

第二部分： 智能自治Agent

章宗长

2023年3月7日

内容安排

2.1

智能Agent

2.2

智能Agent的体系结构

2.3

演绎推理Agent

2.4

实用推理Agent

2.5

反应式Agent

2.6

混合式Agent

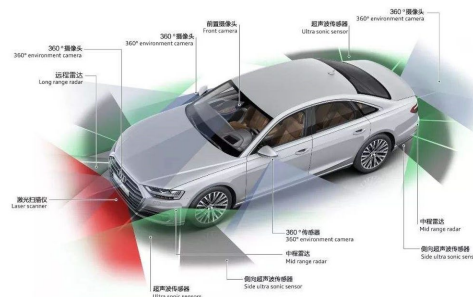
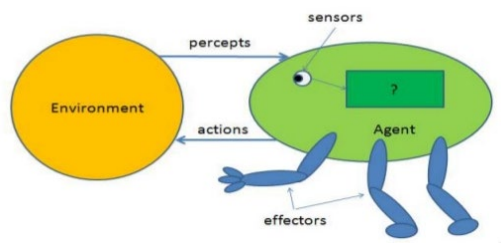
智能Agent的体系结构

- 智能Agent的抽象体系结构
- 告诉Agent要做什么
- 智能Agent的实现型体系结构

Agent的结构

Agent = 体系结构 + 程序

- **程序**实现的是把感知信息映射到行动的函数
- **体系结构**（Architecture）是构造Agent的方法论，研究如何用软件或硬件的方法实现Agent



- 一般而言，体系结构为程序提供来自传感器的感知信息，运行程序，并把程序计算出来的行动决策送达执行器

Agent的体系结构 & 抽象体系结构

■ Agent体系结构的研究内容：

- Agent的内部模块集合如何组织起来
- 它们的相互作用关系如何
- Agent感知到的信息如何影响它的行为和内部状态
- 如何将这些模块用软件或硬件的方式形成一个有机整体

■ Agent的抽象体系结构

- 独立于Agent体系结构的具体实现技术
- 与具体的支撑平台和环境没有任何联系
- 有助于在一个较高抽象层次上描述和分析构成Agent的部件、部件之间的关系以及不同Agent的性质

Agent的抽象体系结构

- 假设环境是任何离散的瞬时状态的有限集合 E :

$$E = \{e, e', \dots\}$$

- 假设Agent有一个可执行动作的清单，它们改变环境的状态:

$$\text{(有限的) 动作集合 } Ac = \{\alpha, \alpha', \dots\}$$

- Agent在环境中的一次运行 (run) r 是环境状态与动作交替的一个序列:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_{u-1}} e_u$$

运行 (run)

- Agent在环境中的一次运行 r :

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_{u-1}} e_u$$

- 设:

- \mathcal{R} 是所有可能的（ E 和 Ac 上的）有限序列集合
- \mathcal{R}^{Ac} 是以动作结束的序列所组成的子集
- \mathcal{R}^E 是以状态结束的序列所组成的子集

- 用 r, r', \dots 代表 \mathcal{R} 的成员

环境

- **状态转移函数**表示Agent的动作作用于环境的效果：

$$\tau: \mathcal{R}^{Ac} \rightarrow 2^E$$

- 假设环境是**历史依赖的**（history dependent）
 - 环境的下一个状态不仅仅是由Agent执行的动作和当前的状态决定，早时Agent所做的动作对决定当前状态也起部分作用
- 环境可以是**不确定的**（non-deterministic）
 - 在某个状态下执行一个动作的结果具有不确定性
- 如果 $\tau(r) = \emptyset$ （假设 r 是由一个动作作为结束），则 r 不存在可能的后继状态（系统结束 r ）

环境、Agent模型

- 形式上，环境 Env 是一个三元组 $Env = \langle E, e_0, \tau \rangle$ ：
 - E 是环境状态的集合
 - $e_0 \in E$ 是初始状态
 - τ 是状态转移函数

- **Agent模型**是一个把运行映射到动作的函数：

$$Ag : \mathcal{R}^E \rightarrow Ac$$

- Agent根据到当前为止的历史决定执行什么动作
- 设 \mathcal{AG} 是所有Agent的集合

系统

- 系统是Agent和环境构成的对
- 任何系统都有与之相关的可能的运行集合
- $\mathcal{R}(Ag, Env)$ 表示Agent在环境 Env 中的运行的集合
- 假设 $\mathcal{R}(Ag, Env)$ 只包含可以结束的运行
 - 即，运行 r 不存在可能的后继状态： $\tau(r) = \emptyset$

- 形式上，序列：

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \dots)$$

表示Agent Ag 在环境 $Env = \langle E, e_0, \tau \rangle$ 中的一次运行，如果：

1. e_0 is the initial state of Env ;
2. $\alpha_0 = Ag(e_0)$; and
3. for all $u > 0$,
 $e_u \in \tau((e_0, \alpha_0, \dots, \alpha_{u-1}))$, and
 $\alpha_u = Ag((e_0, \alpha_0, \dots, e_u))$.

- 两个Agent (Ag_1 和 Ag_2) 对环境 Env 是行为等价的，当且仅当 $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$
- 两个Agent (Ag_1 和 Ag_2) 是完全行为等价的，当且仅当它们对所有环境是行为等价的

