Co-operative genetic evolution of logic rules

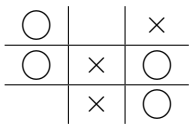
white paper

October 13, 2019

系统分为两部分：

- 逻辑规则引擎 (logic rules engine)
- 基因算法 逻辑规则学习器 (genetic inductive learning of logic rules)

初步 测试 是「打井游戏」(tic-tac-toe)，例如：

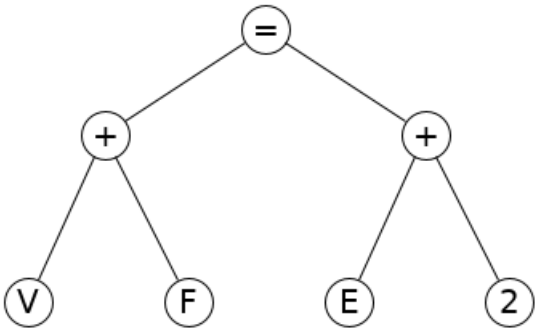


(1)

1 Genetic learning of rules

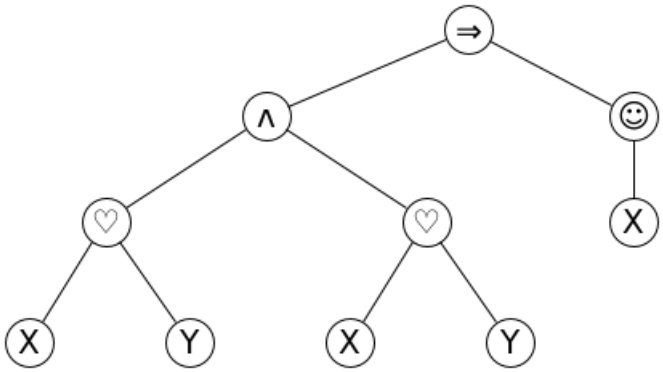
1.1 What is a logic rule?

例如，数学式子可以表示成 tree, $V + F = E + 2$：



(2)

逻辑 rule 必然包含 $\Rightarrow$，可以将 rule 分成 **head** 和 **tail** 两部分：

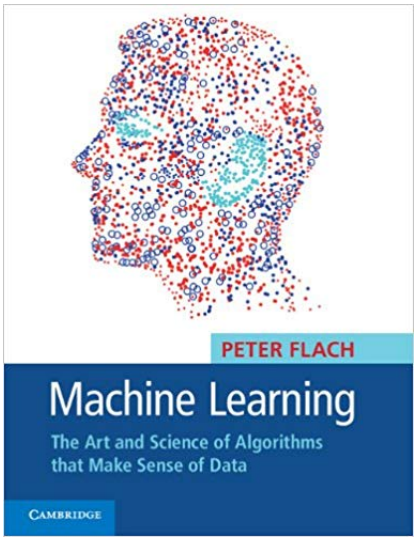


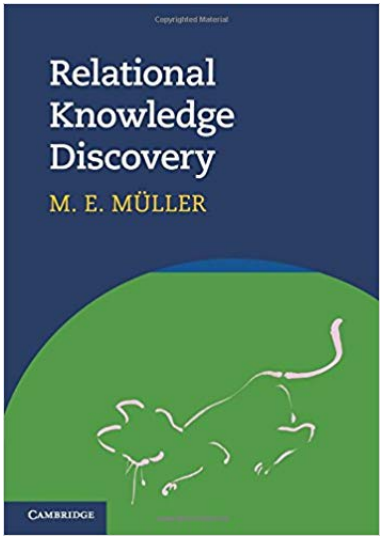
(3)

$$\forall X, Y. \quad \underbrace{X \heartsuit Y \wedge Y \heartsuit X}_{\text{head}} \Rightarrow \underbrace{\text{☺} X}_{\text{tail}} \tag{4}$$

（这些可以在数学上严格定义，在论文中会详细定义）

Logic rule 的学习，即 inductive rule learning 又叫 inductive logic programming (ILP). 有基础的书介绍，例如：





(5)

