第四章 产生式系统

戴洪良
计算机科学与技术学院/人工智能学院
hongldai@nuaa.edu.cn

产生式（Production）的由来

1

1943年，美国数学家 E.L.Post

提出Post canonical system，也叫Post production system，使用了字符串更改规则，在迭代过程中会匹配替代，产生新字符串，模型每条规则称为产生式

2

1972年，西蒙(Simon)与纽厄尔(Newell)

把产生式理论用于解释过程性知识的获得机制，认为人经过学习，头脑中存储了一系列“如果…那么…”形式的知识

产生式系统（Production System）

- 产生式系统是一种采用特定形式*规则*的推理系统，它使用的规则称作*产生式规则（Production Rules）*，或简称*产生式（Productions）*

产生式规则（Production Rule）

前提，或称前件，形式为 $p1, p2, p3, \ldots$

结论或操作，或称后件，形式为 $a1, a2, a3, \ldots$

IF conditions THEN actions

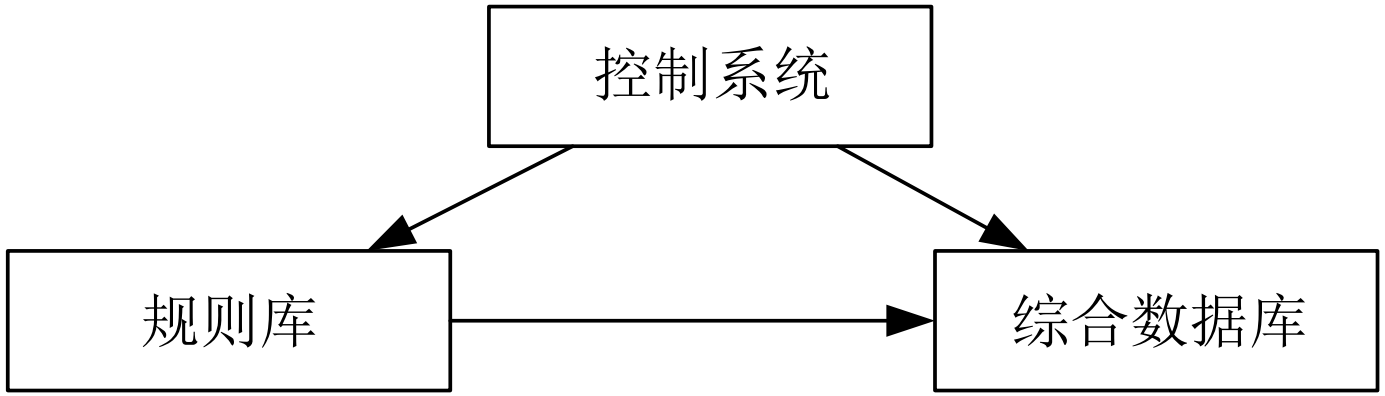
- 前提（conditions）满足后，可执行操作（actions）
- 产生式规则可以表述多种逻辑语境，比如“原因-结果”，“条件-结论”，“前提-操作”，“事实-进展”，“情况-行为”等等
- 是AI中知识表示的一种重要方式

例如：

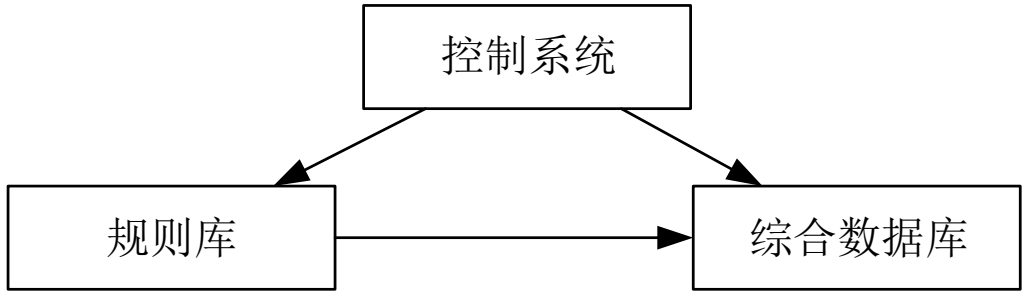
1.风乍起，吹皱一池春水（原因-结果）；
2.如果平面图形由三条边首尾相连而成，且三边相等，则其三角相等（条件-结论）；
3.如果今天能够完成这项任务，并且后续工作可以放一放的话，那么明天我们就可以放假（前提-操作）；
4.随风潜入夜，润物细无声（事实-进展）；
5.如果温度低于0℃，水就会结冰，如果温度高于100℃，水就会沸腾（情况-行为）。

