Lecture7 逻辑智能体

1. 基于知识的智能体

逻辑 AI

> The idea is that an agent can **represent knowledge of its world**, its goals and the current situation **by sentences in logic** and **decide what to** do by **inferring** that a certain action or course of action is appropriate to achieve its goals.
>
> —— John Mcarthy in Comcepts of logic AI, 2020

- 智能体需要关于世界的“**知识 knowledge**”，来帮助它做出好的决策
- **知识 knowledge** = 一组语句（知识库），一种表达知识的形式化语言
- **语句 sentence** = 关于这个世界的一个定义
- 一个基于知识的智能体由下面两个部分组成
  - **知识库**：特定范围的内容（用语句表达）
  - **推断机制**：特定范围独立的算法

介绍

一个**基于知识的智能体 Knowledge-based Agent KB** 必须能够

- 表达状态、行动等
- 获得新的预测
- 更新对于世界的表达
- 推断世界隐藏的特征
- 推断正确的行为

使用**声明式**的方法来建立一个智能体

- 添加新的语句：**Tell**（告诉它它需要知道的东西）
- 查询已知的：**Ask**（问它自己该做些什么，答案应该遵循 KB）

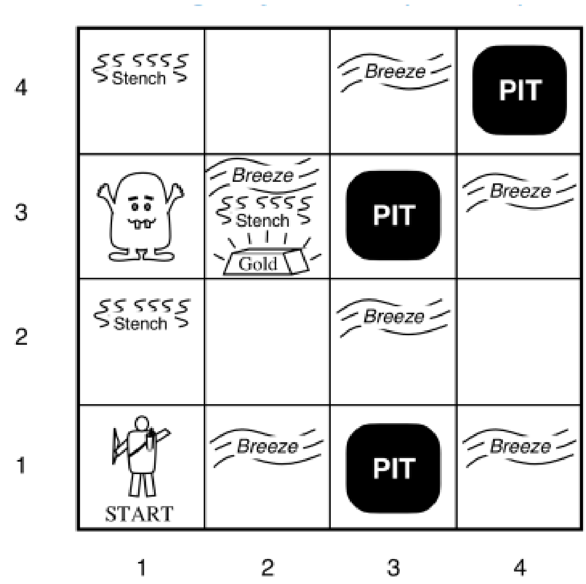
```
function KB-Agent(percept) return an action
    KB, a knowledge base
    t, a counter, initial 0, indicating time

    TELL(KB, Make-Percept-Sentence(percept, t)) // 让KB更新，扩大知识库
    action ← ASK(KB, Make-Action-Query(t)) // 让KB对t做推理
    TELL(KB, Make-Action-Sentenct(action, t)) // KB把推理出来的结果加入知识库
    t ← t + 1
    return action
```

2. 示例 Wumpus World

游戏介绍

游戏规则



有一个 4x4 方格状的地图，现在智能体从 START 位置出发，进行搜索

- **怪兽格子 Wumpus**：如果智能体走到了有怪兽的位置，智能体会被吃掉，游戏失败
  - 怪兽邻近的四个格子会散发难闻的气味 Stench
- **无底洞格子 PIT**：如果智能体走到了无底洞，它会掉下去，游戏失败
  - 无底洞邻近的四个格子会有风 Breeze
- **奖励格子 GOLD**：如果智能体走到了奖励格子，它会获得奖励

我们想要智能体根据推断，不受伤地探索完整片区域

评分标准

- 获得 GOLD：+1000
- 死掉（被怪物吃掉或者掉进洞里）：-1000
- 每进行一次行动：-1
- 每使用一次弓箭：-10

游戏在智能体死亡或者离开洞穴结束

初始状态

- 4x4 的格子
- 智能体从格子 $[1,1]$ 出发，朝向右边
- GOLD 和怪物的位置是随机选择的，它们可以在不是 $[1,1]$ 的任何位置
- 每一个不是起始点的方格中，有洞穴的概率是 0.2

