules Meyer
6 November 2020

**Abstract**

Usually the focus of Artificial Intelligence (AI) game research is on learning strategies for specific games. This thesis reversed this focus by looking for methods capable of learning game rules in general. The goal is to learn the rules of the simple board game Tic Tac Toe by observing played games in a way that can be used for more complicated games. This will be done by testing the performance of the symbolic AI algorithm ProbFOIL$^+$ and two non-symbolic AI algorithms, the k-Nearest-Neighbor (KNN) algorithm and a Decision Tree (DT) algorithm. Both ProbFOIL$^+$ and DT succeed in learning the win rule of Tic Tac Toe, but no algorithm succeeds in learning a more complex rule. In this case the Recall of ProbFOIL$^+$ is very low whereas KNN and DT both overfit on the data. The strengths and weaknesses of both symbolic and non-symbolic AI seems to supplement each other and therefore it is suggested that future work focuses on combining these two.

# Table of contents

# 1. Introduction

Games have always been a popular topic in Artificial Intelligence (AI) research. Usually the focus is on learning specific game strategies. Such as learning to play Chess or Go. The inspiring work from Google DeepMind (Evans et al., 2019) on the Apperception Engine focuses on letting a program make sense of sensory data. What they mean by making sense is constructing a symbolic causal theory that explains the sensory sequence and that satisfies a set of unity conditions, as a form of unsupervised program synthesis. The unity conditions are the key constraints on the system. There are four types of elements in the system which are objects, predicates, sets of atoms and sequences of sets of atoms. Each of these elements has its own form of unity. Evans et al. (2019), succeeded in letting the Apperception Engine make sense of a variety of sensory sequences such as games, IQ tests, rhythm (nursery rhymes), occlusion tasks and more. The output of the Apperception Engine is readable for human, thus it is possible to see what the program has learned. The interesting thing about this approach is that program can make sense of things by itself. Instead of programming the rules, the system will have to learn the rules itself.

Closely related to this is the General Game Playing (GGP) competition from the Association for the Advancement of Artificial Intelligence (AAAI). In GGP the goal is to develop general algorithms that can learn to play different games (Genesereth et al., 2005). General game players are systems able to accept declarative descriptions of arbitrary games at run time and able to use such descriptions to play those games effectively (without human intervention) (Genesereth et al., 2005). In order to perform well, general game players must incorporate various AI technologies, such as knowledge representation, reasoning, learning, and rational decision making. These capabilities must work together in an integrated fashion. For the competition and general game learning experiments they have developed a Game Description Language (GDL), which is also used by Evans et al. (2019). GDL is a general way to describe all the aspects of a game in First Order Logic (FOL). Inspired by these approaches to AI this thesis will try to contribute to the paradigm of making sense of games. In order to achieve this the focus will be on learning the rules of board games. Starting with Tic Tac Toe, which is a simple board game with clear rules.

The main goal of this study is to learn the rules of Tic Tac Toe from datasets of played games. This will be done in a format closely related to GDL, so that the rule learning can be applied to other board games as well. A part of this research will also be the creation of datasets. In order to learn the rules, both Inductive Logic Programming (ILP) as well as machine learning techniques will be used. The sub goal is to compare the effectiveness of using a symbolic approach compared to a machine learning approach.

This thesis is organized as follows. An overview of related work on ILP systems can be found in the background chapter 2. This chapter also contains the basic concepts of rule learning and an brief explanation of the algorithms that are used. In chapter 3 the methodology is explained. First the rules to be learned are described and then how the datasets are configured to learn these rules. In chapter 4 the results are discussed. Chapter 5 contains the discussion and suggestions for future work. Chapter 6 is the conclusion.

| | Predicted True | Predictive False | |
|---|---|---|---|
| Real True | TP | FN | P |
| Real False | FP | TN | N |
| | | | M |

**Table 1.** Contingency table from Probabilistic Rule Learning (De Readt and Thon, 2010).

$$precision = \frac{TP}{TP+FP} \qquad m\text{-}estimate = \frac{TP+m\cdot\frac{P}{N}}{TP+FP}$$

$$recall = \frac{TP}{TP+FN} \qquad accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

**Fig. 1.** Performance metrics from Probabilistic Rule Learning (De Readt and Thon, 2010).

# 2. Background

As the main goal is to learn rules of board games by observing game play, we will start with looking at available rule learner systems and the machine learning techniques. This chapter will focus on the available methods for rule learning and discuss the methods that are chosen for the goal of this thesis.

## 2.1. Rule Learning Definitions

This section will explain basic terminology of machine learning and the performance metrics that are used throughout the paper.

Machine learning can be categorized as supervised, semi supervised or unsupervised. Supervised learning means that a program will learn from labelled data. It does this by splitting the data into test data and training data. The program learns a hypothesis from the test data and then tests this hypothesis on the training data. The program needs to predict the correct label for the training data. Unsupervised learning means finding organization in unlabeled data. Semi supervised is a combination of supervised and unsupervised. If the output is a finite set then this is called classification. If the output is a number then this is a form of regression learning. A hypothesis is said to generalize well if it correctly predicts output for new examples. A learning model that summarizes data with a set of parameters that is independent of the number of training examples is called a parametric model (Russell and Norvig, 2010).

In order to measure the performance of a program or a hypothesis a contingency table is used as shown in figure 1. P stands for all the positive examples, N for all the negative examples and M for all the examples. There are four ways of classifying a data point which are:

- TP: true positive
- FP: false positive
- FN: false negative
- TN: true negative

The performance metrics that will be used in this thesis are shown in figure 2 and are the same as in the paper from De Readt and Thon (2010). Accuracy stands for the amount of correctly classified data divided by the total amount of data. Recall stands for how many relevant items are selected. Precision stands for the proportion of how many selected items are relevant. The m-estimate is a variant of precision that is more robust against noise in the data (Džeroski et al., 1993).

A learning model is said to generalize well if the model fits the target function. Overfitting occurs when the learning model is more complex than necessary to fit the target function (Abu-Mostafa et al., 2012). If the dataset is small then there should be a strong restriction on the amount of

hypothesis to avoid overfitting. Overfitting becomes less likely if the amount of training examples increases (Russell and Norvig, 2010).