可以上 https://b-ok.org/ 下载。

Inductive logic learning 的参考书还有：

- *Logical and relational learning* [L. D. Raedt 2008]
- *Inductive logic programming - from machine learning to software engineering* [Bergadano and Gunetti 1996]
- *Logic for learning* [Lloyd 2003]
- *Simply logical: intelligent reasoning by example* [Flach 1994]
- *Foundations of rule learning* [Fürnkranz, Gamberger, and Lavrač 2012]
- *Latest advances in inductive logic programming* [Muggleton and Watanabe 2015]
- *Statistical relational aritificial intelligence* [Raedt et al. 2016]

- *Learning language in logic* [Cussens and Džeroski 2000]
- *Relational data mining* [Džeroski and Lavrač 2001]
- *Mathematical aspects of logic programming semantics* [Hitzler and Seda 2011]

这本身是一门颇深奥的学问，需要一段时间学习，但暂时不看也罢，只需要知道它是一种 combinatorial search，在 logic rules 的 lattice（格）中搜寻。这是一种 离散的、符号的空间。

1.2 Genetic algorithm

要应用 GA 很容易，只需适当地 定义 cross-over 和 mutation.

Cross-over：随机地选取一个节点，将两个式子在这点交叉。

Mutation：随机选择一个节点，在这个节点下换上一棵「随机树」(random tree)

1.3 协同进化 (co-operative co-evolution)

这部分是新的。但它意思很明显，就是说：进化的目标不只是一个最优的 个体，而是整个 群族。

The idea is first described in [Freitas 2002]:

6 Genetic Algorithms for Rule Discovery

"We actually go to the extreme of using the problem space as the working search space."
[Janikow 1993, p. 200]

In this chapter we discuss several issues related to developing genetic algorithms (GAs) for prediction-rule discovery. The development of a GA for rule discovery involves a number of nontrivial design decisions. In this chapter we categorize these decisions into five groups, each of them discussed in a separate section, as follows.

Section 6.1 discusses issues related to individual representation. Section 6.2 discusses several task-specific "genetic" operators, i.e., operators developed specifically for a given data mining task, or a class of related data mining tasks. Sections 6.3 and 6.4 discuss task-specific population-initialization and rule-selection methods, respectively. Finally, section 6.5 discusses fitness evaluation.

In general the operators and methods discussed in this chapter were developed for the classification task (section 2.1), but they can also be used in other data mining tasks involving the discovery of prediction rules, such as the dependence modeling task (section 2.2). These operators and methods can be regarded as a form of incorporating task-specific knowledge into the GA, or adapting the GA for the discovery of prediction rules. (6)

6.1 Individual Representation

This section is divided into three parts. Subsection 6.1.1 discusses the pros and cons of two broad approaches (Michigan and Pittsburgh) for encoding rules into a GA individual. Subsections 6.1.2 and 6.1.3 discuss how to encode a rule antecedent and a rule consequent, respectively, into a GA individual.

6.1.1 Pittsburgh vs Michigan Approach

In conventional GAs each individual corresponds to a candidate solution to a given problem. In our case the problem is to discover prediction rules. In general, we are interested in discovering a set of rules, rather than a single rule. So, how can we encode a set of rules in a GA population? In essence there are two approaches.

First, if we follow the conventional GA approach, each individual of the GA population represents a set of prediction rules, i.e., an entire candidate solution. This is called the Pittsburgh approach.

Another approach, which departures from conventional GAs, consists of having an individual represent a single rule, i.e., a part of a candidate solution. This is called the Michigan approach. In the literature the term Michigan approach is often used to refer to classifier systems [Goldberg 1989, chap. 6], which is one specific kind of evolutionary algorithm where each individual represents a rule. In this book we use the term Michigan approach in a broader sense, to refer to any kind of evolutionary algorithm for rule discovery where each individual represents a single prediction rule. (7)

In the Michigan approach there are at least two possibilities for discovering a set of rules. The first one is straightforward. We let each run of the GA discover a single rule (the best individual produced in all generations) and simply run the GA multiple times to discover a set of rules. An obvious disadvantage of this strategy is that it is computationally expensive, requiring many GA runs. The second possibility is to design a more elaborate GA where a set of individuals – possibly the whole population – corresponds to a set of rules. In the remainder of this subsection we assume the use of this kind of "elaborate" version of the Michigan approach.

