产生式

也称产生式规则，简称规则。是表示知识的一种方法，适合表示知识单元中的因果关系。如 P → Q 或 if P then Q.

P被称为前件或执行条件，Q被称为后件或规则体

BNF范式

<产生式> ::= <前提> ⟶ <结论>

<前提> ::= <简单条件> | <复合条件>

<结论> ::= <事实> | <操作>

<操作> ::= <操作名>[(<变元>, …)]

<复合条件> ::= <简单条件> AND <简单条件>[(AND<简单条件>) …] | <简单条件> OR …

::= 定义为　| 或　[] 可选项　<> 语法单元

与蕴含式的区别：产生式可以表达不精确知识（按算法匹配已知事实与前提条件，匹配可以精确，也可不精确）

产生式系统

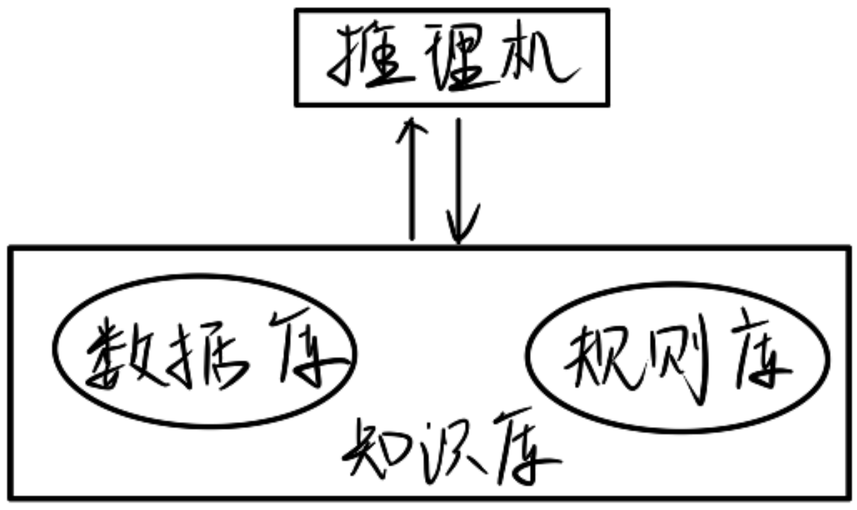
将一组产生式放在一起，一个产生式的结论可以作为另一个产生式的前提，并由此求解问题。由于其知识库主要用于存储规则所以又称为基于规则的系统