## 2.2. Inductive Logic Programming

Common problems with modern state-of-the-art machine learning techniques such as deep learning and neural networks is that they are bad at generalization, interpretability and that they need a large number of training examples (Cropper et al., 2020). This is why we will start by looking at an alternative method: symbolic AI, more specifically ILP. One of the advantages of symbolic AI is that it is explainable, which means that the output is understandable by humans. This will help with the goal of creating AI that can make sense of games by itself since we can track the learning process.

The goal of an ILP system is to learn (induce) a hypothesis from a dataset of positive and negative examples plus background knowledge, that contains as many positive examples as possible and none of the negative examples (Cropper et al., 2020). Logic programming is a type of programming language that is based on if then rules instead of objects or functions. The if then rules are called clauses which have the form Head :- Body. If the clause does not contain a body then this clause is a fact. Otherwise the clause is a rule. A logic program consists of definite clauses. A definite clause contains one non negated atom which is the head and it contains a body. An atom $p(t_1, .., t_n)$ consists of a predicate p of arity n and terms $t_1$, ..,$t_n$. Terms can be constants (lowercase), variables(uppercase) or functors. A literal is an atom or its negation. An atom is ground if it does not contain variables (De Readt et al., 2015).

Cropper et al. (2019) wanted to show that Inductive General Game Playing (IGGP) is difficult for ILP systems and that there are a lot of unresolved issues. They did this by creating a dataset in GDL notation, that is created by game play of the GGP competition. They tested different ILP systems on how well these systems are able to learn the rules of the games in the dataset (Cropper et al., 2019). Next an overview of existing ILP systems is given, including the ones used by Cropper et al. (2019).

One of the first ILP algorithms is FOIL. What this algorithm does is that it repeatedly constructs a clause that agrees with a subset of the positive examples and none of the negative examples. The algorithm then removes the positive examples from the training set and continues this process until there are no more positive examples in the training set (Quinlan, 1990). This is a top-down approach. Aleph is an ILP system that uses a bottom up approach. Both FOIL and Aleph use Prolog as interpreter. Aleph follows these 4 steps (Muggleton, 1995):

1. Select example to be generalised
2. Build the most specific clause that entails the selected example
3. Search for a clause that is more general than the bottom clause
4. Remove the redundant clause and add the best clause from step 3 to the current theory. Return to step 1

Two common disadvantages of these older ILP systems is that they struggle to learn recursive programs and that they require handcrafted background knowledge (Evans and Grefenstette, 2018).

Recursion enables an ILP system to generalise better from a small number of examples. Metagol is an ILP system that can handle recursion as well as predicate invention (Cropper and Muggleton, 2016). What Metagol does is that it uses meta-interpretive learning (MIL). MIL uses metarules (higher order clauses) to restrict the hypothesis space by restricting the form of inducible programs (Cropper and Muggleton, 2016).

The other two systems that Cropper et al. (2019) tested are ILASP and ASPAL. ILASP is a collection of ILP system which can learn Answer Set Programs (ASP). ASP is a type of programming language. An ASP solver finds the set of answer sets for a normal logic program (Evans et al., 2019). ILASP guarantees an optimal inductive solution defined as the length of the hypothesis (Cropper et al., 2019). ASPAL and ILASP are both capable of predicate invention and recursion. Cropper et al. (2019) tested Aleph, Metagol, ASPAL and ILASP on the IGGP dataset. As well as the k-Nearest-Neighbor algorithm, which will be described in the next section. From these systems ILASP had the best performance on solving tasks with 100% accuracy (Cropper et al.,2019). The best performance however was only 40%, this shows that there is a lot of room for improvement in learning game rules by observing game play.

Another disadvantage of ILP systems is that they are not robust to noise. A possible solution for this is using a probabilistic ILP system. ProbFOIL$^+$ is such a system, since ProbFOIL$^+$ is used for the experiment of this thesis it is discussed in more depth.

### 2.2.1. ProbFOIL$^+$

The ProbFOIL$^+$ algorithm can learn rules through ILP. The reason for choosing to use ProbFOIL$^+$ is that it is the best available inductive rule learner written in Python, that can handle predicates with multiple arities. ProbFOIL$^+$ combines the FOIL algorithm for rule learning with ProbLog, which is a probabilistic version of Prolog (De Raedt et al., 2015). When the probabilities are set to 1 and 0 ProbFOIL$^+$ functions in the same way as deterministic rule learners such as FOIL (Quinlan, 1990). In 2010 de Readt and Thon introduced ProbFOIL and in 2015 they improved ProbFOIL by creating ProbFOIL$^+$. ProbFOIL$^+$ will be used for this thesis.

What ProbFOIL$^+$ essentially does is that it repeatedly adds clauses to the hypothesis until more clauses decrease the quality of the hypothesis. It develops a set of hypothesis that account for all the positive examples, but none of the negative examples. The algorithm calls the LearnRule function until there is no more improvement of the global scoring function. The global scoring function is based on the accuracy of the hypothesis. The LearnRule function searches for a clause that maximizes the local scoring function. The local scoring function is based on the m-estimate of the hypothesis. A beam search strategy is used by the LearnRule function to escape from local maxima (De Raedt et al., 2015).

On the ProbFOIL website https://pypi.org/project/probfoil/ there is a manual on how to use the algorithm, for clarity a brief overview is given in the appendix.

## 2.3. Non-Symbolic AI

One of the main issues with symbolic AI is that it does not scale well. This is why we will look at two non-symbolic and simple statistical machine learning algorithms and compare these to the performance of ProbFOIL$^{+}$.

### 2.3.1. Decision Tree

Decision Tree learning shares the same principles as the top down ILP approach. It starts with a general rule and gradually specializes so that it fits the data (Russell and Norvig, 2010). Instead of using first-order literals it uses a vector of attributes as input. The hypothesis is a decision tree instead of a set of clauses. Decision tree learning is a non-parametric supervised learning method.