智能体分析

智能体有如下的 Action

- 向左转，向右转，向前走，向下拿东西，释放东西，射击

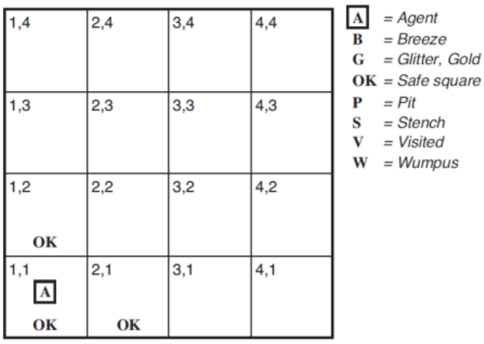
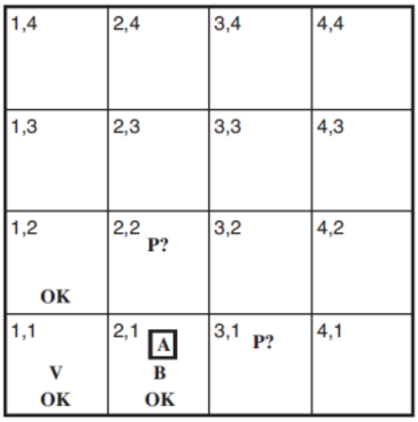
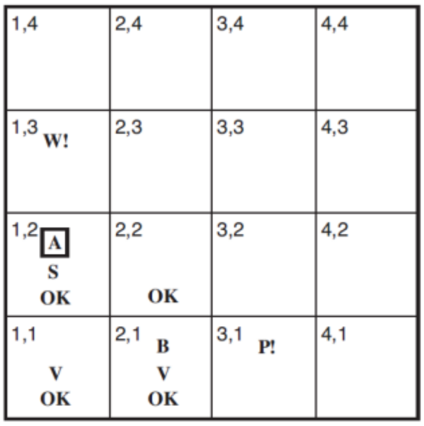
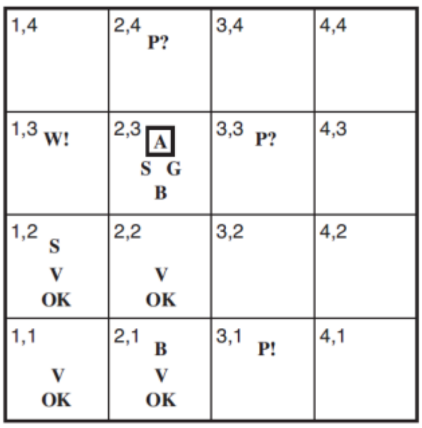
智能体有如下的传感器（表示成一个**二维的数组**）

- 感知臭味 Stench
- 感知风 Breeze
- Eg [Stench, Breeze]

Wumpus World 的特点

- 部分可观测
- 静态的（环境给定后不变）
- 离散的
- 单一智能体
- 确定性的
- 有顺序的

探索 Wumpus World

搜索	描述
	初始状态 没有 Breeze 和 Stench [1,2] 和 [2,1] 都是安全的
	智能体选择 [2,1] 在这里感受到了 Breeze -> [2,2] 和 [3,1] 可能有 PIT
	智能体回去选择 [1,2] 在这里感受到了 Stench -> [2,2] 是安全的，[1,3] 有怪物，[3,1] 有 PIT
	智能体选择 [2,2] 没有 Stench 或者 Breeze -> [2,3]，[3,2] 是安全的 智能体选择 [2,3] **在这里有 GOLD** 在这里感受到 Stench -> 是 [1,3] 的怪物 在这里感受到 Breeze -> [2,4] 和 [3,3] 可能有 PIT
	智能体已经找到 GOLD，原路返回即可

3. 命题逻辑 Propositional Logic

- **知识库 Knowledge Base**：一组按照规范的格式表达的语句集合
- **逻辑 Logics**：描述知识的规范的语言，用来提取结论
  - **语法 Syntax**：定义语言中结构良好的句子
  - **语义 Semantic**：定义语句的真值表或者含义
- **推断 Inference**：一种从其它的语句中推导出新的语句的过程
- **逻辑蕴含 Logical Entailment**：一种语句与语句之间的关系，它意味着一个语句从逻辑上跟随其它语句

$$KB \vDash \alpha$$

命题逻辑 PL 是最简单的逻辑

- **语法**：定义了允许使用的句子或命题
- **定义**：命题是一种陈述性的陈述，不是真就是假

示例

- 2 + 2 = 4 是一个表示为真的命题
- $W_{1,3}$ 是一个命题，如果有怪物在格子 [1,3]，那么它为真
- “如果 [1,2] 有臭味，那么怪物在 [1,3]” 是一个命题
- “How are you” 或者 “Hello” 不是命题一般来说，作为疑问、命令或意见的陈述句不是命题