产生式系统（Production System）

- 产生式系统通常由三部分组成



产生式系统结构



- 综合数据库——人工智能系统的数据结构中心。是一个动态数据结构，用来存放初始事实数据、中间结构和最后结果。
- 产生式规则库——作用在综合数据库上的一些规则的集合。每条规则都有一定的条件，若数据库中内容满足这些条件可调用这条规则。
- 控制系统——负责产生式规则的前提条件测试或匹配，规则的调度和选取，规则体的解释和执行。

综合数据库

- 综合数据库不是常规意义的数据库
  - 英文叫Global Database或Working Memory，里面存放特定结构的数据
- 存放的数据：初始事实数据、中间结果和最后结果
- 在系统执行过程中动态变化
- 不同系统可用不同的方式表示数据

例: 数据可为命题、谓词，如$Red(car2)$

也可用如下形式表示每条数据

$$(type\ \ attribute_1: value_1\ \ \ldots attribute_n: value_n)$$

- (person age: 27 home: toronto)
- (goal task: putDown importance: 5 urgency: 1)
- (student name: john department: computerScience)

规则

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

- 其前提和操作也可为不同形式

前提例:

- 合取式：$A(x) \wedge B(x) \wedge C(y)$
- (type attr1 spec1 attr2 spec2 ...)
  - (person age {<23 ∧ >6})

规则

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

- 其前提和操作也可为不同形式
- 操作可以为
  - 增加（ADD）： 向数据库增加数据
  - 删除（REMOVE）：从数据库删除一条数据
  - 修改（MODIFY）：更改数据库中的数据
  - 等

例:

$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$$

$$(\text{Student name } x) \Rightarrow \text{ADD (Person name } x)$$

规则

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

- 其前提和操作也可为不同形式
- 操作可以为
  - 增加（ADD）： 向数据库增加数据
  - 删除（REMOVE）：从数据库删除一条数据
  - 修改（MODIFY）：更改数据库中的数据
  - 等

例:

IF the car is dead
THEN action:check_the_fuel; ADD “step 1 is complete”

IF “step 1 is complete”
AND the fuel task is full
THEN action:check_the_battery; ADD “step 2 is complete”

控制系统

- 控制整个产生式系统的执行

- **识别（recognize）**：找出哪些规则是可用的。也即，那些前提已在当前数据库中满足了的规则。
- **冲突化解（resolve conflict）**：在可用的规则中，选择应该执行其中的哪一条（which rule to "fire"）
- **执行（act）**：通过执行被选择的那些规则，改变数据库

还需检验状态是否满足终止条件。

产生式系统的适用范围

1）知识杂乱、事实众多、无统一理论的领域

2）该领域的知识能够抽象出来 且 可分解为一组独立的动作，以便用规则加以表示

产生式系统 – 例

- 任务：把三块砖头从大到小分别放到位置1, 2, 3上
- 数据库初始数据：

  (counter value: 1)

  (brick name: A size: 10 position: heap)

  (brick name: B size: 30 position: heap)

  (brick name: C size: 20 position: heap)

- 规则：

1. IF (brick position: heap name: n size: s)
   −(brick position: heap size: $\{> s\}$)
   −(brick position: hand)
   THEN MODIFY 1 (position hand)

2. IF (brick position: hand)
   (counter value: i)
   THEN MODIFY 1 (position i)
   MODIFY 2 (value $[i+1]$)

产生式系统 – 例

- 任务：把三块砖头从大到小分别放到位置1, 2, 3上
- 数据库初始数据：

(counter value: 1)

(brick name: A size: 10 position: heap)

(brick name: B size: 30 position: heap)

(brick name: C size: 20 position: heap)

执行规则1后，数据库变为：

(counter value: 1)

(brick name: A size: 10 position: heap)

(brick name: B size: 30 position: hand)

(brick name: C size: 20 position: heap)

- 规则：

1. IF (brick position: heap name: n size: s)
   –(brick position: heap size: $\{> s\}$)
   –(brick position: hand)
   THEN MODIFY 1 (position hand)

2. IF (brick position: hand)
   (counter value: i)
   THEN MODIFY 1 (position i)
   MODIFY 2 (value $[i+1]$)

产生式系统 – 例

- 任务：把三块砖头从大到小分别放到位置1, 2, 3上
- 数据库初始数据：

(counter value: 1)