Scikit uses a version of the CART algorithm (Pedregosa et al., 2011). The CART algorithm constructs binary trees (Breiman et al., 1984). Based on the Gini splitting criteria this algorithm uses a divide-and-conquer approach. It divides the problem into smaller sub problems that it solves recursively. The Gini Index is a measure of impurity that measures how often a randomly chosen element would be incorrectly identified. Unfortunately this Decision does not support categorical data, therefore all the datasets are structured in numbers and later converted to tags such as "win(x)" for the readability of the tree. This can of course influence the tree.

### 2.3.2. k-Nearest-Neighbors

For this experiment the Scikit KNeighborsClassifier is used for supervised learning. It is also possible to use the k-Nearest-Neigbors (KNN) algorithm for unsupervised learning. The algorithm classifies data points by looking at the classifications of the nearest neighbors of the data point. It uses clusters to predict the class of the data. The variable k stands for the number of clusters that the algorithm will look at to classify the data point. It classifies the point with the label of the cluster that has the most nearest neighbors.

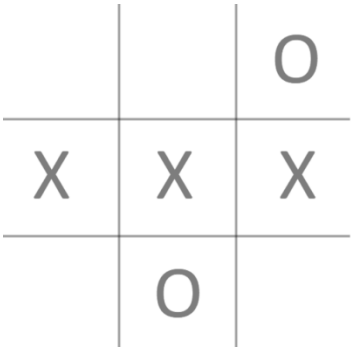



**Fig. 2.** Example of terminal state of Tic Tac Toe.

| | | |
|---|---|---|
| 1,1 | 1,2 | 1,3 |
| 2,1 | 2,2 | 2,3 |
| 3,1 | 3,2 | 3,3 |

**Fig. 3.** Cell numbering of Tic Tac Toe board.

# 3. Methodology

Tic Tac Toe is a well known board game. The states of Tic Tac Toe consist of a 3 x 3 grid where each cell is either blank or marked with an x or an o. It is a two player alternating turn game. On each turn a player with control does a move by marking a cell with an x or o depending on whether the player has the role x or o. Therefore the role that a player can have is either x or o. If a player has managed to mark three cells in a row then that player has won and the game terminates. The three in a row can be vertically, horizontally or diagonally. If the entire board is filled with x's and o's, but there are no three in a row then the game terminates with a draw. An example of a terminate state is shown in figure 2. In the IGGP dataset of Cropper et al., (2019) Tic Tac Toe is described with 32 rules.

This chapter will start with modelling of the rules that we want to learn. Then it will describe how the datasets are configured and finally how the algorithms are tested.

## 3.1. Modelling Rules of Tic Tac Toe

At the start of the game every cell is empty, in GDL this is the initial relation. There are 9 cells on the board which are numbered as shown in figure 3. For each rule there are multiple ways to model the rule. For this thesis the win rule and the legal move rule are examined. The win rule is described as a binary first order predicate. The target predicate is described as learn(win/2).

The win rule is described with the following 8 rules:

**Win Rule**

win(A,B) : - $cell_{11}$(A,B), $cell_{12}$(A,B), $cell_{13}$(A,B).
win(A,B) : - $cell_{21}$(A,B), $cell_{22}$(A,B), $cell_{23}$(A,B).
win(A,B) : - $cell_{31}$(A,B), $cell_{32}$(A,B), $cell_{33}$(A,B).

win(A,B) : - $cell_{11}$(A,B), $cell_{21}$(A,B), $cell_{31}$(A,B).
win(A,B) : - $cell_{12}$(A,B), $cell_{22}$(A,B), $cell_{32}$(A,B).
win(A,B) : - $cell_{13}$(A,B), $cell_{23}$(A,B), $cell_{33}$(A,B).

win(A,B) : - $cell_{11}$(A,B), $cell_{22}$(A,B), $cell_{33}$(A,B).
win(A,B) : - $cell_{13}$(A,B), $cell_{22}$(A,B), $cell_{31}$(A,B).

Where A stands for the player x or o (or e if the cell is empty ) and B stands for the number of the game.

The legal move rule is more complex to describe than the win rule. Therefore the example is only written for one cell, of course the rule applies to each cell of the board. The main point of the legal move rule is that if the cell is empty in the previous state then the move is legal. Since Tic Tac Toe is an alternating turn game, it should also be the turn of the player for the move to be legal. Even though notation should not influence the results, different notations are tried out to test this and to find the most usable notation. Therefore the legal move rule is modelled in three ways:

legal_move(A,B,C,D) :- $cell_{11}$(E,F,D), $cell_{11}$(A, C, D).

A: the player x or o
C: current state
D: game number
E: empty cell, contains no player x or o
F: previous state

Which means that $cell_{11}$ was empty in the previous state F and has player A in the current state C of game D.

legal_move(A,B,C,D) :- cell(E,B,F,D), cell(A, B, C,D).

A: the player x or o
B: position of the cell
C: current state
D: game number
E: empty cell, contains no player x or o
F: previous state

Which means that the cell was empty in the previous state F and has A in the current state.

legal_move(A,B,C,D) :- empty(B,F,D), cell(A,B,C,D).

A: the player x or o
B: position of the cell
C: current state
D: game number
F: previous state

Which means that cell in position B was empty in the previous state F and has player A in the current state C (of game D).

## 3.2. Dataset preparation using Minimax

In order to learn the rules different datasets are created and tried out so that the algorithms can perform in the best way possible. For logic programming a different dataset had to be created than for the machine learning algorithms. The description of the datasets per algorithm will be discussed in this chapter and the datasets can be found on the github page of this project.

