多智能体第一次作业

201300086 史浩男 人工智能学院

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。

1、室内控制温度Agent

- **环境**: 物理环境，状态是不同的温度，温度完全可观察，没有不确定性，可能多Agent（室内空间过大，不同位置都要有Agent），序贯的（调节温度后会影响下一时刻温度），连续的（温度连续变化）
- **动作库**: 测试当前温度，控制空调升温，控制空调降温。前提条件是测温和控温设备正常，供电正常
- **设计目标**: 保持室内不同区域的温度都稳定在预设值或预设区间

2、扫地机器人 Agent

- **环境**: 物理环境，房间内的垃圾和障碍物，状态是房间内各个位置的清洁程度，环境是部分可观察的（视觉盲区），有不确定性的， 单 Agent ，序贯的 (每次扫地都会影响下一次扫地)，连续的 (时间和空间连续变化).
- **动作库**: 前后左右移动，吸尘，检测障碍物，检测垃圾。前提条件：电量充足，没有损坏
- **设计目标**: 尽可能清洁房间，避免损坏自己或其他物体，避免伤害人

3、围棋Agent

- **环境**: 物理环境：棋盘。状态是棋盘上棋子的分布。环境是完全可观察的，没有不确定性的， 单 Agent ，序贯的 (每次落子都会影响棋局)，离散的
- **动作库**: 运算状态得分，落子。前提条件：未达到棋局结束条件
- **设计目标**: 赢得棋局

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）

(1)

一个纯反应式 Agent的行动函数是 $\mathrm{f}:\mathrm{E}\rightarrow\mathrm{Ac}$

构造一个标准 Agent B，它的行动函数是 $\mathrm{g}:\mathcal{R}^{\mathrm{E}}\rightarrow\mathrm{Ac}$，定义为

$$\tau\left(\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_1，\ \mathrm{a}_1，\ \ldots，\ \mathrm{e}_{\mathrm{n}}\right)\right)=\mathrm{f}\left(\mathrm{e}_{\mathrm{n}}\right)$$

则对于任何环境 $Env=\langle E，\ e_0，\ \tau\rangle$, 有$\mathcal{R}(\mathrm{A}，\ \mathrm{Env})=\mathrm{R}(\mathrm{B}，\ \mathrm{Env})$

即两个 Agent 是完全行为等价的

(2)

假设环境 $Env=\langle\{\mathrm{e}_0，\ \mathrm{e}_1\}，\ e_0,\tau\rangle$

我们构造一个标准 Agent A，并假设存在一个纯反应式 Agent B 与 A 对于 Env 行为等价

- 状态转移函数为 $\tau\left(\left(\mathrm{e}_0，\ \mathrm{a}_0\right)\right)=\{\mathrm{e}_0\}，\ \tau\left(\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0，\ \mathrm{a}_1\right)\right)=\{\mathrm{e}_1\}$，其余情况 $\tau(\mathrm{r})=\emptyset$
- 行动函数定义为 $\mathrm{h}\left(\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0\right)\right)=\mathrm{a}_1$，其余情况 $\mathrm{h}(\mathrm{r})=\mathrm{a}_0.$

则 $\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0，\ \mathrm{a}_1，\ \mathrm{e}_1，\ \mathrm{a}_0\right)\in\mathcal{R}(\mathrm{A}，\ \mathrm{Env}),\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0，\ \mathrm{a}_1，\ \mathrm{e}_1，\ \mathrm{a}_0\right)\in\mathcal{R}(\mathrm{B}，\ \mathrm{Env})$

- 输入分别为 $\left(\mathrm{e}_0\right)$ 与 $\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0\right)$时B的行动函数
  $\mathrm{f}\left(\mathrm{e}_0\right)=\mathrm{f}\left(\mathrm{e}_0\right)=\mathrm{a}_0，\ \mathrm{f}\left(\mathrm{e}_0，\ \mathrm{a}_0，\ \mathrm{e}_0\right)=\mathrm{f}\left(\mathrm{e}_0\right)=\mathrm{a}_1$，与纯反应式Agent决策完全基于当前状态的定义产生矛盾.

由反证法，证毕

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。

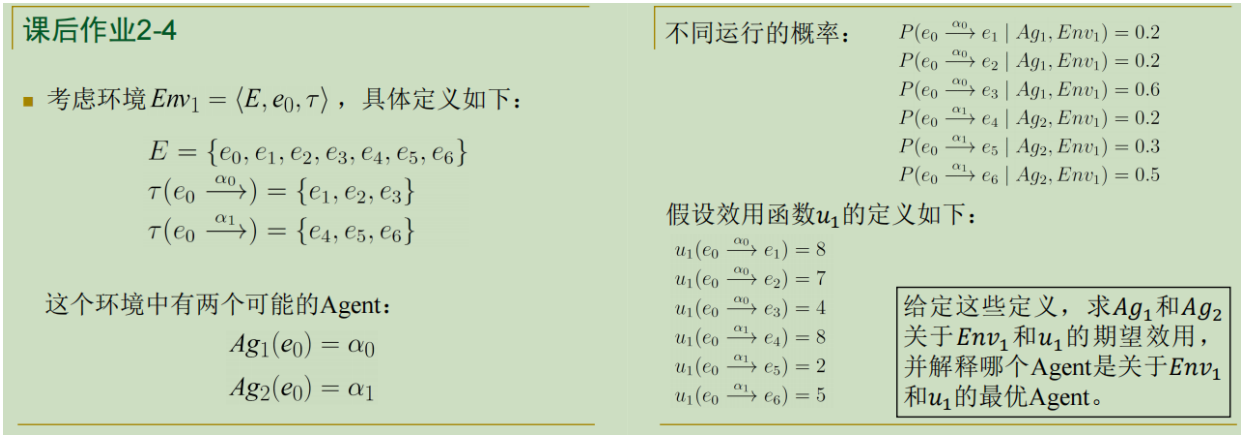
假设环境 $Env=\langle\{\mathrm{e}_0\}，\ e_0,\tau\rangle$