(brick name: A size: 10 position: heap)

(brick name: B size: 30 position: heap)

(brick name: C size: 20 position: heap)

执行规则1和2后，数据库变为：

(counter value: 2)

(brick name: A size: 10 position: heap)

(brick name: B size: 30 position: 1)

(brick name: C size: 20 position: heap)

- 规则：

1. IF (brick position: heap name: n size: s)
   –(brick position: heap size: $\{> s\}$)
   –(brick position: hand)
   THEN MODIFY 1 (position hand)

2. IF (brick position: hand)
   (counter value: i)
   THEN MODIFY 1 (position i)
   MODIFY 2 (value $[i+1]$)

产生式系统运行过程

- **识别（recognize）**：找出哪些规则是可用的。也即，那些前提已在当前数据库中满足了的规则。
- **冲突化解（resolve conflict）**：在可用的规则中，选择应该执行其中的哪一条（which rule to "fire"）
- **执行（act）**：通过执行被选择的那些规则，改变数据库

  - 产生式系统的运行过程就是推理机不断的运用规则库中的规则，作用于动态数据库，不断进行推理并检测目标条件是否被满足的过程。

  - 产生式系统运行过程是从初始事实出发，寻求到达目标条件的通路的过程。所以也是一个搜索的过程。

识别（recognize）规则

- 产生式系统中可能有大量的规则
- 每次都单独地查看每条规则是否满足可能导致运行过慢

- 但是：
  - 每回合对动态数据库的改变很小
    - 上一回合前提满足的规则在新的回合很可能依然满足
  - 很多规则的一些前提条件是相同的
    - 单独检查每条规则的每个前提条件会导致重复计算

Rete算法

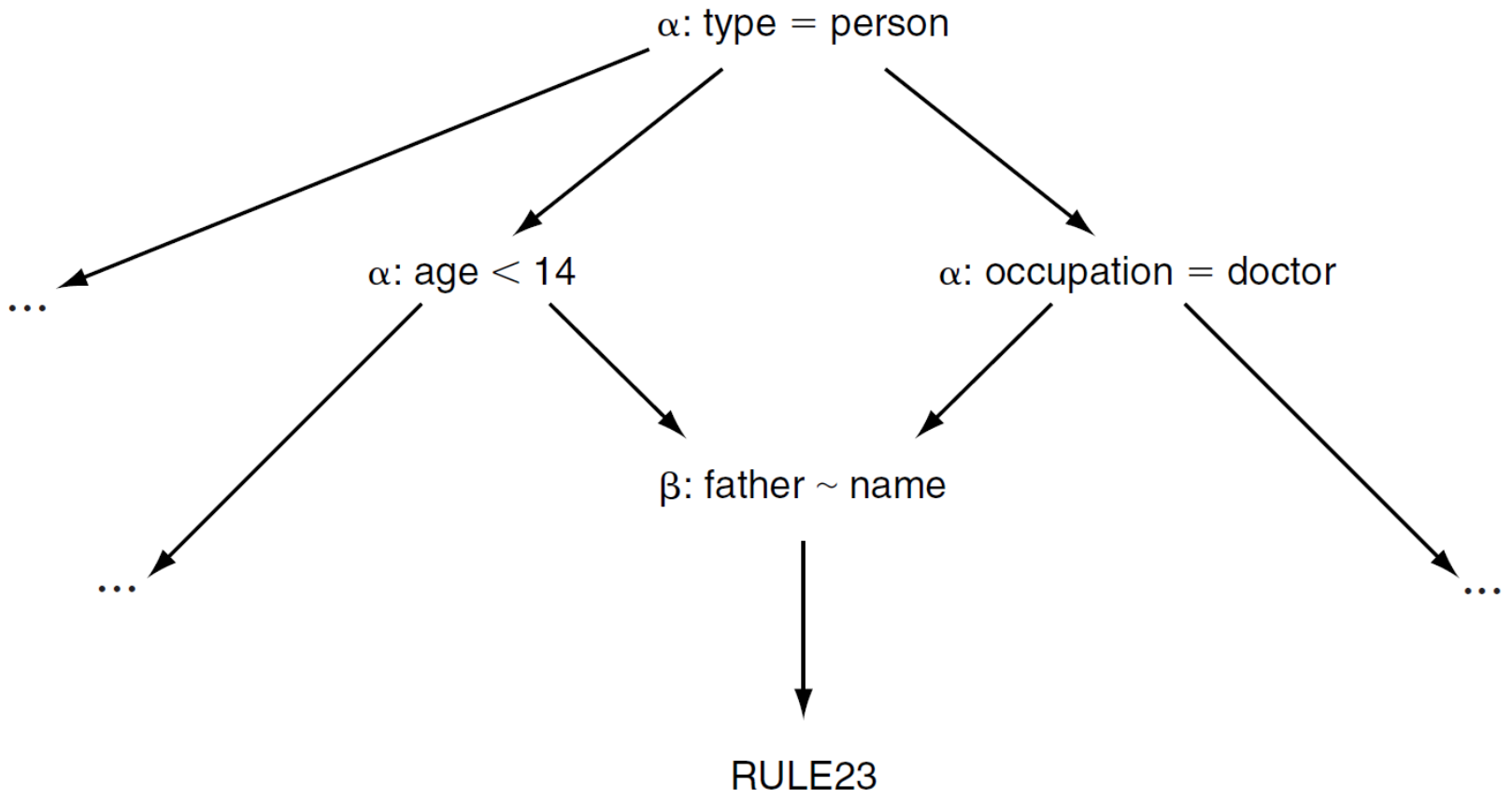
- 将所有规则的前提构建成一个网络
  - 网络中每个条件节点包含一个或多个条件
    - alpha节点包含对单条数据的条件
    - beta节点处理涉及多条数据的条件
  - 终止节点对应一条规则
  - 不同规则如果有相同的前提条件，可以共享利用同样的节点