语法

原子命题 Atomic Proposition

单一命题符号，每个符号都是一个命题，符号：大写字母，可包含下标

复合命题 Compound Proposition

由原子命题用**括号**和**逻辑连接词**构成

让 p, p_1, p_2 是命题

- **否定 Negation**：$\neg p$ 也是一个命题
  - 我们把一个原子命题和它的否定称为**文字 literal**
  - 例如 $W_{1,3}$ 是 positive literal，$\neg W_{1,3}$ 是 negative literal
- **合取 Conjunction**：$p_1 \wedge p_2$
  - 例如 $W_{1,3} \wedge P_{3,1}$
- **析取 Disjunction**：$p_1 \vee p_2$
  - 例如 $W_{1,3} \vee P_{3,1}$
- **蕴含 Implication**：$p_1 \rightarrow p_2$
  - 例如 $W_{1,3} \wedge P_{3,1} \rightarrow \neg W_{2,2}$
- **相互蕴含 If and only if**：$p_1 \leftrightarrow p_2$
  - 例如 $W_{1,3} \leftrightarrow \neg W_{2,2}$

语义

- 语义定义了判断句子真实性的规则
- 语义可以由真值表指定
- 布尔值范围：T，F
- n - 元组 $(x_1, x_2, ..., x_n)$
- 对于 n - 元组的操作 $g(x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$
- 定义：真值表通过在 n -元组上定义一个操作符 g，为每个元组指定布尔值
- 真值表的行数：2^n

真值表

<table>
<tr><th>语法</th><th>真值表</th></tr>
<tr><td>否定 Negation</td><td>
<table>
<tr><th>p</th><th>$\neg p$</th></tr>
<tr><td>T</td><td>F</td></tr>
<tr><td>F</td><td>T</td></tr>
</table>
</td></tr>
<tr><td>合取 Conjunction</td><td>
<table>
<tr><th>p</th><th>q</th><th>$p \wedge q$</th></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
</table>
</td></tr>
<tr><td>析取 Disjunction</td><td>
<table>
<tr><th>p</th><th>q</th><th>$p \vee q$</th></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
</table>
</td></tr>
<tr><td>异或 Exclusive OR</td><td>
<table>
<tr><th>p</th><th>q</th><th>$p \oplus q$</th></tr>
<tr><td>T</td><td>T</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
</table>
</td></tr>
<tr><td>蕴含 Implication</td><td>
<table>
<tr><th>p</th><th>q</th><th>$p \rightarrow q$</th></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>F</td><td>T</td></tr>
</table>
</td></tr>
<tr><td>双向蕴含 Biconditional / If and only if IFF</td><td>
<table>
<tr><th>p</th><th>q</th><th>$p \leftrightarrow q$</th></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td></tr>
<tr><td>F</td><td>F</td><td>T</td></tr>
</table>
</td></tr>
</table>

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

运算符的优先级

1. 括号（从里到外）
2. 否定
3. AND
4. OR
5. 蕴含
6. 双向蕴含
7. 从左到右

逻辑等价 Logical Equivalence

两个表达式 p 和 q 是逻辑等价的，当且仅当它们的真值表的各列是一样的，我们把它写作

$$p \equiv q$$

p	q	$\neg p$	$\neg p \vee q$	$p \rightarrow q$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

例如这里，我们可以得到

$$(\neg p \vee q) \equiv (p \rightarrow q)$$

运算符的性质

性质	公式
交换律	$p \wedge q \equiv q \wedge p$ $p \vee q \equiv q \vee p$
结合律	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ $(p \vee q) \vee r \equiv q \vee (p \vee r)$
Identity 元素	$p \wedge True \equiv p$ $p \vee Ture \equiv Ture$ $\neg(\neg p) \equiv p$ $p \wedge p \equiv p$ $p \vee p \equiv p$ $p \wedge \neg p \equiv False$ $p \vee \neg p \equiv Ture$
分配律	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) \equiv (p \wedge q) \vee (p \wedge r)$
DE Morgan 定律	$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$ $\neg(p \vee q) = (\neg p) \wedge (\neg q)$