组成三要素

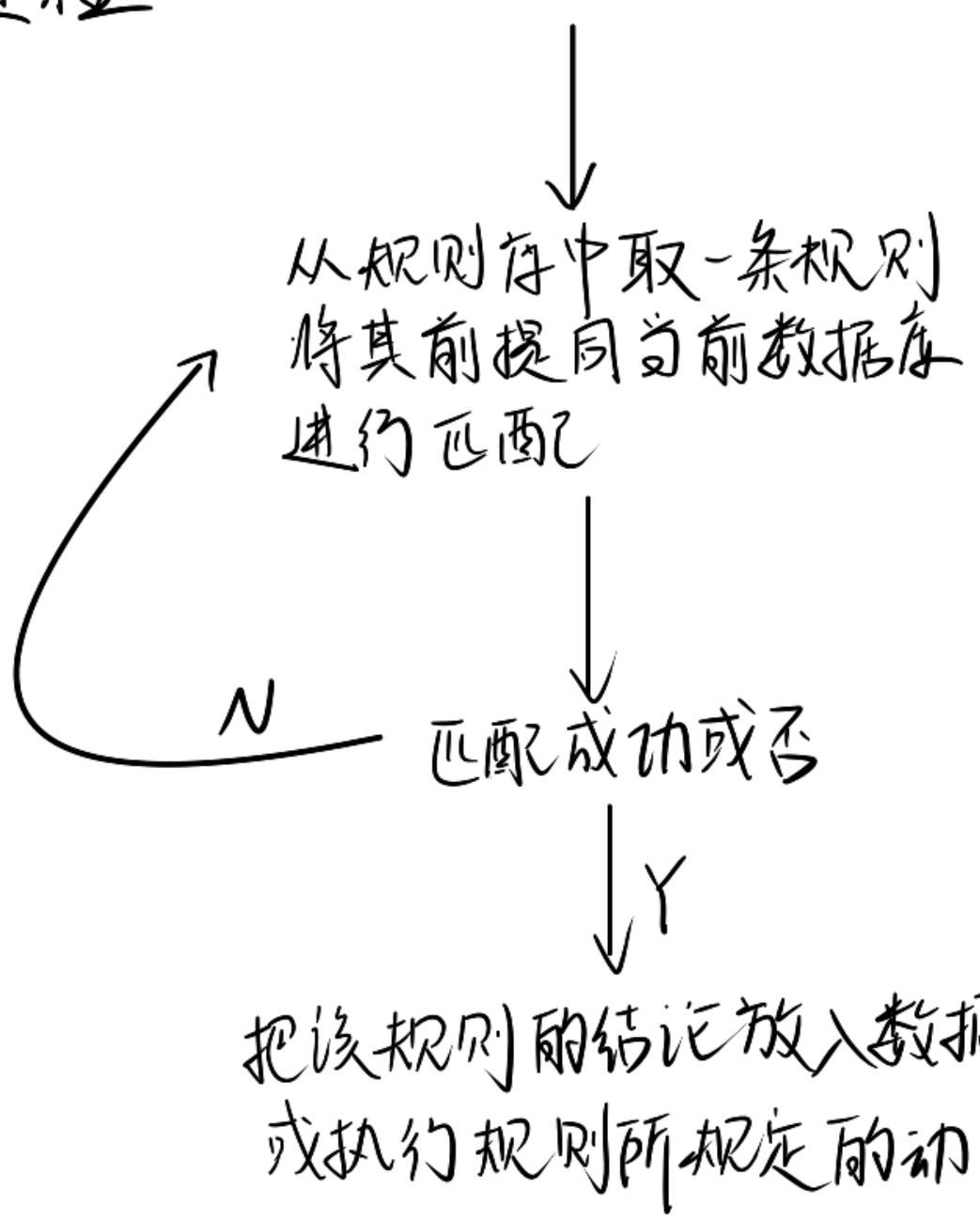
① 综合数据库：存放问题求解过程中各种当前信息

② 产生式规则集

③ 控制策略（推理机）：规则的解释或执行程序



运行过程



1、匹配：按一定策略从规则集选取规则并与全局数据基中的已知事实进行匹配（将前提与已知事实进行比较，若相似度大于一定阈值则匹配成功）

(1) 索引匹配：为状态建立可用产生式索引表，减少搜索范围

(2) 分层匹配：将产生式分成若干层或组，按一定特征进行搜索

(3) 过滤匹配：边匹配边按某些特征（或参数）对可用产生式进行精选

2、冲突：匹配成功的规则可能不止一条，这就叫产生了冲突
推理机必须用相应策略进行消解.

(1) 按规则编排的顺序：只选择最早匹配成功的一条
(2) 规模排序：只选前提条件或情况元素最多的一条
(3) 就近排序：最近用过的产生式优先(或反之)
(4) 数据排序：给情况以不同的优先权，与优先权
较高的情况相匹配的产生式优先使
用
(5) 上下文排序：利用上下文限制
(6) 专一性排序：按针对性.

3、操作：执行规则的右部。对于不确定性知识，在执行每一条
规则时还要按一定算法计算结论的不确定性

4、在适当时刻结束系统运行.