- 对数据库中新产生的一条数据，用一个token代表它，进入网络
  - 如果token满足节点对应的条件，则其副本被传给该节点的子节点
  - token进入终止节点意味着对应规则的前提被满足
  - Beta节点包含一个左存储区和一个右存储区，分别存放来自“左边”和”右边”的token，对它们综合进行判断来确定节点条件是否被满足。如满足，用一个新的token代表该数据对。
  - 对涉及超过两条数据的条件，可用串联多个beta节点处理

Rete算法 – 例

RULE23:

```
IF (person name:x age:{< 14} father:y)
   (person name:y occupation:doctor)
THEN...
```



冲突化解（Conflict Resolution）

- 每回合中，所有可应用的规则称为冲突集（conflict set）
- 同时应用冲突集中所有规则可能导致出现不一致的知识和结果
- 因此通过冲突化解选择其中一条执行

冲突化解（Conflict Resolution）

- Order：直接选择第一条
  - 最简单且很常用
- Refractoriness:
  - 不应重复对同样的数据使用同样的规则
- Specificity：基于规则的具体性
  - 如果满足一组条件的数据是满足另一组条件的数据的子集，则这组条件比另一组更具体。一般可选择最具体的。

冲突化解（Conflict Resolution）

- Recency: 基于规则或数据是否近期被用过
  - 选择匹配最近被创建或更改的数据的规则：让系统关注刚刚在做的；
  - 选择离上次使用最久（least recently used）的规则：给不同的规则平等的机会。
- Saliency
  - 设定不同规则的优先程度。每次执行优先程度最高的规则。