推理

演绎推理

如果 p 为真，且 $p \rightarrow q$，那么 q 为真，写作

$$\frac{p \quad p \rightarrow q}{q}$$

否定演绎推理

如果 p 为假，且 $p \rightarrow q$，那么 q 为假，写作

$$\frac{\neg p \quad p \rightarrow q}{\neg q}$$

霍恩子句 Horn Clauses

如果一个命题的表达为

$$p_1 \wedge ... \wedge p_n \rightarrow q$$

那么演绎推理法也能在霍恩子句上起作用

如果 $p1,...,p_n$ 均为真，且 $p_1\wedge...\wedge p_n \to q$ 为真，那么 q 为真，写作

$$\frac{p1,...,p_n \quad p_1\wedge...\wedge p_n \to q}{q}$$

常见的推理

- Addition $\frac{p}{p\vee q}$
    - 如果 p 为真，那么 $p \vee q$ 为真
- Simplification $\frac{p\wedge q}{q}$
    - 如果 p 和 q 为真，那么 q 为真
- Disjunctive-syllogism $\frac{p\vee q \quad \neg p}{q}$
    - 如果 $p \vee q$ 为真，且 p 为假，那么 q 为真
- Hypothetical-syllogism $\frac{p\to q \quad q\to r}{p\to r}$
    - 如果 p 蕴含 q，q 蕴含 r，那么 p 蕴含 r

蕴含和推理 Entailment and Inference

语义

通过 **Model Checking** 确定逻辑蕴含

枚举所有的模型，然后证明语句必须满足所有的模型

$$KB \vDash \alpha$$

语法

通过 **Theorem Proving** 验证所有的推导的逻辑蕴含，使用推断的规则来建立一个对于 α 的验证，不需要枚举和检查所有的模型

$$KB \vdash \alpha$$

可靠性 Soundness

推理的算法不可能推断出错误的公式，也就是，只得到逻辑蕴含的语句

$$\{\alpha | KB \vdash \alpha\} \subseteq \{KB \vDash \alpha\}$$

完整性 Completeness

能够推断出**所有的**逻辑蕴含的语句

$$\{\alpha \mid KB \vdash \alpha\} \supseteq \{KB \vDash \alpha\}$$

有效性 Valid

一个语句是有效的 valid（也叫 tautology）如果它在所有模型上都是 True 的

可满足性 Satisfiability

一个语句是**可满足的 satisfiable**，如果在一些情况下它在模型中是 True

- $p \vee p$

一个语句是**不可满足的 unsatisfiable**，如果它在模型中不可能是 True

- $p \wedge \neg p$

如何确定蕴含

给定一个知识库 KB（一系列逻辑蕴含语句），给定一个查询 α，输出 KB 是否能推导出逻辑蕴含 α，记作 $KB \vDash \alpha$

有两种方法

- Model Checking：枚举所有的模型（真值表）
- Theorem Proving：句法派生的规则，如 Modus Ponens （向后链接和向前链接）一个证明是一个推理规则应用的序列

使用命题逻辑描述 Wumpus World

- $P_{i,j}$：如果 $[i,j]$ 上有 PIT，那么 $P_{i,j}$ 为 True
- $B_{i,j}$：如果 $[i,j]$ 上有 Breeze，那么 $B_{i,j}$ 为 True

我们得到了几条语句

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1}$
- $R_3 : B_{2,1} \leftrightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$
- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$

Model Checking

模型：将 T 或 F 赋值给每一个命题逻辑符号

在 KB 为 True 的每个模型中，如果 α 都是 True，那么 α 确实为真

在 Wumpus World 中，我们有五条语句

$$KB = R_1 \land R_2 \land R_3 \land R_4 \land R_5$$

- 我们要始终保持 $KB = True$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$	$true$	$true$	$true$	$false$	$false$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$true$	$true$	$false$	$true$	$false$	$false$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$true$	$true$	$false$	$true$	$true$	$false$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$true$	$true$	$true$	$true$	$true$	$true$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$true$	$true$	$true$	$true$	$true$	$true$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$true$	$true$	$true$	$true$	$true$	$true$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$true$	$false$	$false$	$true$	$true$	$false$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$true$	$true$	$false$	$true$	$false$