此时关于运行的效用函数可以定义为$\mathrm{u}\left(\mathrm{e}_0\right)=0, \mathrm{u}\left(\mathrm{e}_0, \mathrm{a}_i, \mathrm{e}_0\right)=1$

假设可以通过与状态有关的效用函数定义：

输入分别为 $\left(\mathrm{e}_0\right)$ 与 $\left(\mathrm{e}_0, \mathrm{a}_i, \mathrm{e}_0\right)$时的效用函数 $\mathrm{u}\left(\mathrm{e}_0\right)=\mathrm{u}\left(\mathrm{e}_0, \mathrm{a}_i, \mathrm{e}_0\right)$，产生矛盾

由反证法，证毕



$$\begin{aligned} EU(Ag_1, Env) &= 0.2 \times 8 + 0.2 \times 7 + 0.6 \times 4 = 5.4 \\ EU(Ag_2, Env) &= 0.2 \times 8 + 0.3 \times 2 + 0.5 \times 5 = 4.7 \end{aligned} \tag{1}$$

因此最优Agent是Ag_1

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）。

无需讨论Dirt(x,y)，因为在计算new时，必然清扫过(x,y)位置的垃圾

```
def new(DB,p):
    new_DB_p=[p.In(x,y)]#(x,y)当前位置
    if p.Facing(d) not in DB: #d为当前方向
        new_DB_p.append(Facing(d))
    return new_DB_p
```

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？

增加一个动作：turnLeft，表示逆时针旋转90°

清扫规则不变，见ppt

补全导航规则:

$$
\begin{gathered}
\mathrm{In}(0,0) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(0,0) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(0,1) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(0,1) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(0,2) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(0,2) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(0,2) \wedge \mathrm{F}acing(east) \rightarrow Do(forward) \\
\mathrm{In}(1,2) \wedge \mathrm{F\,acing}(east) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(1,2) \wedge \mathrm{F\,acing}(south) \wedge \neg \mathrm{Dirt}(1,2) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(1,1) \wedge \mathrm{F\,acing}(south) \wedge \neg \mathrm{Dirt}(1,1) \rightarrow \mathrm{Do}(forward) \\
In(1,0) \wedge \mathrm{F}acing(south) \wedge \neg \mathrm{Dirt}(1,0) \rightarrow \mathrm{Do}(turnLeft) \\
\mathrm{In}(1,0) \wedge \mathrm{F}acing(east) \rightarrow Do(forward) \\
\mathrm{In}(2,0) \wedge \mathrm{F\,acing}(east) \rightarrow \mathrm{Do}(turnLeft) \\
\mathrm{In}(2,0) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(2,0) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2,1) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(2,1) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2,2) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(2,2) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(2,2) \wedge \mathrm{F\,acing}(east) \rightarrow \mathrm{Do}(turn)
\end{gathered} \tag{2}
$$

直观、优美、紧凑的定义我没有找到，只能主观处理

我认为解决方案很清洗，因此直观、紧凑，但看上去并不优美

课后作业2-7

把真空吸尘器世界的规模扩大到10x10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了的例子进行编码？对规则进行一般化处理，实现一个更一般的决策机制。