求解问题一般步骤：

1，初始化全局数据基，把问题的初始已知事实送入数据基中
2，若规则库中存在尚未使用过的规则，而且它的前提可与数据基中的已知事实匹配，则转3步，若不存在这样的事实，则转5步。
3，执行当前选中的规则，并对该规则做上标记，把该规则执行后的结论送入数据基中。如果该规则的结论是某些操作，则执行之。
4，检查数据基中是否已包含了问题的解，若已包含，则终止问题的求解过程，否则转2步；
5，要求用户提供进一步的关于问题的已知事实，若能提供，则转2步，否则终止问题的求解过程。
6，若规则库中不再有未使用过的规则，则终止求解过程。

表示：1、DATA ← 初始数据库
2、until DATA满足终止条件 do
3、begin
4、在规则集中选择能作用到DATA上的规则R.
5、DATA ← R作用到DATA上的结果
6、end.

规则的模式匹配.
假设系统有N条规则，每条规则的条件部分(LHS)有P个模.

式，在某个时点有M个事实要处理，则：

1、从N条规则中取出一条r

2、从M个事实中取P个事实的一个组合C

3、用C测试LHS(r)，若为True，则将事实部分RHS(r(c))加入队列中

4、如果M个事实还存在其它组合C'，则go to 3

5、取出下一条规则，go to 2.

模式匹配算法

由于以上处理涉及组合，过程比较复杂，有必要设计算法来优化匹配效率

- Markov算法

  按优先级对规则集合排序，每次选择优先级最高的规则应用

- Rete算法

  这是目前使用最广泛的模式匹配算法，基于以下两个假设

  - 时间冗余性：事实在推理过程中变化是缓慢的，每个执行周期中，只有少数事实在变化，因此影响的规则也只占很小的比例，所以可以只考虑每个周期中已匹配的事实

  - 结构相似性：许多规则常常包含类似的模式和模式组

基本思想：对每个模式创建一个匹配元素表记录该模式可以匹配当前WorkingMemory(当前事实)中的哪个事实。当一个新事实加入WM时，找出其能匹配的模式并加入对应匹配元

表表中；当一个事实从WM中删除时，同样找出其能匹配的模式并从对应匹配元素表中删除

步骤：1. 将初始数据(fact)输入WM

2. 使用Pattern Matcher比较rule和fact

3、若执行规则存在冲突，则将冲突的规则放入冲突集合.

4、解决冲突，按激活顺序将规则放入Agenda

5、使用规则引擎执行Agenda中的规则，重复2~5直到Agenda中无规则.

产生式系统特点

优：自然 直观

数据驱动 知识无序性

控制系统与问题无关

数据、知识和控制相对独立(模块化)

缺：效率低、不能表示具有结构性的知识(相对于if...then这种过程性知识，结构性指事物的层次、结构、类属关联等。如学生属于人，汽车由发动机构成)

例、3个传教士和3个野人渡河，一条可坐2人的船，要求在河的两岸传教士人数不能少于野人人数.(M-C问题)

设三元组(m, c, b) $0 \le m \le 3$ $0 \le c \le 3$ $b \in \{0, 1\}$.

m：左岸传教士数量

c：左岸野人数量.

b：船在左/右岸(1为左岸)

初始：(3, 3, 1) 目标：(0, 0, 0)

规则集：

IF (m, c, 1) AND m≥1 THEN (m-1, c, 0)　　// 1传教士过河
IF (m, c, 1) AND c≥1 THEN (m, c-1, 0)　　// 1野人过河
IF (m, c, 1) AND m≥1 AND c≥1 THEN (m-1, c-1, 0)　// 1传1野过河
IF (m, c, 1) AND m≥2 THEN (m-2, c, 0)　　// 2传教士过河
IF (m, c, 1) AND c≥2 THEN (m, c-2, 0)　　// 2野人过河

IF (m, c, 0) THEN (m+1, c, 1)　　// 1传教士划船回来
IF (m, c, 0) THEN (m, c+1, 1)　　// 1野人划船回来
IF (m, c, 0) THEN (m+1, c+1, 1)　// 1传1野划船回来
IF (m, c, 0) THEN (m+2, c, 1)　　// 2传教士划船回来
IF (m, c, 0) THEN (m, c+2, 1)　　// 2野人划船回来