纯反应式Agent

- 决策完全基于当前状态
- 行动可以表示成函数：

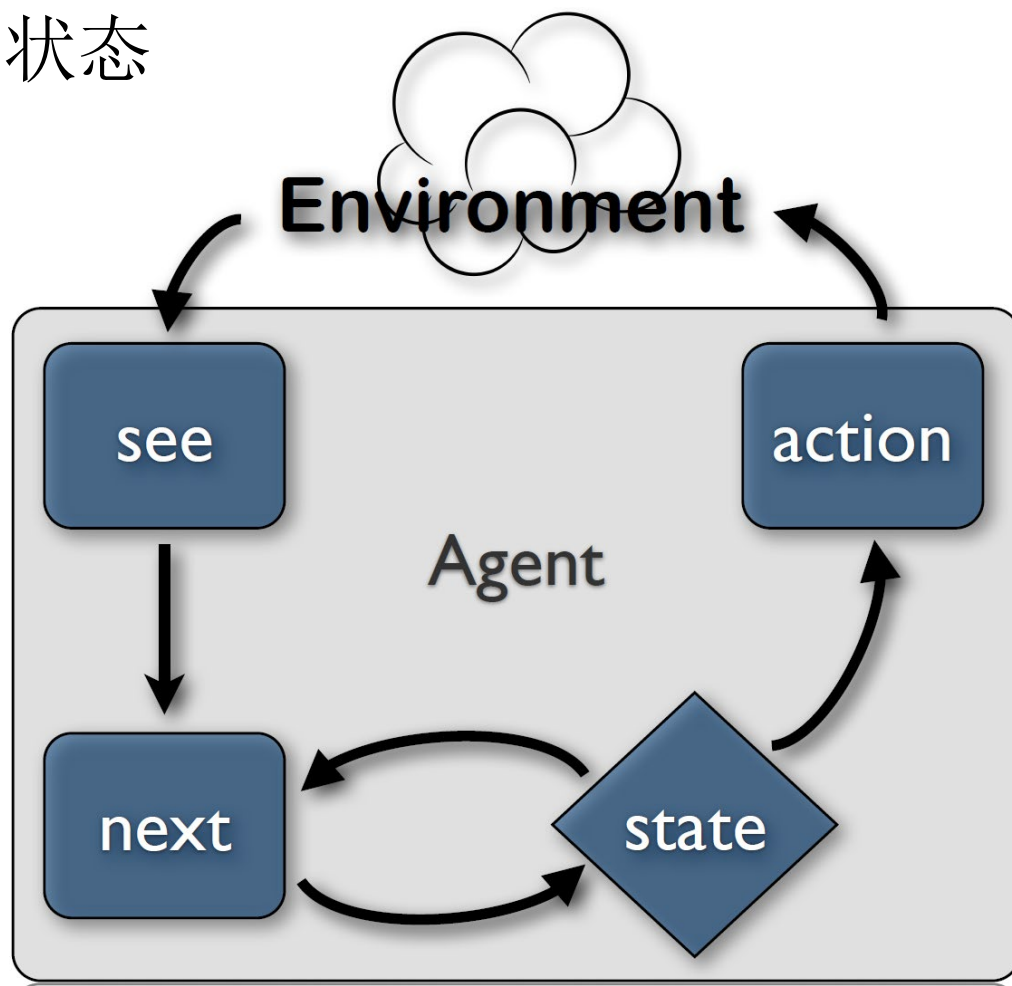
$$action : E \rightarrow Ac$$

- 温度控制器就是一个纯反应式Agent:

$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise} \end{cases}$$

有状态的Agent

- Agent具有内部状态



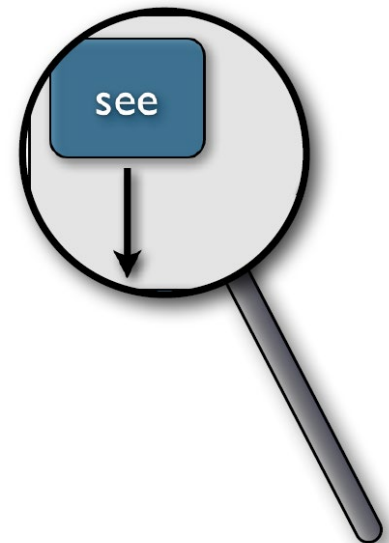
感知

- 感知函数see代表Agent观察环境的能力

- 函数see的输出是一种感知：

$$see : E \rightarrow Per$$

- 实现从环境状态到感知的映射
- 这些Agent有内部数据结构，一般用来记录环境状态和环境的历史
- 设 I 是Agent的所有内部状态的集合



行动选择函数和下一个状态函数

- 行动选择函数 $action$ 代表 Agent 的决策过程
- 函数 $action$ 实现内部状态到动作的映射：

$$action : I \rightarrow Ac$$



- 下一个状态函数 $next$ 实现从内部状态和感知到内部状态的映射：

$$next : I \times Per \rightarrow I$$



Agent控制回路

1. Agent starts in some initial internal state i_0 .
2. repeat forever:
 - Observe environment state, and generate a percept through *see*(...).
 - Update internal state via *next* function,
 - Select action via *action*(...).
 - Perform action.

智能Agent的体系结构

- 智能Agent的抽象体系结构
- 告诉Agent要做什么
- 智能Agent的实现型体系结构

Agent和任务

- 建造Agent是为了执行任务
- 我们需要用某种方式详细说明要执行的任务
- 方式1：告诉Agent怎么去做
 - 优点：对Agent要做什么没有任何不确定性
 - 缺点：我们必须自己考虑究竟如何完成这个任务
- 方式2：告诉Agent要做什么，而不告诉它怎么去做
 - 不直接定义任务，而是使用某种性能度量
 - 任务说明：使用效用（Utility）、使用谓词（Predicate）

状态的效用函数

- 效用是表示状态有多“好”的数值

- 效用越高，状态越好

- 状态的效用函数：

$$u : E \rightarrow \mathbb{R}$$

给每一个环境状态一个实数值

- Agent的任务是使得全局效用最大化

- 一次运行的全局效用的不同定义方式：

- 遇到的最差（好）的状态的效用、所有遇到的状态的平均值/累积值.....