控制策略

- 按照一定策略控制整个基于规则执行的搜索过程

两类：

- 不可撤回的控制策略
  - Hill climbing
- 试探性控制策略
  - 回溯控制策略

不可撤回的控制策略

- 选用规则执行后不撤回，直到达到目标或无法继续

- Hill climbing (爬山法)
  - 每次选使局部最优的规则
  - 需要有函数对每个状态打分
  - 每次选能获得最大进展的规则

- 例：8数码问题

初始状态：

2	8	3
1	6	4
7		5

目标状态：

1	2	3
8		4
7	6	5

产生式规则：空格左、上、右、下移；前提：不移出边界

不可撤回的控制策略

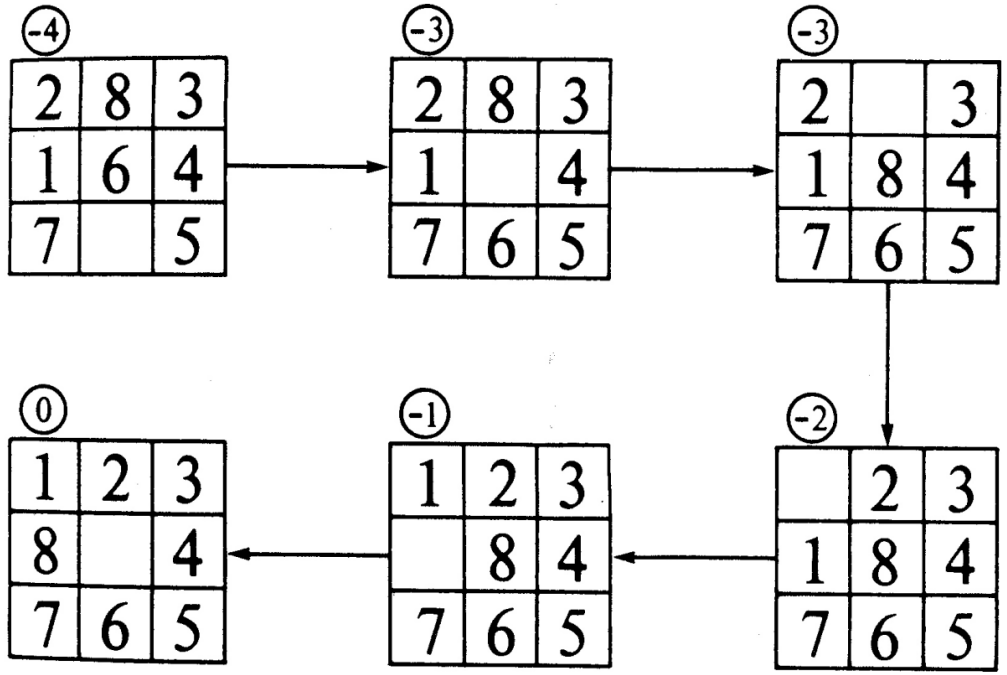
- Hill climbing (爬山法)
  - 每次选使局部最优的规则
    - 执行后的状态与目标状态相比，位置不相同的数字数最小
    - F(s)=-与目标状态位置不相同的数字数

初始状态：

2	8	3
1	6	4
7		5

目标状态：

1	2	3
8		4
7	6	5



不可撤回的控制策略

- 执行情况取决于初始状态
- 多数情况下找不到解

- 可能适用的场景
  - 执行一条“不好”规则对今后执行“好的”规则不产生影响

试探性控制策略

- 产生冲突时可尝试不同规则

- 回溯控制策略
  - 控制系统先选用一条规则，如果发现这条规则的选用不能导致产生解，则系统“忘掉”选用规则所涉及的步骤和产生的状态，然后选用另外一条规则，重新进行试探。
  - 类似深度优先搜索

回溯控制策略

- 只存储初始节点到当前节点的路径，占用空间较小。
- 每次可评估规则的“好坏”，优先选取好的规则尝试
  - 总的时间花费不易定论
  - 最好情况复杂性很低：当控制系统掌握较多的有关解的知识时，则回溯次数大为减少，效率高。
  - 最坏情况复杂性很高：当控制系统一点也没掌握有关解的知识时，则规则选取是任意的，回溯次数高，效率低。
- 为了避免进入无限循环的尝试，可设置搜索深度限制强行回溯
  - 问题：太深：效率低；太浅：可能找不到解。
- 当深度限制可变时，通常能找到解
- 是实际用得多、应用最广的一种搜索策略

Forward & Backward Chaining

- Forward Chaining
  - 即正向产生式系统，也称为数据驱动（Data Driven）
  - 从初始状态出发，不断地应用规则，直到产生目标状态为止。
- Backward Chaining
  - 即反向产生式系统，也称为目标驱动（Goal Driven）
  - 从目标状态出发，利用反向的产生式规则不断地产生子目标，直到产生出与初始状态相同的子目标为止。
- 执行过程中的操作可比命题/谓词逻辑更灵活、复杂

专家系统简述

专家系统简述

- In artificial intelligence, an expert system is a computer system emulating the decision-making ability of a human expert.

WIKIPEDIA
The Free Encyclopedia

- 专家系统通过利用主要以if-then规则表示的知识进行推理，而非通常的程序性代码，来解决复杂问题。

- 产生式是构建专家系统的一种重要方式

专家系统简述

- 在人工智能领域，专家系统是较早取得实际成果，并获得商业回报的分支领域
- DENDRAL系统是第一个成功投入使用的专家系统，1965年由斯坦福大学开始研发，1968年研制成功。
  - 它的作用是分析质谱仪的光谱，应用化学知识，帮助化学家判定物质的分子结构。
- DENDRAL系统研发团队核心包括人工智能科学家爱德华·费根鲍姆（Edward Feigenbaum）和遗传学家约书亚·莱德伯格（Joshua Lenderberg）
- “In DENDRAL, knowledge is uniformly represented in a very general form: production rules.”