另一种更精简的表示

IF (m, c, 1) AND 1≤i+j≤2 THEN (m-i, c-j, 0)
IF (m, c, 0) AND 1≤i+j≤2 THEN (m+i, c+j, 1)

约束

左岸 $m=0$ 或 $m\geq c$

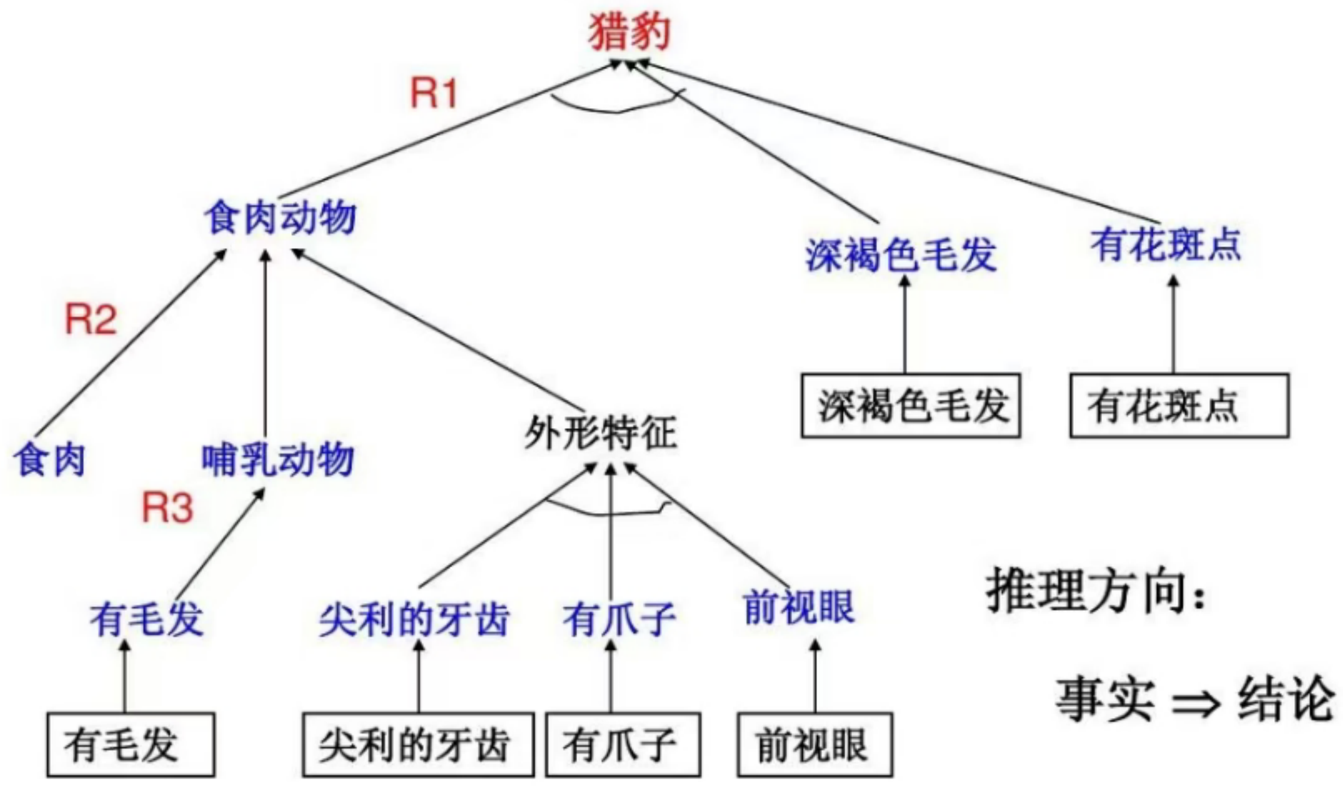
右岸 $3-m=0$ 或 $(3-m)\geq(3-c)$

$0\leq m\leq 3$　　$0\leq c\leq 3$

产生式系统推理

正向推理：从初始事实/数据出发，正向使用规则推理，朝目标方向前进

1、将初始事实置入动态数据库
2、用动态数据库中的事实匹配目标条件，若目标条件满足则成功，结束
3、用规则库中各规则的前提匹配动态数据库中的事实，将匹配成功的规则组成待用规则集
4、若待用规则集为空则失败，退出
5、用冲突消解策略将待用规则集中各规则结论加入动态数据库，或执行其动作，转步2