- 缺点：当给每个状态赋予一个效用后，很难使用长期的观点

- 一种可能的方法：给未来状态的效用乘上折扣

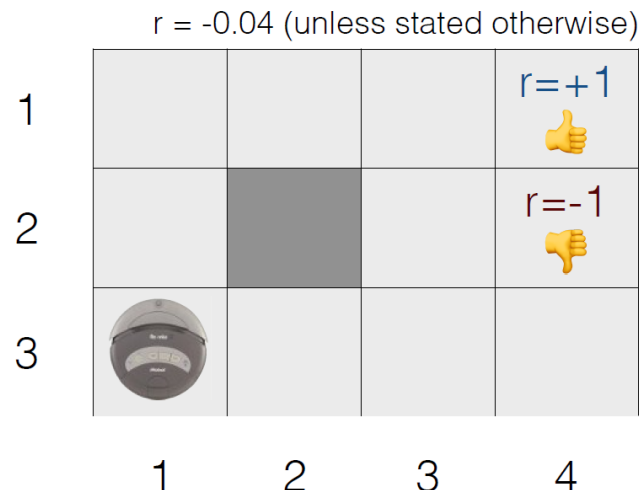
强化学习的做法

例子：栅格世界

- **目标：**选择动作以最大化未来的奖赏
 - 每个动作导致到达一个赋予了立即奖赏值的状态
 - 奖赏具有延迟性
 - 好的策略会为了得到更多的长期奖赏而牺牲立即奖赏

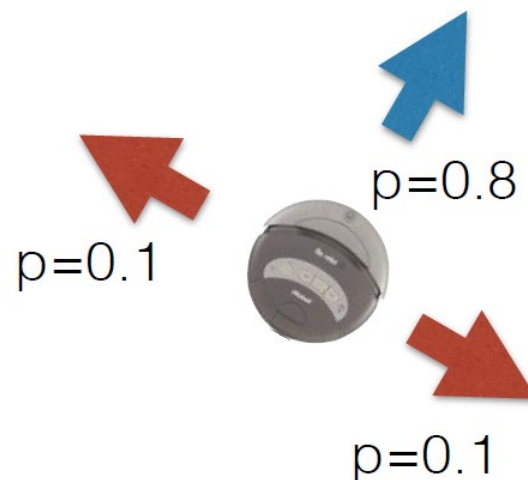
- 假设环境是**确定性的**

- Agent能以概率1达到想去方向的相邻栅格
- 最优策略：[上, 上, 右, 右, 右]
- 累积奖赏： $-0.04 \times 4 + 1 = 0.84$
- 负奖赏 -0.04 使得Agent尽快到达目标



- 假设环境是**不确定性的**

- 例如，Agent能以概率0.8达到想去方向的相邻栅格，各以概率0.1达到垂直方向的相邻栅格



- 使用策略[上, 上, 右, 右, 右]成功到达目标的概率

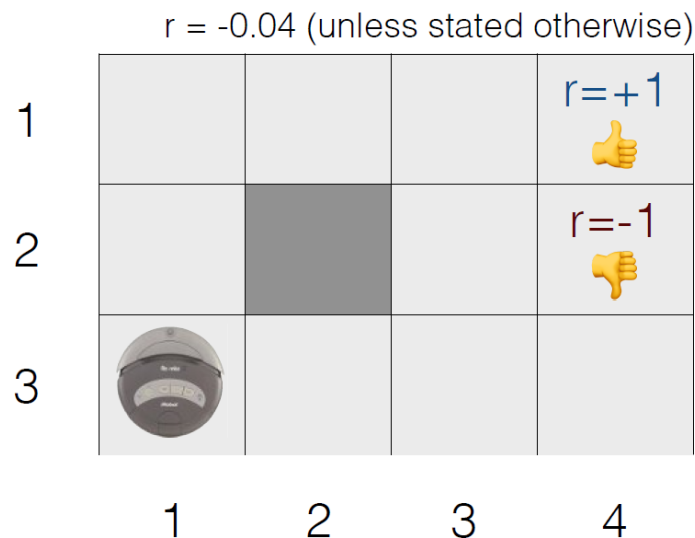
- 可能按计划路线到达目标

$$p_1 = 0.8^5 = 0.32768$$

- 还可能偶然到达目标

$$p_2 = 0.1^4 \times 0.8 = 0.00008$$

- 总概率 $p = p_1 + p_2 = 0.32776$



运行的效用函数

- 另一种可能的方法：不是把效用赋给单个状态，而是把效用赋给这些状态的运行：

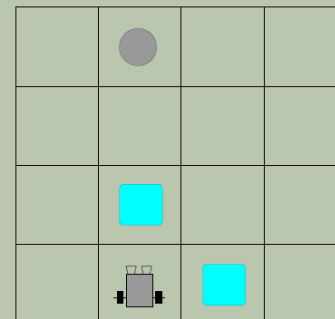
$$u : \mathcal{R} \rightarrow \mathbb{R}$$

- 这种方法使用了长期的观点
- 效用如何得到呢？
 - 在有些任务中，比较容易得到，比如：瓦片世界
 - 在另外一些任务中，比较难得到，用“要实现的目标”更方便，由此提出了用谓词说明任务的方法

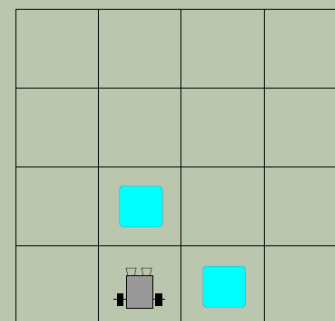
瓦片世界

- 仿真的二维网格环境
 - 有Agent、障碍物、洞穴
- Agent可以向上、下、左、右四个方向运动
 - 如果Agent在一个瓦片旁边，则可以推这个瓦片
- Agent必须用瓦片填满洞穴，它得分的点数通过用瓦片填满的洞穴计算
- 随着洞穴的随机出现和消失，瓦片世界动态变化

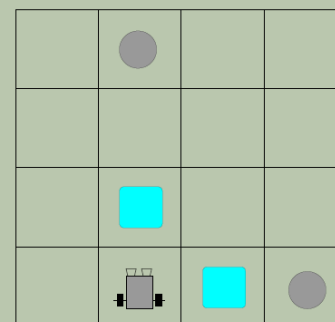
The agent starts to push a tile towards the hole.



But then the hole disappears!!!