It is important to understand the advantages and disadvantages of the Pittsburgh and Michigan approaches. The first step for this understanding is to bear in mind that these two approaches cope with the problem of *rule interaction* in different ways. The problem of rule interaction consists of evaluating the quality of a rule set as a whole, rather than just evaluating the quality of each rule in an isolated manner. In other words, the set of *best rules* is not necessarily the *best set* of rules.

Two very early papers using genetic algorithm to learn logic rules are: [Giordana, Saitta, and Zini 1994] and [Augier, Venturini, and Kodratoff 1995]. There is also [Liu and Kwok 2000] which is an improvement over the SIA approach used in the above 2 papers.

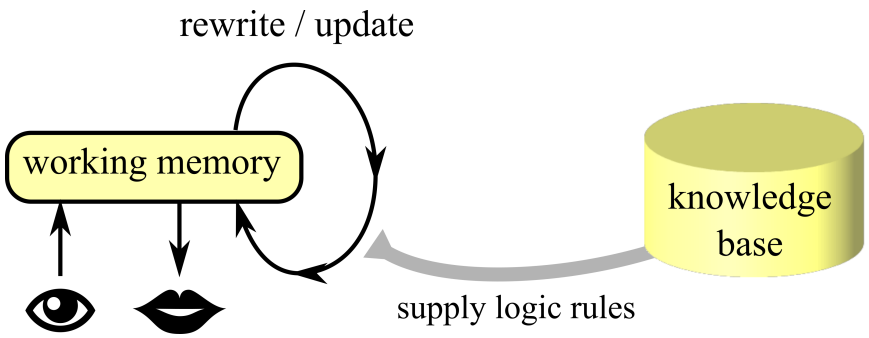
"A genetic algorithm for text classification rule induction" [Pietramala et al. 2008] uses genetic algorithm to learn rules for document classification. However, it uses only simple, propositional rules over n-grams.

One problem of genetic algorithms is that the genes represent **discrete** entities, so that crossing-over and mutations usually result in discrete **jumps** from one individual to another. This is in contrast to deep learning where gradient descent is continuous. Perhaps one way to circumvent this problem is to evolve a massive number of rules, and let the action of the intelligent agent be the **average** of a number of fired rules. We will explore this idea.

2 Logic rule engine

其实这部分和 genetic algorithm 或 learning 都无关，但 logic rules 必需靠 逻辑引擎 运行 才能发生作用。

逻辑 AI 引擎 的基本运作 如下：

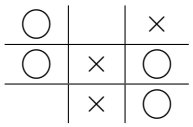


(8)

举例来说，例如打井游戏，游戏的 **状态** 是由一些 逻辑命题 描述：

$$
\begin{aligned}
&X(0,2)\\
&X(1,1)\\
&X(2,1)\\
&O(0,0)\\
&O(1,0)\\
&O(1,2)\\
&O(2,2)\\
&\square(0,1)\\
&\square(2,0)
\end{aligned}
\tag{9}
$$

表示这状态：



(10)

一条 logic rule 可以是这样：

$$X(x,y) \wedge X(w,y) \wedge \neq (x,w) \Rightarrow \text{ColumnWin}(x,y) \tag{11}$$

表示如果有两个 X 在同一直行，则这一行差一步就可以赢。

其实我也不知道 打井游戏 的正确 rules 是怎样，这要等 learn 出来之后看看。

在 (11) 式里我用了 $\neq$ 这个 关系 (relation) 或 谓词 (predicate)，其实是要额外定义的 (externally defined)。详细来说这就像一种 programming language.