专家系统特点

- 符号化推理：知识应被符号化表示
  - 可能的表示方法：规则、框架（Frame）、描述逻辑
- 处理的问题较复杂
  - 一般针对较窄、专一的领域
  - 以专家水平的能力解决复杂问题
- 高质量的效果
- 知识与处理（推理）分离
  - 方便根据问题选取知识；重用；方便知识更新
- 可对其输出进行解释

专家系统特征

1. 具有专家水平的领域知识
2. 能进行有效的推理
3. 具有获取知识的能力
4. 具有灵活性
5. 具有透明性
6. 具有交互性
7. 具有实用性
8. 具有一定的复杂性及难度

《知识表示与处理》

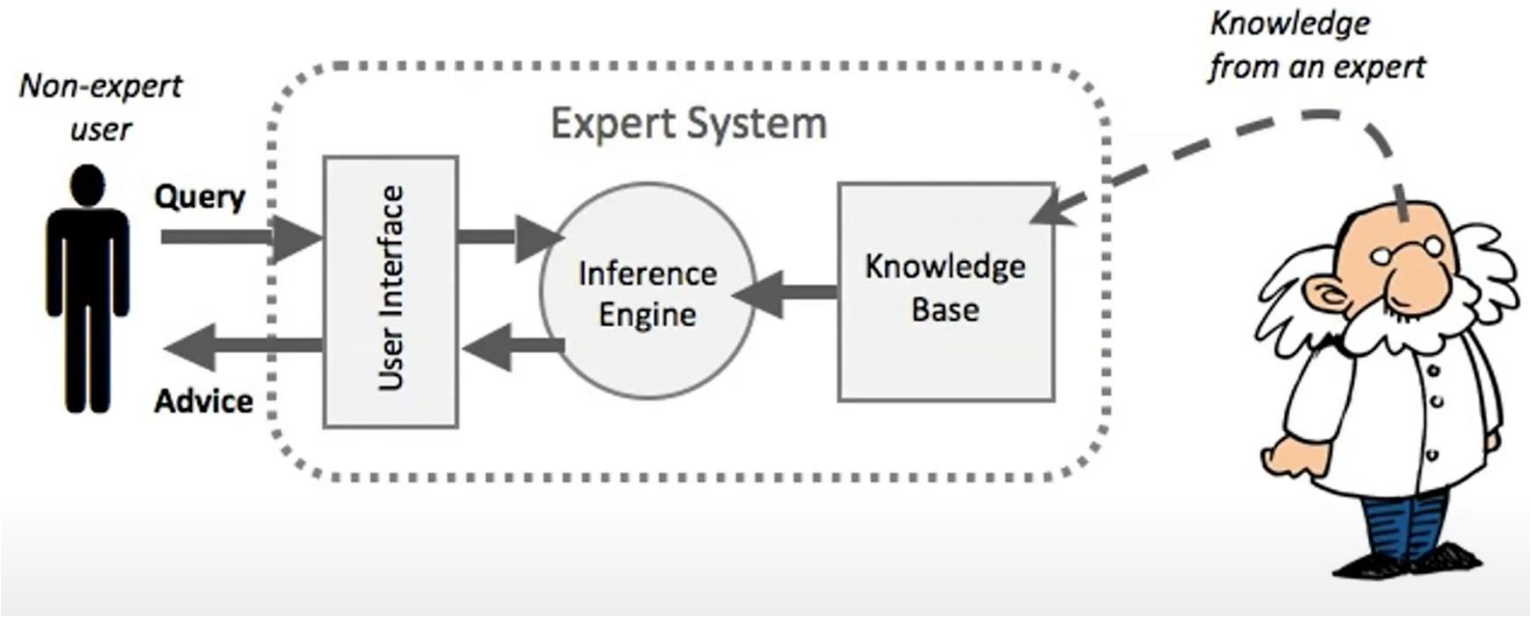
专家系统简述 – 分类

- 解释专家系统
  - 如卫星图像云图分析、集成电路分析、化学结构分析等
- 预测专家系统
  - 如恶劣天气预报、农作物病虫害预报等
- 设计专家系统
  - 如计算机结构设计专家系统、大规模集成电路设计专家系统等
- 规划专家系统
  - 军事指挥调度系统、火车运行调度专家系统
- 监视专家系统
  - 如防空监视与报警
- 控制专家系统
  - 如控制交通管制、商业管理
- 调试专家系统
  - 如维修站调试维修设备
- 教学专家系统
  - 如积分符号与定理证明系统
- 维修专家系统
  - 如有些电视维护修理系统