- 枚举情况如下，在 $KB = True$ 的唯三条语句中，我们发现
- 这些是可以确定的
  - $B_{1,1} = False$
  - $B_{2,1} = True$
  - $P_{1,1} = False$
  - $P_{1,2} = False$
  - $P_{2,1} = False$
- 还有不能确定的 $P_{2,2}, P_{3,1}$

Theorem Proving

推断也可以被表示成一个搜索问题

- **Initial state:** The initial KB
- **Actions:** all inference rules applied to all sentences that match the top of the inference rule
- **Results:** add the sentence in the bottom half of the inference rule
- **Goal:** a state containing the sentence we are trying to prove.

- 初始状态：初始的 KB
- 行动：应用推理规则到基于顶部规则的所有语句
- 结果：在推理规则的下半部分添加句子
- 目标：一个包含了我们想证明的所有语句的状态

Theorem Proves 比起 Model Checking 来说，可以以一种更有效的方式来搜索证明

- 真值表的数量是指数级的
- 推断的核心是重复的应用推理规则到 KB 上
- 只要在知识库中找到合适的前提，就可以应用推理

推理肯定是可靠的 Sound，但是如何保证完整性 Completeness，有两种手段

- **通过归结 Proof by Resolution**（一种强大的归结规则）
- **前向 / 反向连接 Forward or Backward chaining**：在限定的命题形式上使用推理词

Theorem Proving - 归结 Proof by Resolution

合取范式 Conjunctive Normal Form

合取范式是一种表述关系，它的表述大致如下

$$(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D) \wedge F$$

- 括号内的是原子命题的吸取
- 括号与括号之间是合取

所有的命题逻辑都可以被转换成合取范式，并且对于 CNF 的推理规则是可靠 sound 且完整 complete 的

- KB = CNF 范式

转换成合取范式

给定命题 $B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1}$

1. 简化，将 $\alpha \leftrightarrow \beta$ 转换成 $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
2. 将 $\alpha \rightarrow \beta$ 替换成 $\neg\alpha \vee \beta$
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
3. 使用 DE Morgan 定律将 $\neg$ 放在括号内
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
4. 使用分配律
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

这里**每个括号内的成分**被称为一个**子句 Clause**

单元归结 unit resolution

$$\frac{\ell_1 \vee \cdots \vee \ell_k \quad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

- 如果 ℓ_1 和 m 是互补的文字，那么可以进行上述归结

例如

$$\frac{P_{1,3} \vee P_{2,2} \quad \neg P_{2,2}}{P_{1,3}}$$

单元归结 = 子句 + 文字 -> 新的子句

CNF 归结

$$\frac{\ell_1 \vee \cdots \vee \ell_k \quad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

- 对于两个子句 $\ell_1 \vee \cdots \vee \ell_k$ 和 $m_1 \vee \cdots \vee m_n$ 来说
- 如果 ℓ_i 和 m_j 是互补的元素
- 那么可以归结成 $\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$