Later, a much more convenient hole appears (bottom right)



瓦片世界的效用函数

- Agent在一次运行 r 中的效用函数:

$$u(r) \hat{=} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- Agent在运行 r 中填满的洞穴越多，效用 $u(r)$ 越高
 - 如果Agent成功填满了所有出现的洞穴，则 $u(r) = 1$
 - 如果Agent没有能成功地填满一个洞穴，则 $u(r) = 0$
- 瓦片世界可以使我们检验Agent的**反应性**
 - Agent需要有对环境变化的反应能力，以利用出现的机会

期望效用

- 用 $P(r \mid Ag, Env)$ 表示把 Agent Ag 放在环境 Env 中出现运行 r 的概率

- 显然有：

$$\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1$$

- 则，Agent Ag 在（给定 P 和 u 的）环境 Env 中的期望效用为：

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env)$$

最优Agent

- 在环境 Env 中的最优Agent是使期望效用最大化的Agent:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} EU(Ag, Env)$$

- 当然，一个Agent是最优的并不意味着它将是最好的；只是在平均方面，我们可以期望它是做得最好的

例子

- 考虑环境 $Env_1 = \langle E, e_0, \tau \rangle$ ，具体定义如下：

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2\}$$

$$\tau(e_0 \xrightarrow{\alpha_1}) = \{e_3, e_4, e_5\}$$

- 这个环境中有两个可能的Agent:

$$Ag_1(e_0) = \alpha_0$$

$$Ag_2(e_0) = \alpha_1$$

- 不同运行的概率：
$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.4$$
$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.6$$
$$P(e_0 \xrightarrow{\alpha_1} e_3 \mid Ag_2, Env_1) = 0.1$$
$$P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2$$
$$P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.7$$

- 假设效用函数 u_1 的定义如下：

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8$$

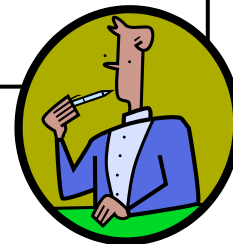
$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 11$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_3) = 70$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_4) = 9$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_5) = 10$$

- 求不同Agent的期望效用，并指出最优Agent



解：

由期望效用的计算公式

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r)P(r \mid Ag, Env)$$

可得：

$$Ag_1 \text{的期望效用} = (0.4 \times 8) + (0.6 \times 11) = 9.8$$

$$Ag_2 \text{的期望效用} = (0.1 \times 70) + (0.2 \times 9) + (0.7 \times 10) = 15.8$$

因此，最优Agent为 Ag_2

受限的最优Agent

- 有些Agent不能在一些机器（计算机）上实现

- 用 \mathcal{AG}_m 表示可以在机器 m 上实现的Agent:

$$\mathcal{AG}_m = \{Ag \mid Ag \in \mathcal{AG} \text{ and } Ag \text{ can be implemented on } m\}$$

- 用 Ag_{bopt} 表示可以在机器 m 上实现的受限最优Agent:

$$Ag_{bopt} = \arg \max_{Ag \in \mathcal{AG}_m} EU(Ag, Env)$$

使用谓词的任务说明

- 使用谓词来代替效用函数
- 如果效用的取值范围是集合 $\{0, 1\}$ ，则可以说效用函数 $u : \mathcal{R} \rightarrow \mathbb{R}$ 是一个谓词
- 使用 Ψ 表示谓词说明：

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$

- 用 $\Psi(r)$ 表示 $r \in \mathcal{R}$ 满足 Ψ
- 当且仅当 $u(r) = 1$ ， $\Psi(r)$ 为真

任务环境

- 一个任务环境是一个 $\langle Env, \Psi \rangle$ 对，其中 Env 是环境， Ψ 是定义在运行上的谓词
- 用 \mathcal{TE} 表示所有任务环境的集合
- 一个任务环境说明了：
 - Agent 所在的系统的特性
 - 评价 Agent 失败或者成功完成任务的标准
- 用 $\mathcal{R}_\Psi(Ag, Env)$ 表示 Agent Ag 在环境 Env 中的满足 Ψ 的所有运行的集合：

$$\mathcal{R}_\Psi(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}$$

- 如果有:

$$\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$$

则称Agent Ag 在任务环境 $\langle Env, \Psi \rangle$ 中获得成功



在**最坏**情况下对成功的定义

如果 Ag 在 Env 中的每次运行都满足说明 Ψ , 则 Ag 在任务环境 $\langle Env, \Psi \rangle$ 中是成功的, 即:

$$\forall r \in \mathcal{R}(Ag, Env) \text{ we have } \Psi(r)$$

- 对Agent成功的**乐观**定义: 至少有一次运行满足 Ψ

$$\exists r \in \mathcal{R}(Ag, Env) \text{ such that } \Psi(r)$$

成功概率

- 如果环境是不确定的，则 τ 返回的是一个可能的状态的集合
 - 可以定义这个状态集合的概率分布
- 用 $P(r \mid Ag, Env)$ 表示把Agent Ag 放在环境 Env 中出现运行 r 的概率
- 则 Ag 在环境 Env 中满足 Ψ 的概率 $P(\Psi \mid Ag, Env)$ 为

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_{\Psi}(Ag, Env)} P(r \mid Ag, Env)$$

实现型 & 维护型任务

- 两种最常见的任务类型是**实现型任务**和**维护型任务**
- 如果能区分环境状态 E 的子集 G ，使得一旦 G 的一个或多个状态出现在运行 r 中，就有 $\Psi(r)$ 为**真**，则**由谓词 Ψ 说明的任务是实现型任务**
 - 子集 G 的成员被称为**目标（goal）状态**
- 如果能找到环境状态 E 的子集 B ，使得一旦 B 的一个或多个状态出现在运行 r 中，就有 $\Psi(r)$ 为**假**，则**由谓词 Ψ 说明的任务是维护型任务**
 - 子集 B 的成员被称为**失败（failure）状态**