专家系统简述

Category	Problem addressed	Examples
Interpretation	Inferring situation descriptions from sensor data	Hearsay (speech recognition), PROSPECTOR
Prediction	Inferring likely consequences of given situations	Preterm Birth Risk Assessment[68]
Diagnosis	Inferring system malfunctions from observables	CADUCEUS, MYCIN, PUFF, Mistral,[69] Eydenet,[70] Kaleidos,[71] GARVAN-ES1[72] [73] [74]
Design	Configuring objects under constraints	Dendral, Mortgage Loan Advisor, R1 (DEC VAX Configuration), SID (DEC VAX 9000 CPU)
Planning	Designing actions	Mission Planning for Autonomous Underwater Vehicle[75]
Monitoring	Comparing observations to plan vulnerabilities	REACTOR[76]
Debugging	Providing incremental solutions for complex problems	SAINT, MATHLAB, MACSYMA
Repair	Executing a plan to administer a prescribed remedy	Toxic Spill Crisis Management
Instruction	Diagnosing, assessing, and correcting student behaviour	SMH.PAL,[77] Intelligent Clinical Training,[78] STEAMER[79]
Control	Interpreting, predicting, repairing, and monitoring system behaviors	Real Time Process Control,[80] Space Shuttle Mission Control,[81] Smart Autoclave Cure of Composites[82]

大致框架



- 专家系统的目标
  - 代替或辅助人类专家解决现实问题

其他可能组件

- 知识库编辑器
  - 供专家或知识工程师向知识库添加知识或更改知识

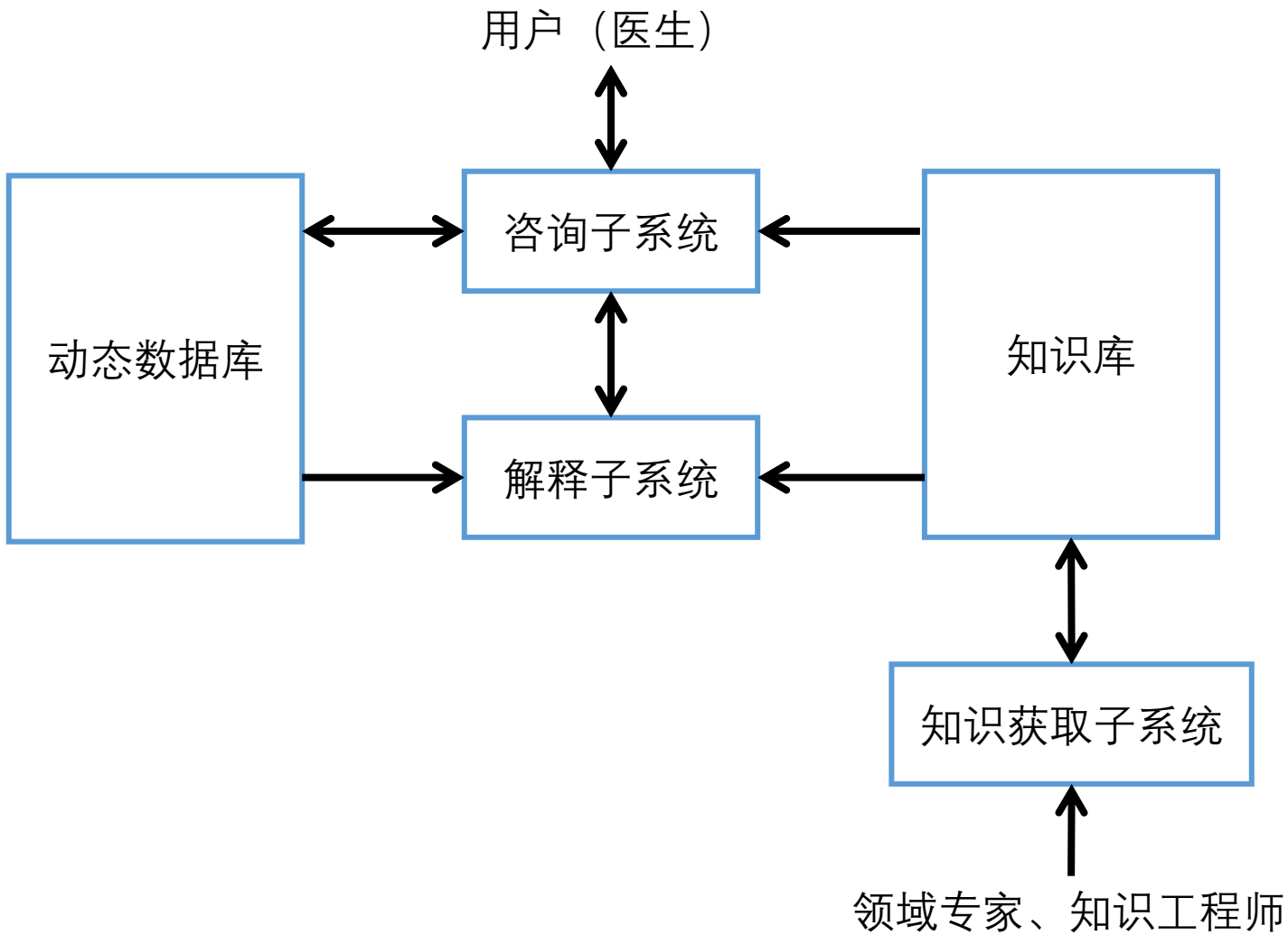
- 解释系统
  - 解释系统的输出
  - 如：医生需知道一个医疗专家系统做出某种诊断的原因
  - 对基于规则的系统，可基于应用规则达到最终目标的路径进行解释
  - 可解释性是相比基于神经网络系统的重要优势