归结算法 Resolution Algorithm

为了证明 $KB \vDash \alpha = True$，只需要证明 $KB \wedge \neg\alpha = False$ 即可

```
function PL-RESOLUTION(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

- 这里的 $PL-RESOLVE$ 依次对两两子句进行单元归结或者 CNF 归结

示例

我们以 Wumpus World 继续作为示例

假设现在知识库为

$$KB = R_4 \wedge R_2 = (B_{1,1} \leftrightarrow (P_{1,2} \wedge P_{2,1})) \wedge \neg B_{1,1}$$

需要推理的 $\alpha = \neg P_{1,2}$

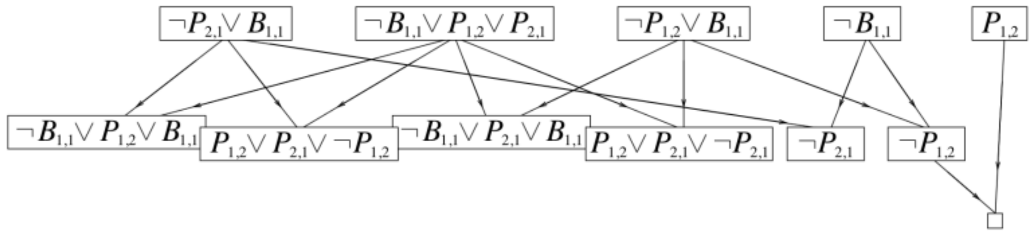
上面已经求了 $B_{1,1} \leftrightarrow (P_{1,2} \wedge P_{2,1}) = (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

所以，所有的子句有

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}), (\neg P_{1,2} \vee B_{1,1}), (\neg P_{2,1} \vee B_{1,1}), (\neg B_{1,1}), (P_{1,2})$

- $P_{1,2}$ 就是 $\neg\alpha$

检测的目标是这些子句最终都要满足为真，两两子句进行归结



最终还剩下 $\neg P_{2,1}, \neg P_{1,2}, P_{1,2}$，要保证这三个都为 True，是不可能的，所以返回 False

Theorem Proving - 前向/反向连接

KB 可以表示为一系列霍恩子句的合取

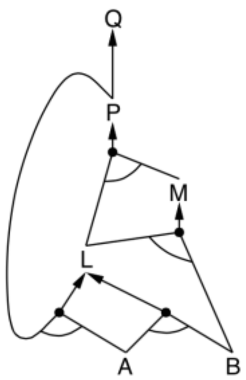
- 霍恩子句是命题逻辑的表达范式 $p_1 \wedge ... \wedge p_n \rightarrow q$

可以被前向 / 反向连接所使用

例如，给定 KB 如下

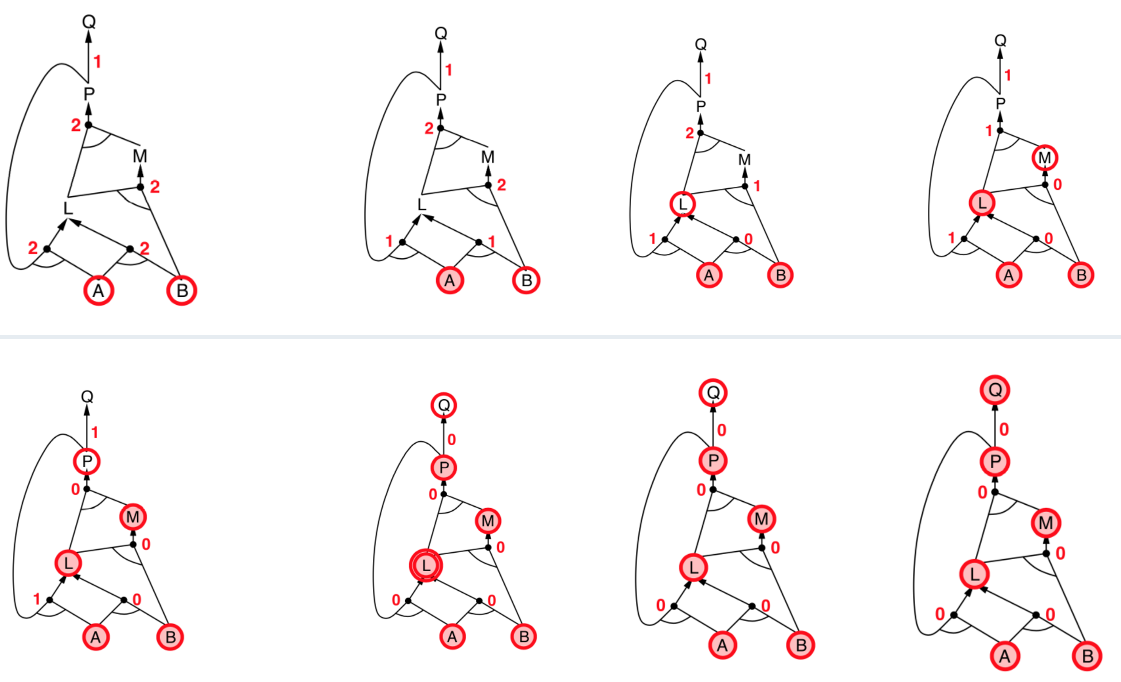
$$
\begin{aligned}
&P \Rightarrow Q \\
&L \wedge M \Rightarrow P \\