The datasets are created by letting an A.I. play minimax and a player that does random moves play against each other. The minimax algorithm works by assuming that each player has an optimal strategy (Russell and Norvig, 2010). If both players would use minimax then the outcome of the game would be a draw each time in the case of Tic Tac Toe. For more complex games it would be a good strategy to create a dataset by letting two AI's play minimax against each other.

| Dataset | Nr of games | Nr of facts | Variation in dataset |
|---|---|---|---|
| ProbWin1 | 50 | 672 | No negative examples |
| ProbWin2 | 50 | 728 | 56 Negative examples |
| ProbWin3 | 100 | 1437 | 115 Negative examples, Size |
| ProbWin4 | 100 | 1455 | 115 Negative examples, Mode |
| ProbWin5 | 1000 | 13022 | 1138 Negative example, Size |

**Table 2.** The used datasets for ProbFOIL$^+$ win rule and their variation. The variation types are explained in section 3.2.1.

| Dataset | Nr of games | Nr of facts | Variation in dataset | Nr of Negative Examples |
|---|---|---|---|---|
| ProbLegal1 | 4 | 399 | $Cell_{11}$(player,state,game) | 13 |
| ProbLegal2 | 3 | 253 | Cell(player,pos,state,game)<br>Mode: Cell(c,c,+,+) | 7 |
| ProbLegal3 | 25 | 2282 | Cell(player,pos,state,game) | 100 |
| ProbLegal4 | 5 | 569 | empty(position,state,game) | 31 |
| ProbLegal5 | 25 | 2440 | empty(position,state,game) | 114 |
| ProbLegal6 | 100 | 9378 | empty(position,state,game) | 379 |
| ProbLegal7 | 25 | 2363 | $Cell_{11}$(player,state,game)<br>instead of legalmove(o,11,0,0)<br>Legalmove(o,cell11,0,0) | 111 |
| ProbLegal8 | 25 | 2100 | Player(). fact defined at init | 73 |

**Table 3.** The used datasets for ProbFOIL$^+$ legal move rule as described in section 3.2.1.

### 3.2.1. ProbFOIL$^+$

A part of the research is trying out which format of datasets would work best with this algorithm. The possible variations of the datasets are:

- Size. Changing the amount of played games.
- Negative examples. Instead of using the automode of ProbFOIL$^+$ to create negative examples, the dataset itself will contain facts with a 0 probability to show that the fact is not valid. There are two types of negative examples. The first one is showing which player did not win, in the form: 0::win(x,1). The other negative examples are created dynamically by allowing the random player to make illegal moves, if the program sees that the move is illegal it will write it down as a negative example: 0::legal_move(x,23,9,0).
- Mode variation. Changing the specifier for which predicate can be added to the learned rule.

For the win rule 5 different datasets are compared. Which are shown in table 2. As explained in the how to use ProbFOIL$^+$ section the datasets contain the settings and facts. The facts are end game configurations for each game in this case. The settings are described as follows:

base(win(player,game). *player can be x, o or e(empty)*
base($cell_{11}$(player,game). *for each cell 11, 12, .., 33*
mode($cell_{11}$(+,+). *for each cell 11, 12, ..., 33*
learn(win/2).

For the legal move rule 8 different datasets are compared which are shown in table 3. The difference in the creation of the datasets for learning the legal_move rule is that the type state is added. This is

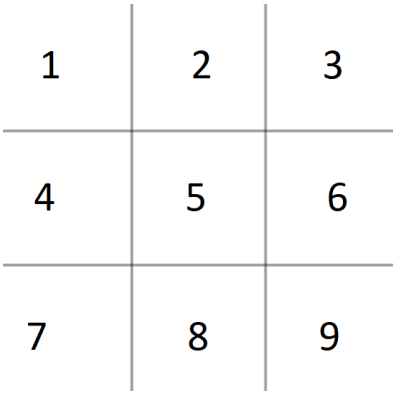



**Fig. 4.** Alternative cell numbering of Tic Tac Toe board as used for dataset MLWin2.

done so that all parts of the game can be represented instead of only the end game configuration. So we will have the predicate:

legal_move(player, move, state, game)

In the legal_move datasets there are also initialization facts to show that in the first state of a game each cell is empty. The following facts will be a legal move and then an update of all the cells and finally which player has won. Then the next game will start. Each dataset contains negative examples since this proved useful when learning the win rule. In table an overview of the datasets are given.

### 3.2.2. Non-symbolic AI

For the decision tree learning and the KNN algorithm two datasets are created for the win rule and two for the legal move rule. Both algorithms are executed with a test size of 0.1.

In figure 5 the MLWin1 dataset is shown. For game 0, in cell11 there is an x, in cell12 there is an x, in cell13 there is an o. The final class represents the winner 1 or 2 or a draw 0. In figure 6 the MLWin2 notation is shown. In this case player 1 always starts, so move1 will always be player 1. The move the player does is in the cell 1 to 9, starting at the left top of to board from left to right as shown in figure 4. Move -1 means that the game is finished and that there are no more moves.

The MLLegal1 dataset in figure 7 works as follows. It shows which player makes a move in a cell and classifies the move as legal which 1 and illegal is 0. In the second dataset MLLegal2 as shown in figure 8 the variable state is added to make it more clear in which stage of the game the move is made.

| | cell11 | cell12 | cell13 | cell21 | cell22 | cell23 | cell31 | cell32 | cell33 | CLASS |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 0 |
| 1 | 0 | 0 | 1 | 2 | 2 | 1 | 0 | 0 | 1 | 1 |
| 2 | 1 | 2 | 1 | 0 | 1 | 2 | 2 | 0 | 1 | 1 |
| 3 | 2 | 2 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 2 |
| 4 | 1 | 1 | 1 | 2 | 0 | 0 | 2 | 0 | 0 | 1 |

**Fig. 5.** Subset of MLWin1 dataset.

| | move1 | move2 | move3 | move4 | move5 | move6 | move7 | move8 | move9 | CLASS |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 5 | 1 | 8 | 2 | 6 | 4 | 9 | 7 | -1 | 2 |
| 1 | 8 | 9 | 1 | 4 | 2 | 7 | 5 | -1 | -1 | 1 |
| 2 | 7 | 5 | 9 | 8 | 2 | 1 | 6 | 3 | 4 | 0 |
| 3 | 1 | 5 | 7 | 4 | 3 | 2 | 9 | 8 | -1 | 2 |
| 4 | 7 | 3 | 1 | 2 | 4 | -1 | -1 | -1 | -1 | 1 |

**Fig. 6.** Subset of MLWin2 dataset**.**

| | game | player | move | legal |
|---|---|---|---|---|
| 0 | 0 | 0 | 6 | 1 |
| 1 | 0 | 1 | 4 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 3 | 1 |
| 4 | 0 | 0 | 7 | 1 |
| 5 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 7 | 0 |
| 7 | 0 | 1 | 7 | 0 |
| 8 | 0 | 1 | 8 | 1 |
| 9 | 0 | 0 | 2 | 1 |