合成Agent

- Agent合成是自动程序设计问题
 - 目标：使程序输入任务环境，并从该任务环境自动产生一个可以在这个环境中取得成功的Agent

- Agent合成算法syn可以理解成一个函数：

$$syn : \mathcal{TE} \rightarrow (\mathcal{AG} \cup \{\perp\})$$

\perp 类似Java中的null

- 合成算法是可靠的（soundness）：如果能返回一个Agent，则这个Agent在作为输入的任务环境中是成功的

- 形式化： $syn(\langle Env, \Psi \rangle) = Ag$ implies

$$\mathcal{R}(Ag, Env) = \mathcal{R}_{\Psi}(Ag, Env)$$

- 问题：即使存在成功的Agent，仍可能返回 \perp

- 合成算法是**完备的**（completeness）：如果存在一个Agent在作为输入的任务环境中是成功的，则保证返回一个Agent

- 形式化：

$$\begin{aligned} \exists Ag \in \mathcal{AG} \text{ s.t. } \mathcal{R}(Ag, Env) = \mathcal{R}_{\Psi}(Ag, Env) \\ \text{implies } syn(\langle Env, \Psi \rangle) \neq \perp \end{aligned}$$

- 问题：即使存在成功的Agent，仍可能返回失败的Agent

- 理想情况下，我们希望得到一个既可靠又完备的合成算法

智能Agent的体系结构

- 智能Agent的抽象体系结构
- 告诉Agent要做什么
- 智能Agent的实现型体系结构

Agent的体系结构

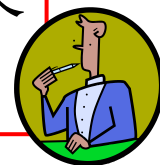
■ 研究内容:

- Agent的内部模块集合如何组织起来
- 它们的相互作用关系如何
- Agent感知到的信息如何影响它的行为和内部状态
- 如何将这些模块用软件或硬件的方式形成一个有机整体


■ Agent的抽象体系结构

- 状态、动作、运行、环境、Agent模型、系统
- 纯反应式Agent、有状态的Agent

如何用计算机系统可以理解和处理的手段来表示Agent的状态，实现构成Agent的各个部件？



Agent的实现型体系结构

- Agent体系结构的设计和实现要**进一步考虑**:
 - **如何**给出相应的数据结构来**表示**Agent的**内部状态**（如果有的话）？
 - **如何实现**构成Agent的**各个部件**，并为它们**设计**相应的**实现算法**？
 - 如：感知部件、状态转换部件、动作选择部件等
 - 实现型体系结构的类型：
 - 知识型体系结构
 - 思维型体系结构
 - 反应型体系结构
 - 混合型体系结构
 - 智能Agent的类型：
 - 演绎推理Agent
 - 实用推理Agent
 - 反应式Agent
 - 混合式Agent
- 

演绎推理Agent

- 具有表示Agent内部状态的部件
- 内部状态被定义为Agent所拥有的知识
- 各个知识条目用逻辑语言来表示
 - 命题逻辑：假设现实世界是由事实构成的
 - 一阶逻辑：假设现实世界是由事实、对象、关系构成的
 - 高阶逻辑：可以把一阶逻辑中的关系本身也视为对象
- Agent的动作选择是一个基于其知识的逻辑演绎或定理证明过程

实用推理Agent

- 具有表示Agent内部状态的部件
- 内部状态表现为Agent所具有的思维状态（mental state）
- Agent的动作选择是一个针对其内部思维状态的实用推理过程
- 实用推理
 - 慎思的过程（deliberation）：想清楚决定做什么样的事
 - 目标手段推理的过程（means-ends reasoning）：根据目标（想做的事）来决定实现该目标所需的手段（需执行的动作）

反应式Agent

- 不具有表示Agent内部状态的部件
- 内部定义了一系列的反应式规则
- 根据感知到的环境信息，激发相应的反应式规则来执行
- Agent的动作选择是一个从情景（situation）到动作的映射

混合式Agent

- 在内部集成了多个不同类型的实现型体系结构
- 组合了多个层次的动作选择部件
- 不同层次的实现型体系结构或者单独运作或者彼此相互作用
 - 共同对感知到的环境输入做出多种方式的响应
 - 如：反应式响应、慎思式响应
- 在运行时能够表现出多种形式的行为特征
 - 如：自治性、反应性、预动性、社会性

小结

■ 智能Agent的抽象体系结构

- 状态，动作，运行，环境，Agent模型，系统
- 纯反应式Agent，有状态的Agent

■ 告诉Agent要做什么

- （状态的、运行的）效用，期望效用，最优Agent，受限最优Agent
- 谓词说明，任务环境，实现型 & 维护型任务
- Agent合成算法（可靠性 & 完备性）