&B \wedge L \Rightarrow M \\
&A \wedge P \Rightarrow L \\
&A \wedge B \Rightarrow L \\
&A \\
&B
\end{aligned}
$$

建立传播图

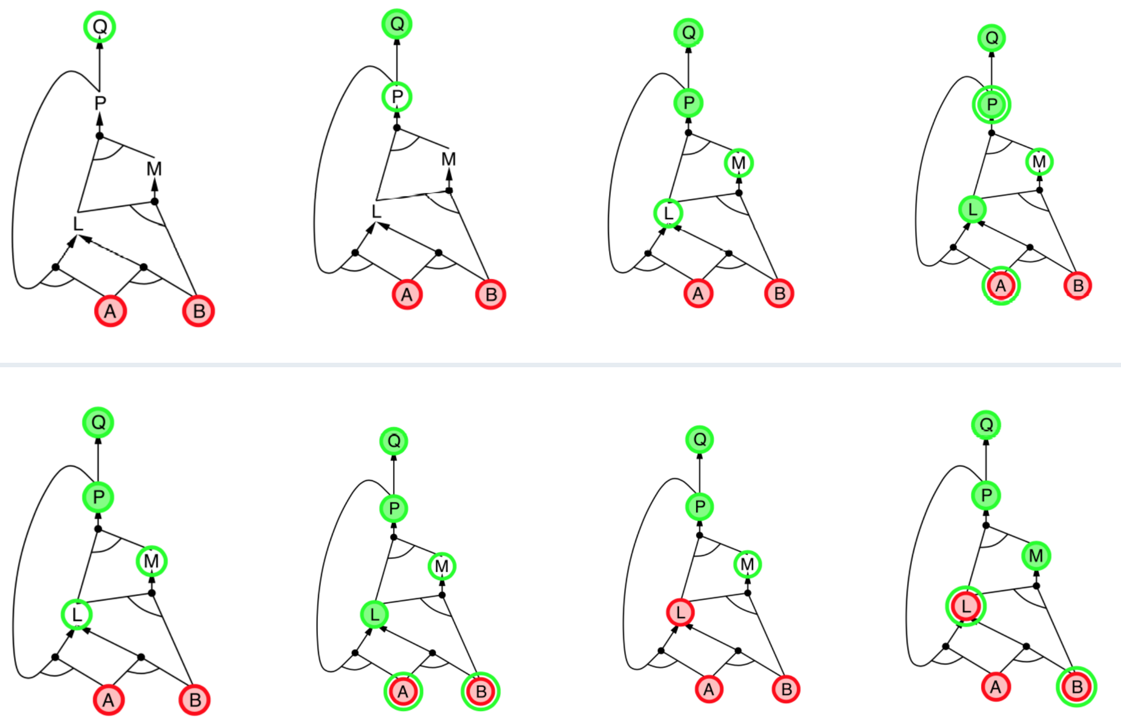


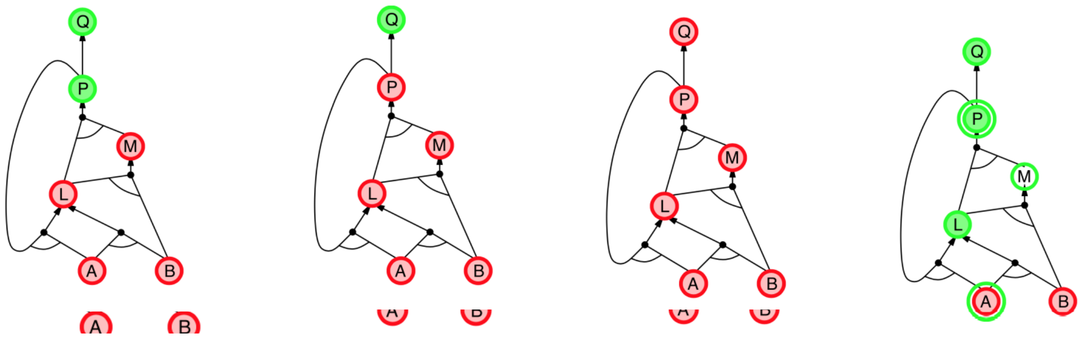
前向连接 Forward Chaining

从 A、B 开始推到 Q



反向连接 Backward Chaining





前向 vs 反向

- 前向连接：从事实出发，无目的的推导
  - 数据驱动的，自动的，无意识的传播
  - 可能会做很多与目标无关的工作
- 反向连接：从目的出发，找到符合目的的要求，更适合求解问题
  - 目标驱动，对于解决问题很合适
  - 反向连接的复杂度可能在 KB 级别的

命题逻辑的局限

- PL 的**表现力不足**以描述我们周围的世界，它不能表达不同对象的信息以及对象之间的关系
- PL 是**不紧凑**的，它不能在不枚举所有的对象的情况下表示一组对象的一个事实，这有时是不可能的

例如，如果我们有一个真空吸尘器来清洁一个 10x10 格子组成的教师，如何用命题逻辑表明格子

- 命题逻辑 $s_1\_is\_clean$ 来表述第一个格子是干净的
- 如何表示所有的格子都是干净的？
  - $s_1\_is\_clean \land s_2\_is\_clean \land \ldots \land s_{100}\_is\_clean$
- 如何表示有些格子是干净的？
  - $s_1\_is\_clean \lor s_2\_is\_clean \lor \ldots \lor s_{100}\_is\_clean$

太复杂了！

4. 一阶逻辑 First Order Logic

语法

- 项 Term
  - 常量符号（$A \quad 10 \quad ShenZhen$）
  - 变量（$x \quad y$）
  - 函数项（$sqrt(x) \quad sum(1,2)$）
- 原子公式 Atomic Formula
  - 运用于项的谓词
    - $borther(x,y)$：x 是 y 的兄弟
    - $above(A,B)$：A 在 B 的前面
- 连结词：Connectives
  - $\land, \lor, \to, \leftrightarrow, \neg$