用读者的想像力脑补一下，可以明白到，这种逻辑引擎，基本上是可以解决任何问题的。

2.1 Rete algorithm

（Rete 在 拉丁文的意思是「网状」）

其实这个算法也和学习无关，它只是 逻辑引擎 的一个 **加速** 算法，大部分 实际使用的逻辑引擎 都需要 Rete 的加速。

基本上，rete 算法将 logic rules 分拆、重组成 **树** 的结构：

$$\boxed{\text{logic rules}} \hookrightarrow \boxed{\text{树}}. \tag{12}$$

这 tree 的意思和 decision tree 差不多。

逻辑引擎 运行时，从 input 输入一些 命题，进入 working memory. 这些 命题 传统上叫作 **WME** (working memory elements). 於是要将 WME 和 知识库中的 logic rules 逐一比较，看看哪条 rule 能够 apply，这过程称为 matching. Rete 的作用是加速这 matching.

Rete 的 decision tree，输入是一个新的 WME，从树的 根 出发，比较有没有 match，如果有则记忆在树的节点里，如果没有则继续往下搜寻。

举个例：「爱一个人但那人不爱你，则不开心」

$$\forall X, Y. \quad \underbrace{X \heartsuit Y \wedge \neg Y \heartsuit X}_{\text{head}} \Rightarrow \underbrace{\dot{\frown} X}_{\text{tail}} \tag{13}$$

在 rete 的树上会有节点检查 有没有 $\heartsuit$ 和 $\neg\heartsuit$ 这两个 WME，如果两个节点都通过，则有一链结会通往 结论的「激活」(fire).