■ 智能Agent的实现型体系结构

- 知识型（演绎推理Agent），思维型（实用推理Agent），反应型（反应式Agent），混合型（混合式Agent）

内容安排

2.1

智能Agent

2.2

智能Agent的体系结构

2.3

演绎推理Agent

2.4

实用推理Agent

2.5

反应式Agent

2.6

混合式Agent

演绎推理Agent

- 作为定理证明器的Agent
- 面向Agent的程序设计
- 并发MetateM

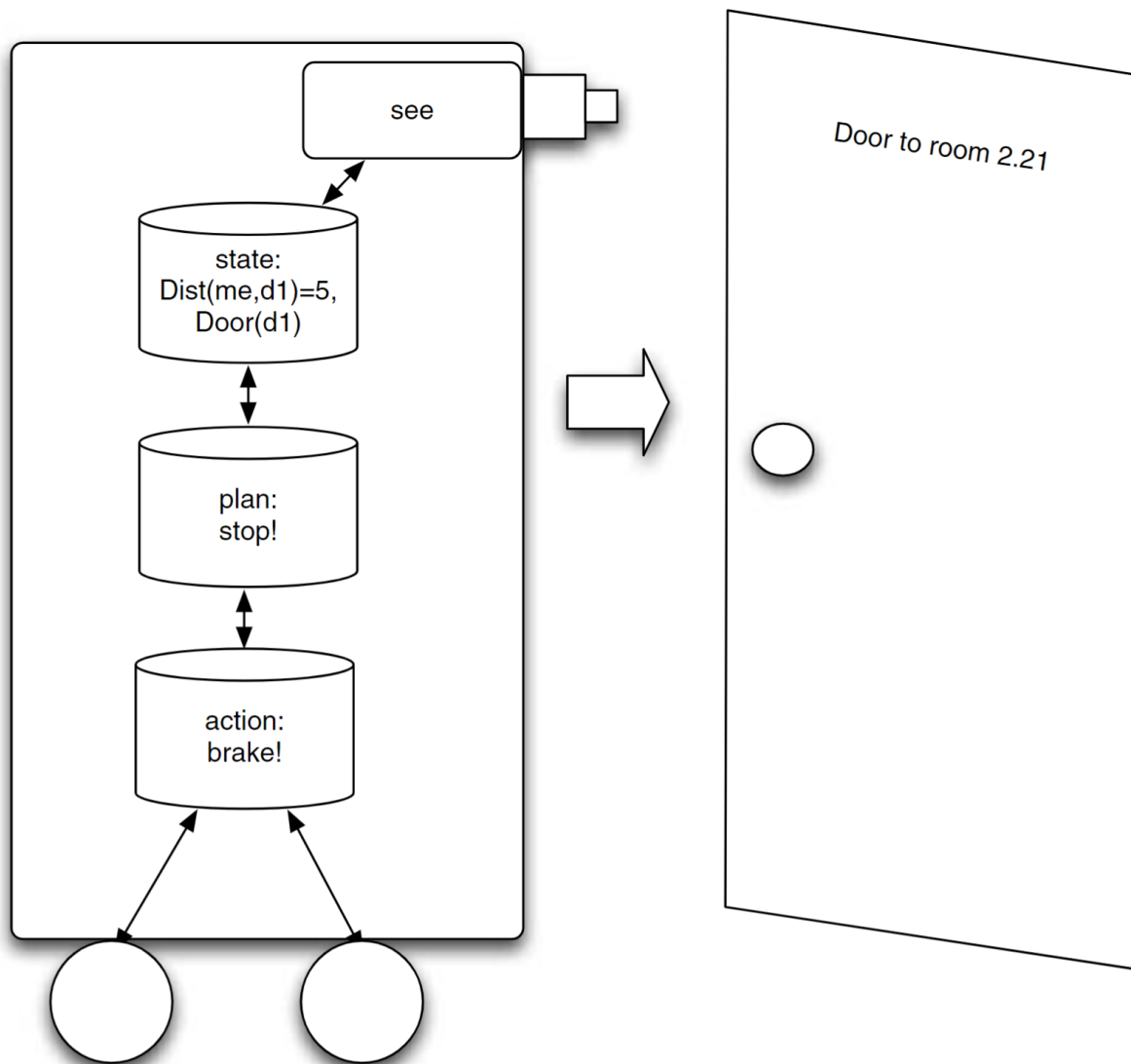
符号 & 演绎推理Agent

- 符号推理Agent: 通过
 - 用符号表示系统环境和期望的行为
 - 用句法规则处理这种表示来产生智能行为
- 演绎推理Agent
 - 符号表示: 一些逻辑公式
 - 句法规则处理: 逻辑演绎或定理证明

一个演绎推理Agent的例子

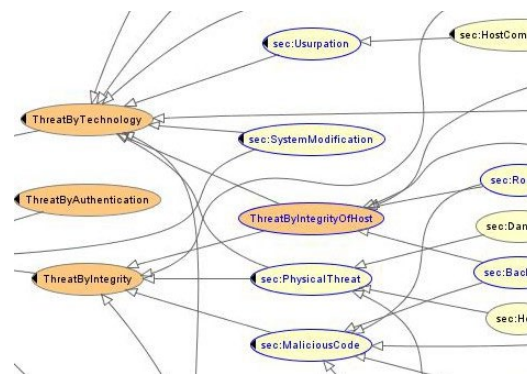
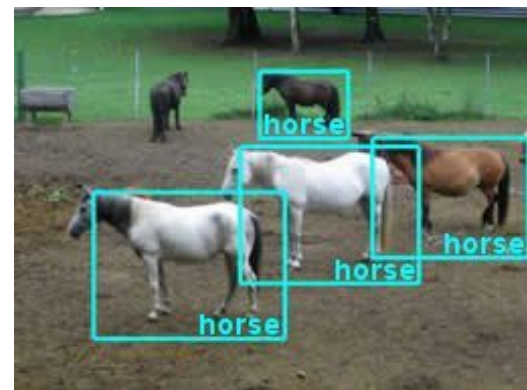
基于一阶逻辑的
内部状态表示

逻辑演绎



两个关键问题

- **转换问题**：把现实世界转换成这个世界的准确恰当的符号表示的问题
 - 导出了视觉、语音理解、学习等方面的研究
- **表示和推理问题**：用符号表示信息并且让Agent使用这种表示进行处理和推理的问题
 - 导出了知识表示、自动推理、自动规划等方面的研究



内部状态的表示

- 内部状态 DB 是由经典一阶谓词逻辑公式组成的数据库
 - 设 L 是经典一阶谓词逻辑公式集合
 - $D = 2^L$ 是数据库 L 的集合
 - DB 是 D 中的一个元素
- 例如，一个Agent的数据库可能包含下列公式：
 - $Open(value221)$
 - $Temperature(reactor4726, 321)$
 - $Pressure(tank776, 28)$