**Fig. 7.** Subset of MLLegal1.

| | game | player | move | state | legal |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 3 | 1 | 1 |
| 1 | 0 | 0 | 5 | 2 | 1 |
| 2 | 0 | 1 | 6 | 3 | 1 |
| 3 | 0 | 0 | 9 | 4 | 1 |
| 4 | 0 | 1 | 1 | 5 | 1 |
| 5 | 0 | 0 | 2 | 6 | 1 |
| 6 | 0 | 1 | 8 | 7 | 1 |
| 7 | 0 | 0 | 4 | 8 | 1 |
| 8 | 0 | 1 | 8 | 9 | 0 |
| 9 | 0 | 1 | 9 | 9 | 0 |

**Fig. 8.** Subset of MLLegal2 dataset.

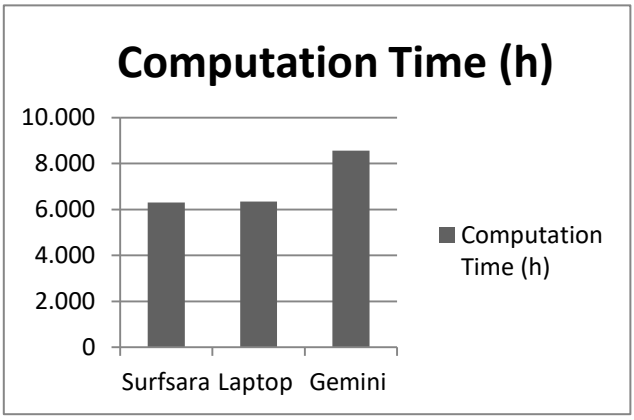



**Fig. 9.** Speed comparison of the ProbLegal7.pl dataset.

### 3.3. Testing the algorithms

For the computation of the results three methods are used. These methods are running the algorithms on the Gemini server of the beta faculty of Utrecht University, on the Surfsara server and on a HP laptop. The specifications the HP laptop are: HP ZBook 15 with a Intel(R) Core(TM) i70400MQ CPU 2.40GHz 2.40GHz processor. On the laptop Python is used to run the algorithms.

If the Surfsara server is used by a student account, which was the case, it usually runs on 3CPUs on LISA. However if the system is busy, it could be downscaled. Therefore it cannot be said exactly how many CPU was used for this example. The same holds for the Gemini server, since the Gemini server was slower than the laptop it was probably less than 8 CPU. A comparison of the speed of the three methods is shown in the result section.

## 4. Results

The computation speed of the devices and servers that are used for this research are compared by using a dataset of 2363 entries for learning the legal move rule with the ProbFOIL algorithm. This dataset contains 25 played games of Tic Tac Toe. As shown in figure 9. the Surfsara server is the fastest with 6.297 hours seconds although it is comparable to the HP laptop with 6.345 hours. The Gemini server is a lot slower with a time of 8.557 hours. The advantage of using a server instead of a personal laptop is that it can keep running without interference. However since the Surfsara server was only accessible once for this project and the Gemini server is slower, the laptop is mostly used.

```
================ FINAL THEORY ================
win(A,B) :- cell11(A,B), cell21(A,B), cell31(A,B)
win(A,B) :- cell11(A,B), cell12(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell11(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), cell32(A,B)
win(A,B) :- \+cell32(A,B), \+cell33(A,B), \+cell23(A,B), \+cell12(A,B), \+cell22(A,B), \+cell21(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell33(A,B), cell31(A,B), cell32(A,B)
win(A,B) :- cell13(A,B), cell23(A,B), cell33(A,B)
```

**Fig. 10.** The learned rules of the ProbWin5 dataset.

| Dataset | Nr of games | Time (s) | Rule evaluations | Accuracy | Precision | Recall | Correct rules |
|---|---|---|---|---|---|---|---|
| ProbWin1 | 50 | 126 | 1339 | 0.98 | 1 | 0.92 | 4/8 |
| ProbWin2 | 50 | 218 | 1652 | 0.97 | 1 | 0.88 | 3/8 |
| ProbWin3 | 100 | 632 | 1872 | 0.96 | 1 | 0.84 | 5/8 |
| ProbWin4 | 100 | 2701 | 545 | 0.96 | 1 | 0.84 | 5/8 |
| ProbWin5 | 1000 | 16284 | 1951 | 1 | 1 | 1 | 8/8 |

**Table 4.** Performance of the learned win rules of the ProbFOIL$^+$ algorithm. All datasets are computed on the HP laptop.

| Learned Rule | Datasets |
|---|---|
| legal_move(A,B,C,D) :- cell(A,B,C,D). | ProbLegal3<br>ProbLegal5<br>ProbLegal6 |
| Legal_move(A,B,C,D) :- \+cell32(A,C,D), cell22(A,C,D), cell11(A,C,D). | ProbLegal1 |
| legal_move(A,B,C,D) :- cell11(A,C,D), \+cell12(A,C,D), \+cell23(A,C,D), cell21(A,C,D), cell31(A,C,D) | ProbLegal7<br>ProbLegal8 |
| legal_move(A,B,C,D) :- cell(o,11,C,D). | ProbLegal2 |

**Table 5.** Learned legal move rules of the ProbFOIL$^+$ algorithm.

| Dataset | Nr of facts | Time (s) | Rule evaluation | Accuracy | Precision | Recall |
|---|---|---|---|---|---|---|
| ProbLegal1 | 399 | 1011 | 476 | 0.964 | 0.111 | 0.036 |
| ProbLegal2 | 253 | 1091 | 2418 | 0.586 | 0.037 | 0.556 |
| ProbLegal3 | 2282 | 1977 | 4 | 0.826 | 0.047 | 0.491 |
| ProbLegal4 | 569 | 154 | 18 | 0.865 | 0.104 | 0.525 |
| ProbLegal5 | 2440 | 6141 | 18 | 0.897 | 0.113 | 0.478 |
| ProbLegal6 | 9378 | 79928* | 18 | 0.903 | 0.111 | 0.465 |
| ProbLegal7 | 2363 | 22843 | 677 | 0.977 | 0.111 | 0.047 |
| ProbLegal8 | 2100 | 41263 | 670 | 0.977 | 0.111 | 0.029 |

**Table 6.** Performance of the learned rules of ProbFOIL$^+$ algorithm. *ProbLegal6 is computed on the gemini server. The other datasets are computed on the HP laptop.