- 等价 Equality

    - $\equiv$
- 量词 Quantifier
    - $\forall, \exists$

量词、等价、连接词可以运用在原子公式上来创造一阶逻辑的语句

示例

所有的方块都是干净的

$$\forall x\ Square(x) \rightarrow Clean(x)$$

存在某些不干净的格子

$$\exists x\ Square(x) \land \neg Clean(x)$$

所有的鸟都会飞

$$\forall x\ bird(s) \rightarrow fly(x)$$

除了企鹅外，所有的鸟都会飞

$$\forall x\ bird(x) \land \neg penguin(x) \rightarrow fly(x)$$

所有的孩子喜欢糖

$$\forall x\ kid(x) \rightarrow likes(x, candy)$$

兄弟是同胞

$$\forall x, y\ brothers(x, y) \rightarrow sibling(x, y)$$

一个人的母亲是一个人的女性的家长

$$\forall x, y\ mother(x, y) \leftrightarrow female(x) \land parent(x, y)$$

推理

- 虽然比 PL 稍微复杂一些，但是有一些程序可以使用基于 FOL 公式的知识库来进行推论
- FOL 的可表达性表明，自动实现自然语言和逻辑表达式之间的转换是可能的，这对于不同的应用程序是非常有价值的，比如个人助理(Siri)，问题/回答系统，以及一般与计算机通信

5. 总结

- 逻辑智能体根据知识库进行推断，来得到新的信息并且做决策
- 逻辑的基本概念
    - **语法 Syntax**：语句的结构
    - **语义 Semantics**：语句的真实性
    - **逻辑蕴含 Entailment**：一个句子给出另一个句子的必然真理
    - **推断 Inference**：从别的语句中推断出的语句
    - **可靠性 Soundness**：推断出的语句只得到逻辑蕴含的语句
    - **完整性 Completeness**：推断出的语句可以得到所有的逻辑蕴含语句
- 对于霍恩子句来说，前向 / 后向连接在线性时间可以完成，对于命题逻辑来说，归结算法是完整的

优缺点

优点

- 可解释性：模型可以被显示表述

缺点

- 不能处理不确定性
- 基于规则的，但不实用数据
- 很难去建模世界的每一个部分