MYCIN专家系统

- 在DENDRAL之后，斯坦福大学又开发了用于帮助医生诊断感染性疾病的MYCIN专家系统，MYCIN系统的成功标志着人工智能进入医疗系统这一重要的应用领域。

- MYCIN的开发始于1972年，终于1978年，最终成为一个性能较高，功能完善的实用系统。该系统在专家系统的发展史中占有很重要的地位。

MYCIN专家系统



- 动态数据库
  - 存放患者的有关数据
- 咨询子系统
  - 推理和人机交互
- 解释子系统
  - 解释系统做出的决策
- 知识库
  - 诊治疾病的知识

MYCIN的运行

- 向医生提问一系列关于病人的问题
- 最后输出
  - 一个包含可能感染的细菌的列表（每条有置信度、诊断原因）
  - 推荐的治疗方案

- 通过对知识库中的规则backward chaining实现

MYCIN的规则

- 知识库中含约600条规则

```
<rule>       ::= <premise> <action> | <premise> <action>
                 <else>

<premise>    ::= ($AND <condition> . . . <condition>)

<condition>  ::= (<func1> <context> <parameter>) |
                 (<func2> <context> <parameter> <value>) |
                 (<special-func> <arguments>) |
                 ($OR <condition> . . . <condition>)

<action>     ::= <concpart>

<else>       ::= <concpart>

<concpart>   ::= <conclusion> | <actfunc> |
                 (DO-ALL <conclusion> . . . <conclusion>) |
                 (DO-ALL <actfunc> . . . <actfunc>)
```

MYCIN的规则

一条普通规则：

IF
1）病原体的鉴别名不确定，且
2）病原体来自血液，且
3）病原体的染色体是革兰氏阴性，且
4）病原体的形态是杆状的，且
5）病原体呈赭色
THEN
该病原体的鉴别名是假单胞细菌，可信度为0.4

Goal Rule：

IF
1）有一种需要治疗的病原体，且
2）可能还有其他需要治疗的病原体，尽管它们还没有从当前的培养物中被分离出来
THEN
1）给出能有效抑制需治疗的病原体的治疗方案
2）选出最佳治疗方案

其他类型的专家系统

- MYCIN属于基于规则的专家系统（Rule-based Expert System），此外还有：
- 模糊专家系统（Fuzzy Expert System）
  - 应用模糊逻辑
- 基于框架的专家系统（Frame-based Expert System）
  - 使用框架表示法
- 混合专家系统（Hybrid Expert System）
  - 结合规则和神经网络模型

专家系统的优缺点

- 优点
  - 自然性
    - 自然的、人类可读的知识表示方式
  - 灵活性
    - 知识与推理分离
  - 可综合多个人类专家的知识或经验
  - 可以利用不完整或不确定的知识

- 缺点
  - 可能无法得出结论或知道是否存在一个结论
  - 只在某个较窄的领域有用
  - 自身无法学习或创新
  - 规则间的关系不清楚
  - 人类专家的经验不一定都准确

END