## 4.1. Learned rules

In this section the learned rules and the performance of these rules are presented and discussed.

### 4.1.1. ProbFOIL$^+$

The default setting of beam size 5 is used for ProbFOIL$^+$. The learned rules of all the win rule datasets can be found in the appendix. In table 4 the results of each dataset of the win rule is shown. With each dataset the results steadily improve. The adding of negative examples improved the rule

learning even though this is not visible from the results. Although ProbWin1 has one more rule completely correct compared to ProbWin2, ProbWin2 has more rules that are almost correct. This is an indication that the added negative examples have a positive effect on the rule learning. The mode adjustment of the ProbWin4 dataset drastically increased the rule evaluations and therefore the time. The mode adjustment did not improve the results.

By improving the notation and setting configuration with small datasets the last test was to try the optimal configuration on a larger dataset. This dataset ProbWin5 achieved the best results by learning all the 8 rules as shown in figure 10. The fifth rule that this dataset has learned was not one of the goal rules to learn. This rule uses the negation as failure notation from Prolog. What it means is that if \+cell(A,B) is true if this cell cannot be inferred. Therefore the rule states that win(A,B) cannot be true if all those other cells are true. This is correct, but this is not an useful and logical game rule. Therefore this can be seen as overfitting on the data.

As discussed in the methodology there are three forms of the legal move that we wanted to learn:

legal_move(A,B,C,D) :- $cell_{11}$(E,F,D), $cell_{11}$(A,C,D).

legal_move(A,B,C,D) :- cell(E,B,F,D), cell(A,B,C,D).

legal_move(A,B,C,D) :- empty(B,F,D), cell(A,B,C,D).

The rules that ProbFOIL$^+$ actually learned are shown in table 5. From this table we can see that the rules are not learned successfully. Which is also shown in the result overview in table 6. The accuracy is fairly high, however the recall and precision are very low and almost zero. This means that there

are relatively few true positives and a high number of true negatives. The true negatives are probably because of the automatically generated negative examples created by the auto mode of ProbFOIL$^+$. The datasets of the legal move rule do not only contain the end configuration, but the entire game. This makes a big impact on the dataset sizes which explains the large computation time.

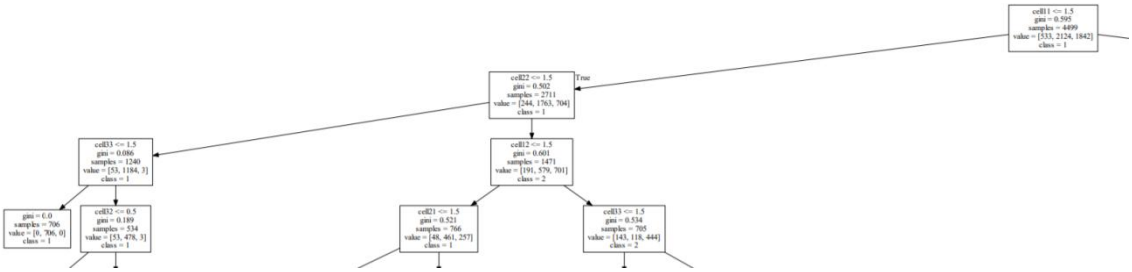


**Fig. 11.** Small example of the DT tree output of the MLWin1 dataset.

```
|--- cell11 <= 1.50
|   |--- cell22 <= 1.50
|   |   |--- cell33 <= 1.50
|   |   |   |--- class: 1
```

**Fig. 12.** Small example of the DT text output of the MLWin1 dataset.

### 4.1.2. Non-symbolic AI

The ouput of the decision tree can be found on the github page. The output is available in pdf format as trees and in text format as logical rules.

For the Win rule the classes are: 0 draw, 1 x, 2 o.

For the Legal Move rule the classes are: 0 illegal, 1 legal.

Within the decision trees some clear rules are present, such as: win (x) :- cell11(x), cell22(x), cell33(x). This can be deduced from the fragment of the tree as shown in figure 11. Although the rule is more clearly visible in the text fragment shown in figure 12. If cell11 and cell22 and cell33 all have a value below 1.5 then the class is 1. Since player x=1 and player o=2 this means that the tree correctly identified this rule. Although some rules are correct, the decision tree overfits on the data by learning all the possible board configurations. For a simple game like Tic Tac Toe this may even work, but for more complex games or rules this would not generalize well. As shown in table 7. the tree has 81 leaves, which shows that the algorithm learns too many rules.

If we look at the results of KNN in table 8 then we see that the results of KNN and Decision tree are similar for the win rule. For the legal move rule KNN is faster, but also has poorer results. For the number of k 15 is used. This seemed like the most effective number although a more extensive test should be done to confirm this.

| Dataset | Depth | Leaves | Time (ms) | Accuracy | Recall | Precision |
|---|---|---|---|---|---|---|
| MLWin1 | 11 | 81 | 302 | 1 | 1 | 1 |
| MLWin2 | 15 | 72 | 434 | 0.998 | 0.976 | 0.979 |
| MLLegal1 | 31 | 2625 | 18791 | 0.743 | 0.721 | 0.691 |
| MLLegal2 | 40 | 2313 | 12639 | 0.767 | 0.739 | 0.721 |

**Table 7.** Results of Decision Tree algorithm on 5000 games of Tic Tac Toe. For the win datasets this are 5000 entries and for the legal move this are 9574. The testsize is 0.1.

| Dataset | Time (ms) | Accuracy | Recall | Precision |
|---|---|---|---|---|
| MLWin1 | 575 | 0.936 | 0.921 | 0.897 |
| MLWin2 | 87 | 0.964 | 0.974 | 0.952 |
| MLLegal1 | 70 | 0.713 | 0.556 | 0.595 |
| MLLegal2 | 1000 | 0.763 | 0.664 | 0.713 |