当然 rete 这个 tree 可以有不同的构造方法，类似 decision tree，例如根据 information gain.

由於 rete 是树状结构，它是 hierarchical（层级化）的。我们可以利用 rete 的层级化，将 进化算法 变成 层级化的。这会是论文最大的贡献。

3 Project status

Our code is on GitHub:
https://github.com/Cybernetic1/GILR

For the Rete algorithm, we adopted a simple open-source implementation called NaiveRete (written in Python) on GitHub. NaiveRete is based on the PhD thesis of Doorenbos 1995. When we fed randomly generated logic formulas into NaiveRete, we discovered some bugs and fixed them.

The Rete algorithm is notoriously difficult to understand and implement. As a result, it is very difficult to conduct experiments in genetic learning of **first-order logic** formulas, which require an efficient logic engine to evaluate. This may explain why this topic is relatively unexplored.

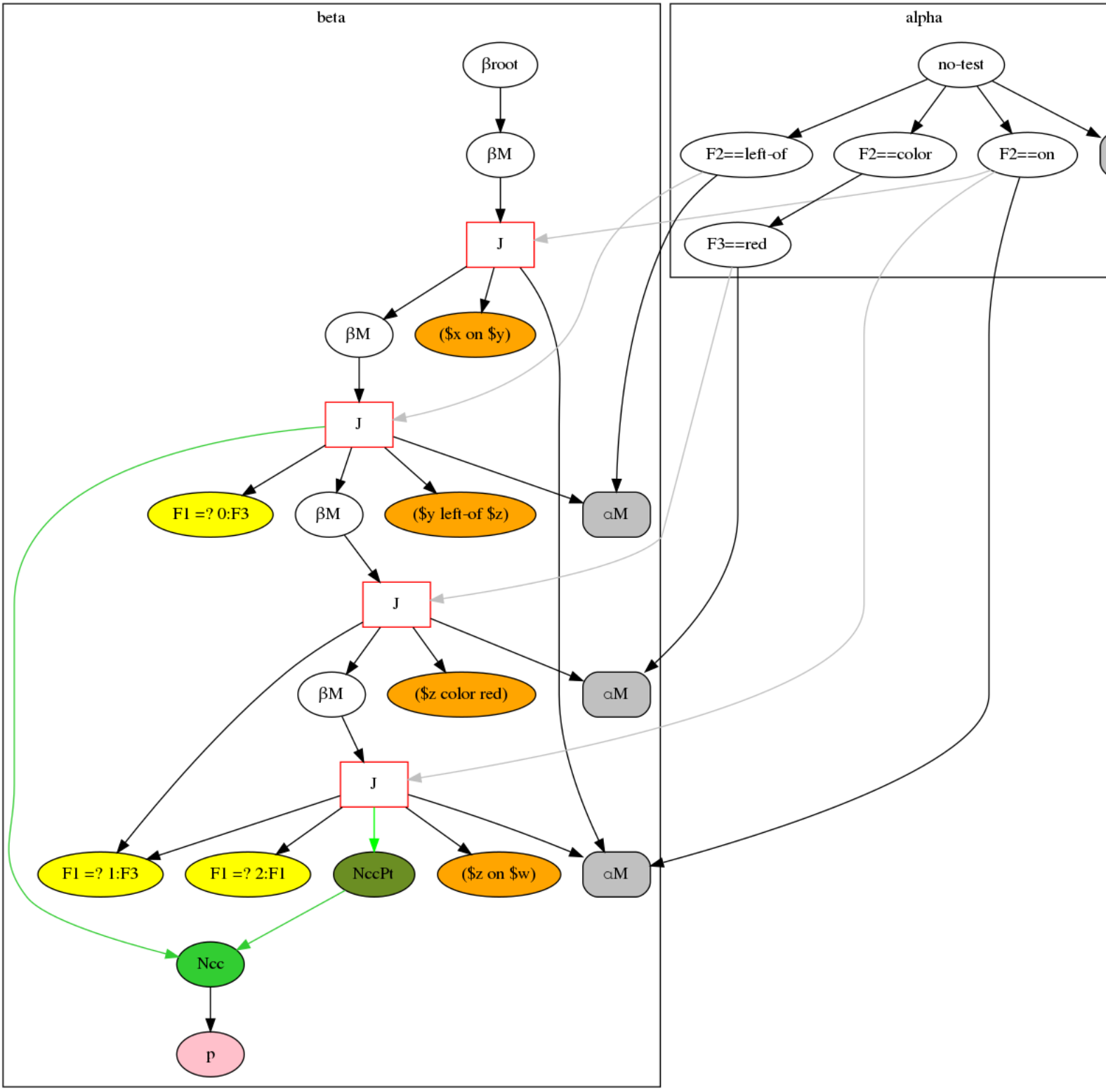
This is a Python example for setting up Rete and invoking it:

