GILR – genetic induction of logic rules

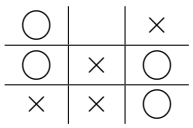
white paper

September 5, 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：

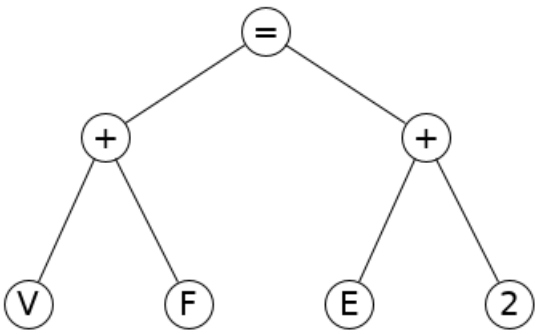


(1)

1 Genetic learning of rules

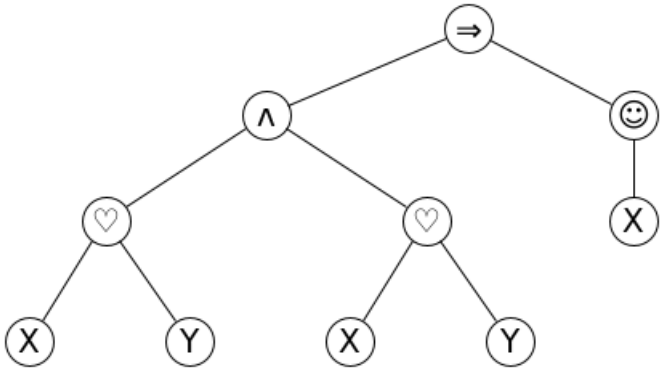
1.1 What is a logic rule?

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：

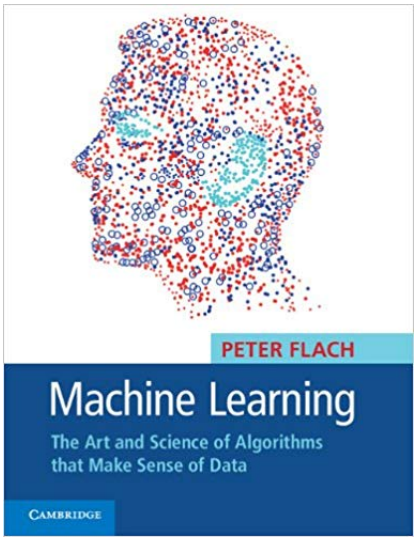


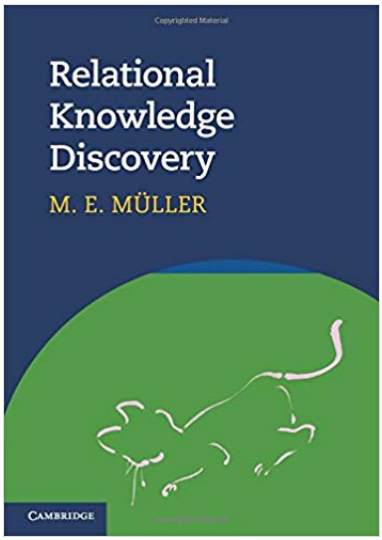
(3)

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

（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。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)

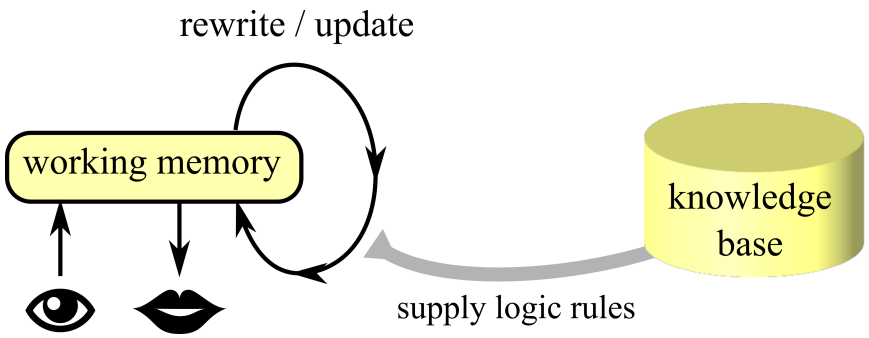
1.3 协同进化 (co-operative co-evolution)

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。

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：



(6)

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：

$$\begin{gathered} X(0,2) \\ X(1,1) \\ X(2,0) \\ X(2,1) \\ O(0,0) \\ O(1,0) \\ O(1,2) \\ O(2,2) \\ \square(0,1) \end{gathered} \tag{7}$$

表示这状态：

○		×
○	×	○
×	×	○

(8)

一条 logic rule 可以是这样：

$$X(x,y) \wedge X(w,y) \wedge \neq (x,w) \Rightarrow \text{ColumnWin}(,) \tag{9}$$

表示如果有两个 X 在同一直行，则这一行可以赢。

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 (9) 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：

$$\boxed{\text{logic rules}} \hookrightarrow \boxed{\text{树}}. \tag{10}$$

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」

$$\forall X, Y. \quad \underbrace{X \heartsuit Y \wedge \neg Y \heartsuit X}_{\text{head}} \Rightarrow \underbrace{\text{☹} X}_{\text{tail}} \tag{11}$$

在 rete 的树上会有节点检查 有没有 ♡ 和 ¬♡ 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。