**Table 8.** Results of KNN algorithm. With k=15 and testsize = 0.1.

| Algorithm | Accuracy | Precision | Recall | Nr games | Name |
|---|---|---|---|---|---|
| ProbFOIL$^+$ | 1 | 1 | 1 | 1000 | ProbWin5 |
| Decision Tree | 1 | 1 | 1 | 5000 | MLWin1 |
| KNN | 0.964 | 0.974 | 0.952 | 5000 | MLWin2 |

**Table 9.** Comparison of the best results of the win rule for each algorithm.

| Algorithm | Accuracy | Precision | Recall | Nr games | Name |
|---|---|---|---|---|---|
| ProbFOIL$^+$ | 0.977 | 0.111 | 0.047 | 25 | ProbLegal7 |
| Decision Tree | 0.767 | 0.739 | 0.721 | 5000 | MLLegal2 |
| KNN | 0.763 | 0.713 | 0.664 | 5000 | MLLegal2 |

**Table 10.** Comparison of the best results of the legal move rule for each algorithm.

| Algorithm | Time | Nr games | Dataset |
|---|---|---|---|
| ProbFOIL$^+$ | 4.5 h | 1000 | ProbWin5 |
| Decision Tree | 302 ms | 5000 | MLWin1 |
| KNN | 575 ms | 5000 | MLWin1 |

**Table 11.** Comparison of the computation times of the algorithms in correlation with the number of games.

### 4.2. Comparison

In this section we will compare the results of each algorithm. In table 9 an overview is given of the best results of the win rule. Both ProbFOIL$^+$ and the DT algorithm succeed in learning the win rule. The ProbFOIL$^+$ does succeed with a smaller dataset and is therefore the most data efficient. In table 10 we can see that none of the algorithms succeed in learning the legal move rule, although the DT algorithm has the best overall results. ProbFOIL$^+$ has a very low recall and precision and has the worst overall performance. In terms of scalability we can see in table 11 that ProbFOIL$^+$ needs about 4.5 hours to compute the rules from 1000 games whereas the non-symbolic AI algorithms need less than a second to compute the rules from 5000 games.

## 5. Discussion and Future Work

As described in the results in chapter 4 the win rule is learned successfully, but the legal move rule is not. The results are in line with the problems as described in the related work in chapter 2. ProbFOIL$^+$

struggles with scalability. DT and KNN struggle with generalization and interpretability. For KNN this can be improved slightly by plotting the clusters that are formed, of course this is not as informative as actual rules. For the DT algorithm rules could be formed from the text document. Due to the overfitting and for more complex rules it would be a complicated task to form rules from the output. In the context of creating systems that can make sense of games by themselves the importance of interpretability of the learned rules is that we can verify whether the system learned the right rules and that the system is able to communicate its knowledge. This does seem to be the crucial advantage of symbolic AI.

There are several ways in which the results could be improved. A mistake that is made while testing ProbFOIL$^+$ with negative examples, is that the auto mode was still on. Therefore it is hard to say what the exact effect is of the added negative examples in the datasets since ProbFOIL$^+$ also automatically added negative examples. Therefore it would be useful to test all the datasets without the auto mode on. Another option that might improve the research is testing different parameters settings such as beam size, values of k, different test and train sizes.

In this thesis ProbFOIL$^+$ is used as a deterministic rule learner. This is not the strongest point of ProbFOIL$^+$, since the key element of the program is being able to handle probabilistic datasets (De Readt et al., 2015). The probability handling can be useful when learning games with probabilistic factors. Still other ILP programs might be more suited for the purpose of learning game rules by observing game play. A recent improvement on ProbFOIL$^+$ is the program Safelearner (Jain et al., 2019). Safelearner scales better than ProbFOIL$^+$ by using lifted probabilistic inference instead of using grounding. Björnsson (2012) also did research for a possible solution to the scaling problems of ILP systems, by learning Deterministic Finite Automata instead of logical rules. His results did improve in speed, but this approach does needs to be further researched before it can effectively learn complicated board game rules by observing (Björnsson, 2012).

The ability to handle noise in data is a factor that is not tested in this thesis. Since ProbFOIL$^+$ is supposed to be better at other ILP systems at handling noise it would be interesting to compare the performance of ProbFOIL$^+$ with noisy datasets to statistical machine learning techniques (De Readt et al., 2015).

It would be interesting to see how the tested algorithms perform on learning different games. The IGGP dataset from Cropper et al. (2019) could be very useful for this purpose. Since the IGGP datasets are written in Prolog it is easy to reconstruct them to facts for datasets that can be used for ProbFOIL$^+$. Only the settings for each game need to be added. In section 2.1 is described how Cropper et al. (2019) tested several algorithms on the IGGP dataset. I think that adding ProbFOIL $^+$, Safelearner and DT to the research might yield new useful insights.

As said in the background chapter a disadvantage of ILP systems is that datasets have to be handcrafted, which is also the case for the ProbFOIL$^+$ algorithm. Although the datasets for KNN and DT are also handcrafted so in this case it does not make a difference.

Both symbolic and non-symbolic AI have their own difficulties when trying to make sense of game rules. The most promising solution lies in combining the two types of AI, since the disadvantages of modern statistical AI could be solved by the advantages of symbolic AI and vice versa (Garnelo and Shanahan, 2019). The related work in this field contains: the Apperception Engine (Evans et al.,

2019), DeepProblog (Manhaeve et al., 2018) and Differentiable Inductive Logic Programming (Evans and Grefenstette, 2018).

For this thesis the rules that we want to learn had to be given to the program, the win rule and the legal move rule. This is of course a bias and not completely letting the program learn the game by itself. Therefore it would be interesting if future research would focus on finding a way to learn the rules without explicitly stating the rules that have to be learned.

# 6. Conclusion

With the algorithms used in this thesis it is possible to learn simple rules such as the win rule of Tic Tac Toe. However for more complicated game rules the disadvantages of the tested algorithms quickly come to light. These disadvantages are a good representation of the difference in abilities between symbolic and non-symbolic AI. In the case of Tic Tac Toe ProbFOIL$^+$ has slightly more advantages compared to KNN and DT. In both fields there are more advanced algorithms available, although there is no algorithm yet that is completely capable of learning all the game rules of the IGGP dataset. This is an interesting challenge that could help the advancement of AI. Hopefully this thesis has contributed with more information about the abilities of ProbFOIL$^+$ in comparison with KNN and DT. The results imply that ILP might still be a useful approach for systems that can make sense of games. The combination of symbolic and non-symbolic AI seems to be the most promising solution for developing such systems.