genetic-programming.py × genifer-lovers.py ×

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import os

from rete.common import Has, Rule, WME, Neg, Ncc
from rete.network import Network

net = Network()

c1 = Has('male', '$a')
c2 = Has('love', '$a', '$b')
c3 = Has('female', '$b')

# net.add_production(Rule(Ncc(c1, Ncc(c2, c3))))
# net.add_production(Rule(Ncc(c2, Ncc(c3))))
# net.add_production(Rule(c1, Ncc(c2)))
# net.add_production(Rule(c1, Ncc(c2, c3)))
# net.add_production(Rule(c2, c3))
p0 = net.add_production(Rule(c3, Ncc(c2, c1)))

wmes = [
    WME('female', 'Mary'),
    WME('female', 'Ann'),
    WME('love', 'John', 'Pete'),        # 基
    WME('love', 'John', 'John'),        # 自恋
    WME('love', 'Pete', 'Mary'),        # 所谓正常
    WME('love', 'Pete', 'John'),        # 互基
    WME('love', 'Mary', 'Ann'),         # Lesbian
    WME('male', 'John'),
    WME('male', 'Pete'),
]
for wme in wmes:
    net.add_wme(wme)

```

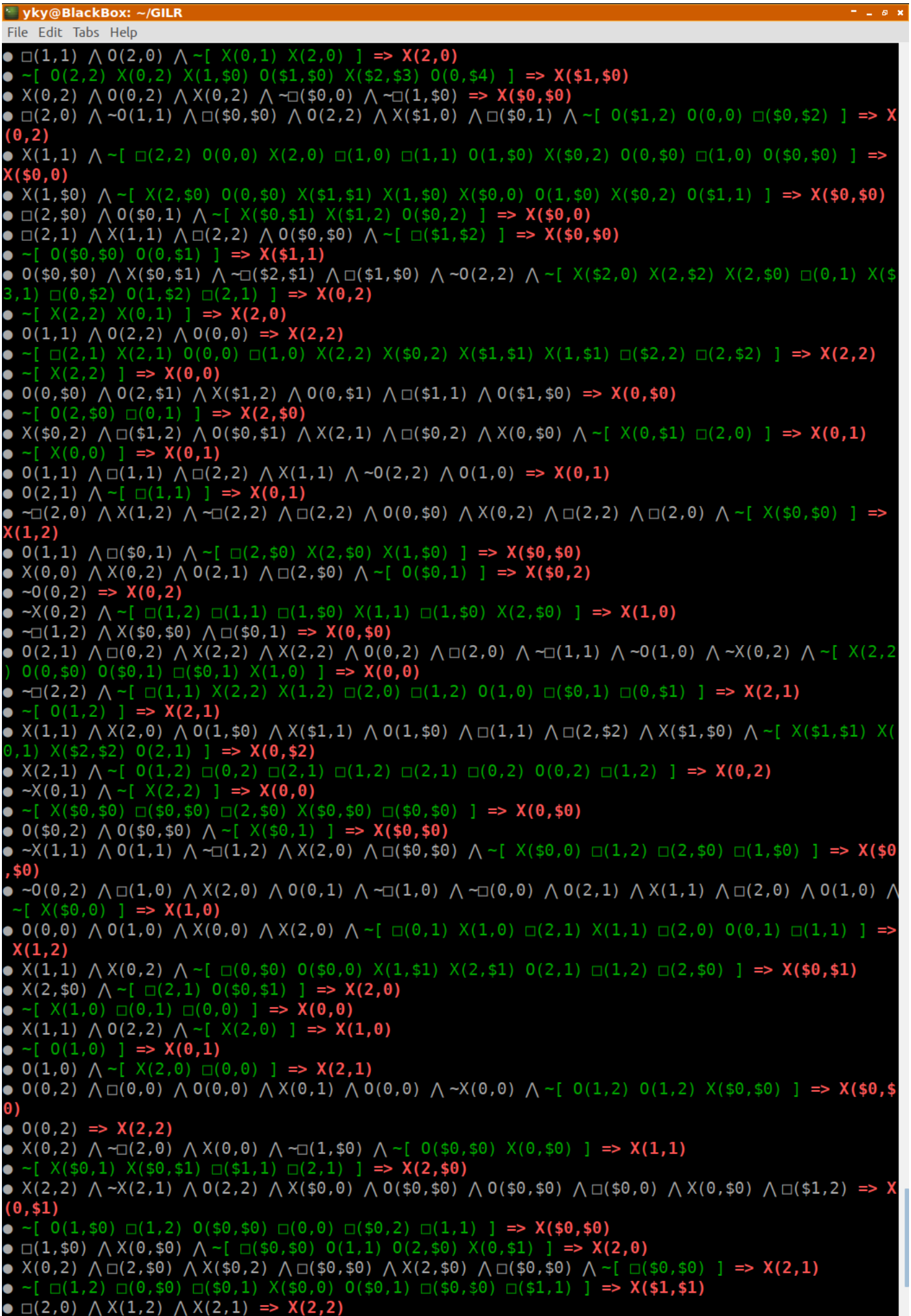
(14)

The Rete network gets very big as more rules are added to it. This is an example Rete network for only 1 rule:



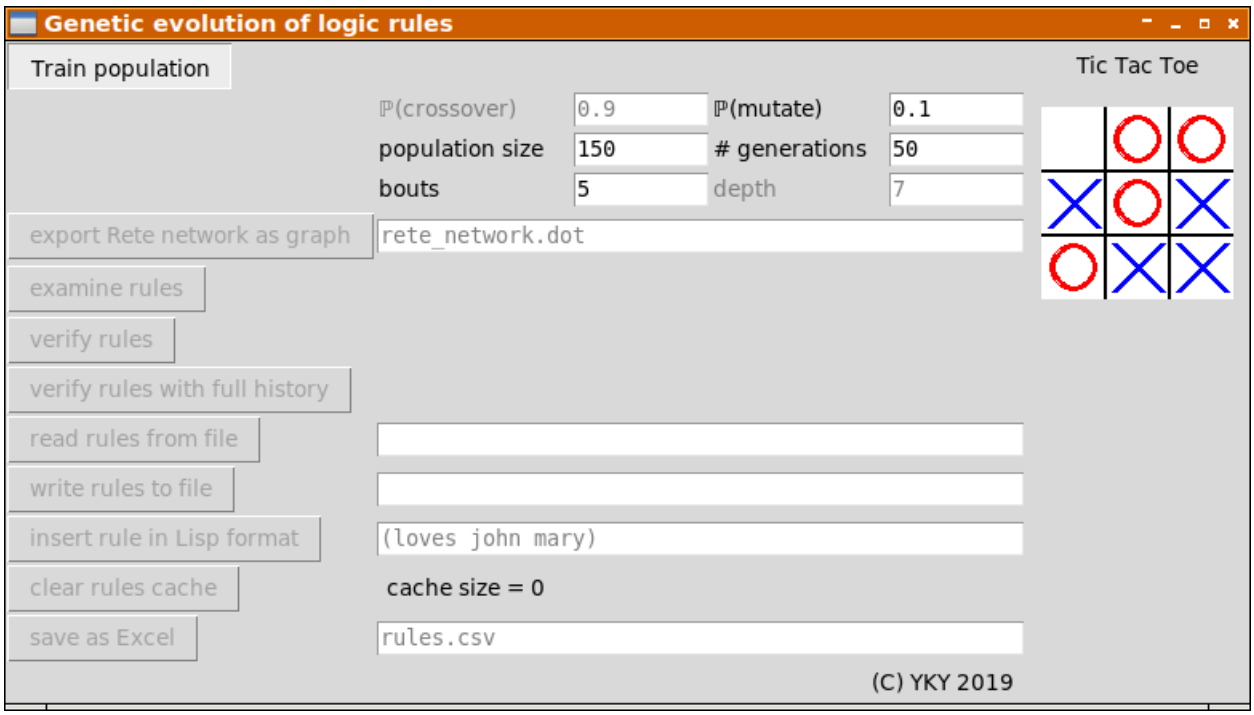
(15)

This is a screenshot of the randomly generated logic rules for Tic-Tac-Toe:



(16)

This is the GUI for the Tic-Tac-Toe learner:



(17)

Currently the genetic algorithm fails to converge for Tic-Tac-Toe. We think this is probably because the rules are "flat" in the sense that each rule looks at the board configuration and immediately concludes with a game move. This class of rules may be inadequate for playing the game. We think that adding **predicate invention** and **multi-step reasoning** would allow this game to be solved perfectly.