各个部件

- 感知函数

$$see : S \rightarrow Per$$

- 状态转移函数

$$next : D \times Per \rightarrow D$$

- 由数据库和感知映射到新的数据库

- 动作选择函数

$$action : D \rightarrow Ac$$

- 根据演绎规则定义
 - 演绎规则：一些简单的逻辑推理规则

动作选择函数的伪代码

- $DB \vdash_{\rho} \varphi$: 可以从数据库 DB 仅使用演绎规则 ρ 证明公式 φ

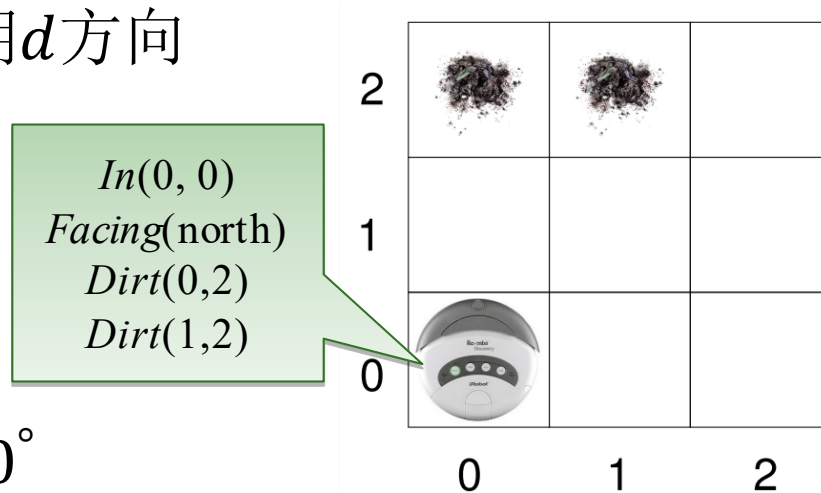
```
for each  $\alpha \in Ac$  do
  if  $\frac{DB \vdash_{\rho} Do(\alpha)}{\text{return } \alpha}$  then
    end-if
end-for
for each  $\alpha \in Ac$  do
  if  $\frac{DB \not\vdash_{\rho} \neg Do(\alpha)}{\text{return } \alpha}$  then
    end-if
end-for
return null
```

通过定理证明的方式试图找到要执行的动作

通过定理证明的方式试图找到不被明确禁止的动作

例子：真空吸尘器世界

- 吸尘Agent的**目标**：清扫网格中的垃圾
- 使用**3个领域谓词**来描述吸尘Agent的**内部状态**（知识库）
 - $In(x, y)$: Agent处于网格中的 (x, y) 位置
 - $Dirt(x, y)$: 在网格的位置 (x, y) 处有垃圾
 - $Facing(d)$: Agent当前面朝 d 方向
- 吸尘Agent的**动作集合**
 - Forward: 向前移动
 - Suck: 打扫垃圾
 - Turn: 沿顺时针方向转动 90°



■ 吸尘Agent的感知函数see

- Agent在网格中的位置
- Agent所在位置是否存在垃圾
- Agent的朝向

■ 吸尘Agent的状态转移函数next

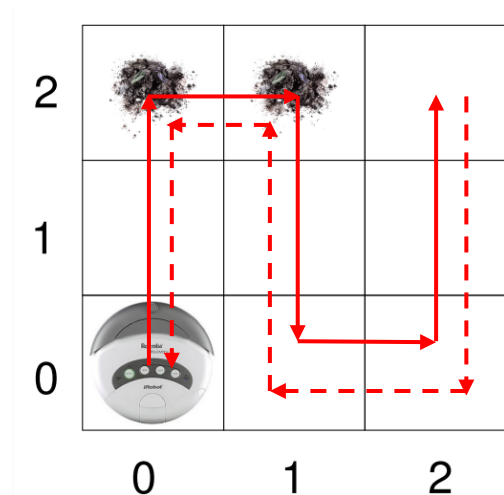
$$next(DB, p) = (DB \setminus old(DB)) \cup new(DB, p)$$

- $DB \setminus old(DB)$: 在知识库中删除旧知识后剩余的条目
- $new(DB, p)$: 根据知识库 DB 和感知部件所接收的感知输入 p 而产生的知识条目

■ 演绎规则的形式：

$$\varphi(\dots) \longrightarrow \psi(\dots)$$

- φ 和 ψ 是任意常元和变元的谓词
- 如果 φ 与Agent的数据库匹配，则可以推导出所有变元实例化的 ψ



■ 一种吸尘Agent的演绎规则

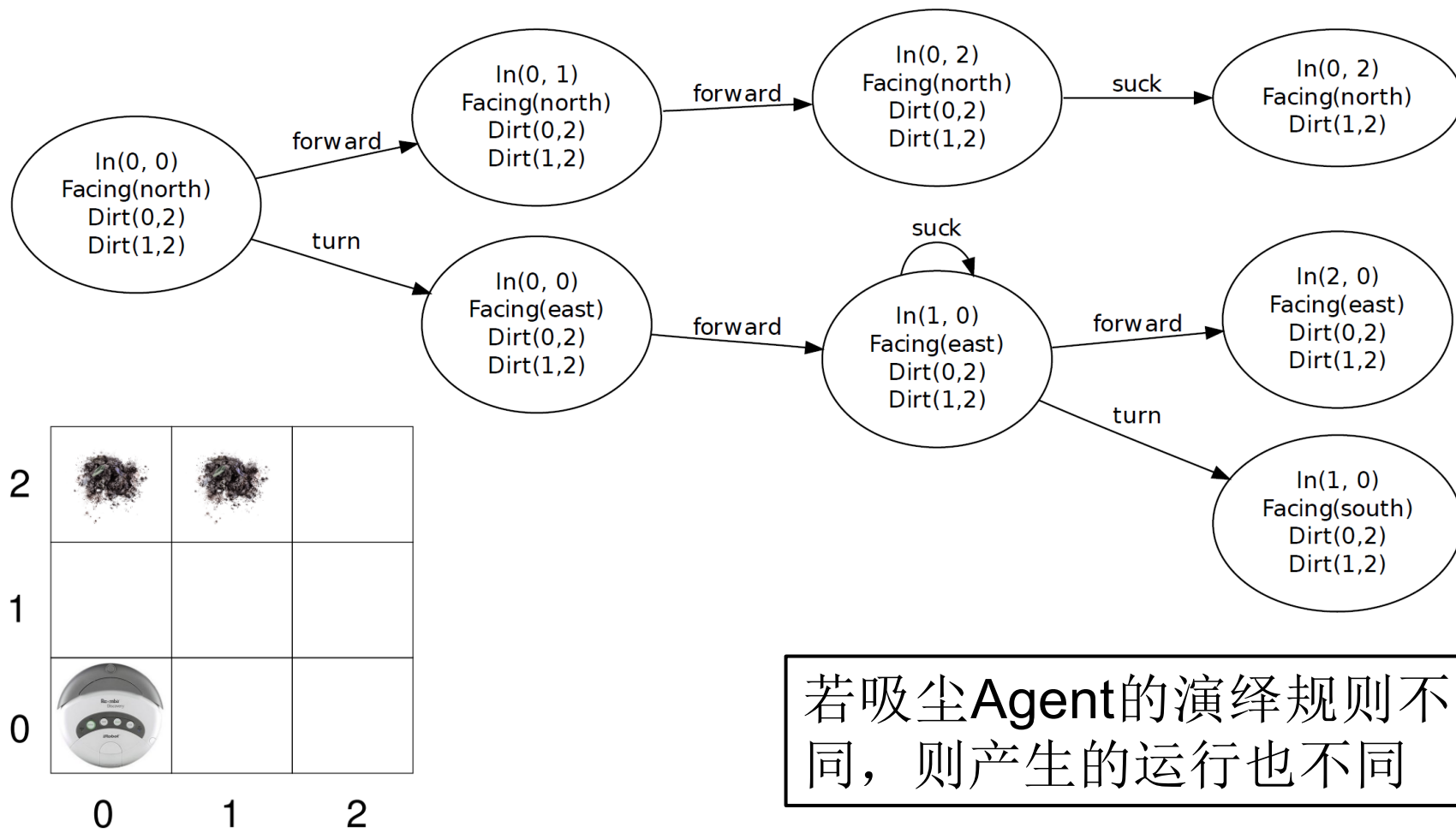
- 清扫规则： $In(x, y) \wedge Dirt(x, y) \longrightarrow Do(suck)$
- 导航规则： $In(0, 0) \wedge Facing(north) \wedge \neg Dirt(0, 0) \longrightarrow Do(forward)$
 $In(0, 1) \wedge Facing(north) \wedge \neg Dirt(0, 1) \longrightarrow Do(forward)$
 $In(0, 2) \wedge Facing(north) \wedge \neg Dirt(0, 2) \longrightarrow Do(turn)$
 $In(0, 2) \wedge Facing(east) \longrightarrow Do(forward)$