# 7. References

https://git.science.uu.nl/s.v.teekens/bsc_thesis_teekens


Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H. T. (2012). *Learning from data* (Vol. 4). New York, NY, USA: AMLBook.

Björnsson, Y. (2012, August). Learning Rules of Simplified Boardgames by Observing. In *ECAI* (pp. 175-180).

Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. J. (1984). Classification and Regression Trees. Wadsworth & Brooks/Cole Advanced Books & Software. *Pacific California*.

Cropper, A., Dumančić, S., & Muggleton, S. H. (2020). Turning 30: New ideas in inductive logic programming. *arXiv preprint arXiv:2002.11002*.

Cropper, A., Evans, R., & Law, M. (2019). Inductive general game playing. *Machine Learning*, 1-42.

Cropper, A., & Muggleton, S. H. Metagol system (2016). https://github.com/metagol/metagol.

De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., & Verbeke, M. (2015, June). Inducing probabilistic relational rules from probabilistic examples. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

De Raedt, L., & Thon, I. (2010, June). Probabilistic rule learning. In *International conference on inductive logic programming* (pp. 47-58). Springer, Berlin, Heidelberg.

Džeroski, S., Cestnik, B., & Petrovski, I. (1993). Using the m-estimate in rule induction. *Journal of computing and information technology*, *1*(1), 37-46.

Evans, R., Hernández-Orallo, J., Welbl, J., Kohli, P., & Sergot, M. (2019). Making sense of sensory input. *arXiv preprint arXiv:1910.02227*.

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

Genesereth, M., Love, N., & Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI magazine*, *26*(2), 62-62.

Garnelo, M., & Shanahan, M. (2019). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, *29*, 17-23.

Jain, A., Friedman, T., Kuzelka, O., Van den Broeck, G., & De Raedt, L. (2019). Scalable Rule Learning in Probabilistic Knowledge Bases. In *The 1st Conference On Automated Knowledge Base Construction (AKBC)*.

Love, N., Hinrichs, T., Haley, D., Schkufza, E., & Genesereth, M. (2008). General Game Playing: Game Description Language Specification (Technical Report No. March 4 2008).

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., & De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems* (pp. 3749-3759).

Muggleton, S. (1995). Inverse entailment and Progol. New Generation Computing, 13(3&4), 245–286.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, *5*(3), 239-266.

Russell, S., & Norvig, P. (2010). Artificial intelligence: a modern approach (Third edition). Essex, England: Pearson.

Spark, A. (2018). Apache spark. *Retrieved January*, *17*, 2018.

# Appendix

## A.1 Abbreviations

AAAI - Association for the Advancement of Artificial Intelligence

AI - Artificial Intelligence

ASP - Answer Set Programming

DFA - Deterministic Finite Automata

DT - Decision Tree

FOL - First Order Logic

GDL - Game Description Language

GGP - General Game Playing

ILP - Inductive Logic Programming

KNN - k-Nearest-Neighbors

MIL - Meta-Interpretive Learning

## A.2 Learned win rules ProbFOIL$^{+}$

ProbWin1:

```
win(A,B) :- cell11(A,B), cell31(A,B), cell21(A,B), \+cell13(A,B)
win(A,B) :- cell11(A,B), cell22(A,B)
win(A,B) :- cell12(A,B), cell11(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), \+cell33(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell32(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
```

ProbWin2:

```
win(A,B) :- cell11(A,B), cell31(A,B), cell21(A,B), \+cell13(A,B)
win(A,B) :- cell11(A,B), cell22(A,B)
win(A,B) :- cell12(A,B), cell11(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), \+cell33(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell32(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
```

ProbWin3:

```
win(A,B) :- cell11(A,B), cell31(A,B), cell21(A,B), \+cell13(A,B)
win(A,B) :- cell11(A,B), cell22(A,B)
win(A,B) :- cell12(A,B), cell11(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), \+cell33(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell32(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
```

ProbWin4:

```
win(A,B) :- cell11(A,B), cell31(A,B), cell21(A,B), \+cell13(A,B)
win(A,B) :- cell11(A,B), cell22(A,B)
win(A,B) :- cell12(A,B), cell11(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), \+cell33(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell32(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
```

ProbWin5:

```
win(A,B) :- cell11(A,B), cell21(A,B), cell31(A,B)
win(A,B) :- cell11(A,B), cell12(A,B), cell13(A,B)
win(A,B) :- cell22(A,B), cell11(A,B), cell33(A,B)
win(A,B) :- cell22(A,B), cell12(A,B), cell32(A,B)
win(A,B) :- \+cell32(A,B), \+cell33(A,B), \+cell23(A,B), \+cell12(A,B), \+cell22(A,B), \+cell21(A,B), \+cell13(A,B)
win(A,B) :- cell31(A,B), cell13(A,B), cell22(A,B)
win(A,B) :- cell22(A,B), cell21(A,B), cell23(A,B)
win(A,B) :- cell33(A,B), cell31(A,B), cell32(A,B)
win(A,B) :- cell13(A,B), cell23(A,B), cell33(A,B)
```

### A.3 How to use ProbFOIL$^+$

The input of ProbFOIL$^+$ requires settings followed by the data. These are facts that are represented as follows:

learn(predicate/arity)  %Target: predicate we want to learn

mode(predicate(mode1,mode2,..) % modeX is the specifier. Modes: which predicates can be added to the rules. There are three possible specifiers for the mode:

- + : the variable at this position must already exist when the literal is added
- - : the variable at this position does not exist yet in the rule
- c : a constant should be introduced here

Types: type information for the predicates

base(predicate(type1, type2, ..) type can be identified by arbitrary Prolog atoms (e.g. person, a, etc.)

Negative examples can be defined by adding zero-probability, for example: 0::Win(x,1).

Another possibility is to let ProbFOIL$^+$ generate negative examples automatically by adding the fact: example_mode(auto).