Then we will apply this algorithm on more challenging tasks such as natural language processing.

References

Augier, Venturini, and Kodratoff (1995). "Learning first order logic rules with a genetic algorithm". In: *KDD-95 Proceedings*.

Bergadano and Gunetti (1996). *Inductive logic programming - from machine learning to software engineering*. MIT.

Cussens and Džeroski, eds. (2000). *Learning language in logic*. Springer.

Doorenbos, Robert (1995). "Production matching for large learning systems". PhD thesis. Carnegie Mellon Univ.

Džeroski and Lavrač, eds. (2001). *Relational data mining*. Springer.

Flach (1994). *Simply logical: intelligent reasoning by example*. John Wiley.

Freitas (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer.

Fürnkranz, Gamberger, and Lavrač (2012). *Foundations of rule learning*. Springer.

Giordana, Attilio, Lorenza Saitta, and Floriano Zini (1994). "Learning disjunctive concepts with distributed genetic algorithms". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. IEEE, pp. 115–119.

Hitzler and Seda (2011). *Mathematical aspects of logic programming semantics*. CRC Press.

Liu, J Juan and J Tin-Yau Kwok (2000). "An extended genetic rule induction algorithm". In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*. Vol. 1. IEEE, pp. 458–463.

Lloyd (2003). *Logic for learning*. Springer.

Muggleton and Watanabe, eds. (2015). *Latest advances in inductive logic programming*. Imperial College Press.

Pietramala et al. (2008). "A genetic algorithm for text classification rule induction". In: *Lecture notes in computer science v5212*.
Raedt, de et al. (2016). *Statistical relational aritificial intelligence*. Morgan & Claypool.
Raedt, Luc De (2008). *Logical and relational learning*. Springer.