反向推理：从目标出发，反向使用规则进行推理，朝初始事实方向前进

1、初始化动态数据库，将初始事实置入动态数据库
将目标条件置入目标链

2、若目标链为空，则推理成功，结束

3、取出目标链中的第一个目标，用动态数据库中的事实同其匹配，若匹配成功，转步2.

4、用规则库中各规则的结论同该目标匹配，若成功，则将第一个匹配成功且未用过的规则前提作为新目标，并取代原来的父目标加入目标链，转步3

5、若该目标是初始目标，则推理失败，退出.

6、将该目标的父目标物回目标链，取代该目标及其兄弟目标，转步3

特点	正向推理	逆向推理
推理驱动方式	数据驱动	目标驱动
优点	算法简单，易于实现	搜索目的性强，推理效率高
缺点	1、搜索目的性弱，可能求解出多个无关的结论； 2、 匹配时，要遍历整个规则库，推理效率低。	1、确定目标的时候，具有盲目性，可能产生假目标 2、当规则的后件是操作而非断言时，即反应型系统，不宜使用此法
用途	主要用于已知初始数据，不知目标的推理；或是**解空间**大的一类推理。	主要用于结论单一或已知目标求证的一类推理。
应用	监控、预测、规划、设计等	选择、分类、故障诊断等

产生式系统分类

按搜索策略分

1、不可撤回的产生式系统：规则使用后，不允许回过头来重新选择其它规则

2、试探性的产生式系统：规则使用后允许返回原出发点，重新选择其它规则

(1) 回溯式：规则使用后记住原来的结点，若搜索困难时可返回，再选其他规则，如有界DFS

(2) 图搜索式：同时掌握若干规则序列的效果，从中寻找问题答案，通常采用树搜索，如BFS

按搜索方向分

正向、反向、双向

其它

可交换的产生式系统：各规则选用次序不重要

可分解的产生式系统：初始数据库可分解为若干部分，分别使用规则直到目标状态为止。终止条件也相应分解

回溯式产生式系统

适用于搜索量小的问题。完备、有效、易于实现、占用存储少

函数/谓词	功能说明
`Term(DATA)`	判断当前数据库 `DATA` 是否满足**终止条件**
`DEADEND(DATA)`	判断当前数据库 `DATA` 是否满足**失败条件**
`APPRULES(DATA)`	选取可作用于 `DATA` 的所有操作，并对规则排序
`DATA(A)`	计算操作作用在当前数据上，生成**新数据库**
`NULL(RULES)`	判断可用规则表是否为空
`FIRST(RULES)`	取出可用规则表的**第一个规则**
`TAIL(RULES)`	去掉可用规则表的第一个元素，更新规则表
`CONS(R, PATH)`	将规则R添加到路径表 `PATH` 的头部，记录搜索路径

1. 步骤1：若满足终止条件 TERM(DATA)，返回 NIL，搜索成功。
2. 步骤2：若满足失败条件 DEADEND(DATA)，返回 FAIL，搜索失败。
3. 步骤3：调用 APPRULES(DATA)，获取当前可用规则表 RULES。
4. 步骤4：循环判断：若规则表为空，返回 FAIL。
5. 步骤5：取出规则表第一个规则 R。
6. 步骤6：更新规则表，剔除已用规则。
7. 步骤7：用规则 R 作用于当前数据，生成新数据库 RDATA。
8. 步骤8：递归调用 BACKTRACK(RDATA)，深度搜索新状态。
9. 步骤9：若递归返回失败，回到循环，尝试下一条规则。
10. 步骤10：递归成功，将当前规则加入路径，返回完整搜索路径。

专家系统