只给出了部分演绎规则，请完善



...

■ 使用不同演绎规则的吸尘Agent的内部状态更新示例



若吸尘Agent的演绎规则不同，则产生的运行也不同

- **规则的优先级问题：**当有多条演绎规则可被用于推理时，如何选择？
 - 为不同的演绎规则设置不同的优先级，选择具有更高优先级的演绎规则所产生的动作执行
 - 例：在真空吸尘器世界中，为了实现吸尘的设计目标，把清扫规则的优先级设为最高
- **动作选择的时效性问题：**当环境在Agent演绎推理的过程中发生了变化，会有怎样的后果？
 - 选出的动作可能不再是最优动作
 - 基于逻辑的演绎推理（定理证明）：计算复杂度高
 - 解决办法：不使用严格的逻辑描述语言和完备的演绎规则集合

- **表达能力问题**：当有些环境信息不适合用逻辑来表示时，带来的转换困难的问题
 - 不适合用逻辑来表示的环境信息：图形图像信息、视频信息等
 - 例：如何把视频信息转换为Dirt(0, 1)?
 - 解决方法：使用非逻辑的表示方法
- **易用性问题**：需要用描述性语言刻画Agent的知识和演绎规则，给不熟悉这种语言的软件开发人员带来的不适应问题
 - 描述性语言：如基于逻辑的程序设计语言Prolog

课后作业2-1

- 举3个你所知道的Agent的例子，尽可能准确地定义以下问题：
 - （1） Agent所处的环境（物理环境、软件环境等），环境中的状态，环境的种类（是否完全可观察、是否有不确定性、是否是多Agent、是否是序贯、是否是连续）。
 - （2） Agent可执行的动作库，以及执行这些动作的前提条件。
 - （3） Agent的设计目标，即要实现什么。

课后作业2-2

■ 证明下面的问题：

（1）对于每一个纯反应式Agent，存在一个行为等价的标准Agent。

（2）存在标准Agent，没有与之行为等价的纯反应式Agent。

即：用Agent模型定义的Agent（见第9页）

课后作业2-3

- 有两种方法通过效用函数定义任务：通过效用与状态的关系（ $u: E \rightarrow \mathbb{R}$ ）或者通过效用与运行的关系（ $u: \mathcal{R} \rightarrow \mathbb{R}$ ）。严格来说，第二种效用函数比第一种效用函数有更强的表达能力。给出一个关于运行的效用函数的例子，这个效用函数不能通过与状态有关的效用来定义。

课后作业2-4

- 考虑环境 $Env_1 = \langle E, e_0, \tau \rangle$ ，具体定义如下：

$$E = \{e_0, e_1, e_2, e_3, e_4, e_5, e_6\}$$

$$\tau(e_0 \xrightarrow{\alpha_0}) = \{e_1, e_2, e_3\}$$

$$\tau(e_0 \xrightarrow{\alpha_1}) = \{e_4, e_5, e_6\}$$

这个环境中有两个可能的Agent:

$$Ag_1(e_0) = \alpha_0$$

$$Ag_2(e_0) = \alpha_1$$

不同运行的概率：

$$P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_0} e_3 \mid Ag_1, Env_1) = 0.6$$

$$P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2$$

$$P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.3$$

$$P(e_0 \xrightarrow{\alpha_1} e_6 \mid Ag_2, Env_1) = 0.5$$

假设效用函数 u_1 的定义如下：

$$u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_2) = 7$$

$$u_1(e_0 \xrightarrow{\alpha_0} e_3) = 4$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_4) = 8$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_5) = 2$$

$$u_1(e_0 \xrightarrow{\alpha_1} e_6) = 5$$

给定这些定义，求 Ag_1 和 Ag_2 关于 Env_1 和 u_1 的期望效用，并解释哪个Agent是关于 Env_1 和 u_1 的最优Agent。

课后作业2-5

- 在真空吸尘器世界的例子中，函数new给出了把谓词加入Agent的数据库中的定义，给出函数new的完整定义（如果需要可以使用伪代码）。

课后作业2-6

- 通过给出真空吸尘器世界例子中的默认规则，完成这个例子。你认为解决方案是否直观？是否优美？是否紧凑？

课后作业2-7

- 把真空吸尘器世界的规模扩大到 10×10 个方格的规模。使用上面给出的方法，大致需要多少条规则对这个放大的例子进行编码？对规则进行一般化处理，实现一个更一般的决策机制。