每个非转弯位置需要1条规则，转弯位置需要两条或4条，总共20个转弯位，约共130条规则

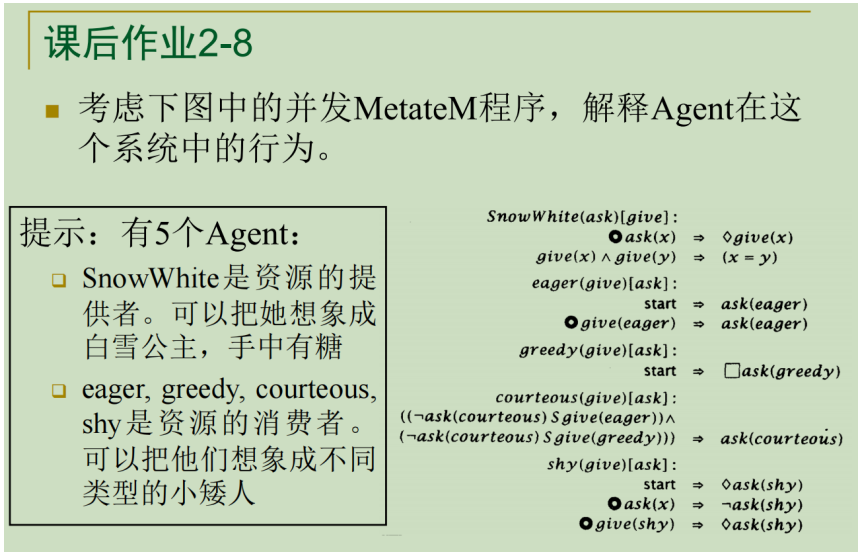
一般决策机制：

清扫规则（优先级最高）

$\mathrm{In(x，y)} \wedge \mathrm{Dirt(x，y)} \rightarrow \mathrm{Do}(clean)$

其他：

$$
\begin{gathered}
\mathrm{In}(2\mathrm{k},\mathrm{m}) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(2\mathrm{k},\mathrm{m}) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(2\mathrm{k},\mathrm{m}) \wedge \mathrm{F\,acing}(east) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2\mathrm{k}+1, n-1) \wedge \mathrm{F}acing(east) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(2\mathrm{k}+1,0) \wedge \mathrm{F\,acing}(south) \wedge \neg \mathrm{Dirt}(2\mathrm{k}+1,0) \rightarrow \mathrm{Do}(\mathrm{turnLeft}) \\
\mathrm{In}(2\mathrm{k}+1,0) \wedge \mathrm{F\,acing}(east) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2\mathrm{k},0) \wedge \mathrm{F}acing(east) \rightarrow \mathrm{Do}(turnLeft) \\
\mathrm{In}(2\mathrm{k},0) \wedge \mathrm{F\,acing}(\mathrm{south}) \wedge \neg \mathrm{Dirt}(2\mathrm{k},0) \rightarrow \mathrm{Do}(\mathrm{turn}) \\
\mathrm{In}(2\mathrm{k},0) \wedge \mathrm{F}acing(west) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2\mathrm{k}+1,0) \wedge \mathrm{F\,acing}(west) \rightarrow \mathrm{Do}(turn) \\
\mathrm{In}(2\mathrm{k}+1, n-1) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(2\mathrm{k}+1, n-1) \rightarrow Do(turnLeft) \\
\mathrm{In}(2\mathrm{k}+1, n-1) \wedge \mathrm{F}acing(west) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(2\mathrm{k},\mathrm{m}) \wedge \mathrm{F}acing(west) \rightarrow \mathrm{Do}(turnLeft) \\
\mathrm{In}(\mathrm{x},\mathrm{y}) \wedge \mathrm{F\,acing}(north) \wedge \neg \mathrm{Dirt}(\mathrm{x},\mathrm{y}) \rightarrow \mathrm{Do}(forward) \\
\mathrm{In}(\mathrm{x},\mathrm{y}) \wedge \mathrm{F\,acing}(\mathrm{south}) \wedge \neg \mathrm{Dirt}(\mathrm{x},\mathrm{y}) \rightarrow \mathrm{Do}(forward)
\end{gathered}
\tag{3}
$$



一共有5个Agent：

SnowWhite

- $SnowWhite(ask)[give]$: 可接收的消息符号为 ask，可发送的消息符号为 give
- $\odot \mathrm{ask}(\mathrm{x}) \Rightarrow \diamond \mathrm{give}(\mathrm{x})$: 如果x昨天要糖了，则将来会把糖给x
- $\mathrm{give}(\mathrm{x}) \wedge \mathrm{give}(\mathrm{y}) \Rightarrow (\mathrm{x}=\mathrm{y})$: 如果把糖给了 x和 y，则他们定是同一个人，即只会把糖给一个人

eager

- $eager(give)[ask]$: 可接收的消息的符号为 give，可发送的消息的符号为 ask
- $start \Rightarrow \mathrm{ask}(eager)$: 在系统启动后，他会立即发送一条要求糖的消息.
- $\odot \mathrm{give}(eager) \Rightarrow \mathrm{ask}(eager)$: 如果收到了糖，就会再次发送要求糖的消息

greedy

- $greedy(give)[ask]$: 可接收的消息的符号为 give，可发送的消息的符号为 ask
- $start \Rightarrow \Box\,\mathrm{ask}(greedy)$: 在系统启动后，他会不断发送要求糖的消息

courteous

- $courteous(give)[ask]$: 可接收的消息的符号为 give，可发送的消息的符号为 ask
- $((\neg\,\mathrm{ask}(courteous)Sgive(eager)) \wedge (\neg\,\mathrm{ask}(courteous)Sgive(greedy)) \Rightarrow \mathrm{ask}(courteous)$: 如果 eager 和 greedy 没有要求糖，他才会发送一条要求糖的消息.

shy

- $\mathrm{shy}(give)[ask]$: 可接收的消息的符号为 give，可发送的消息的符号为 ask
- $start \Rightarrow \diamond\,\mathrm{ask(shy)}$: 在系统启动后，他将会发送一条要求糖的消息.
- $\odot\,\mathrm{ask(x)} \Rightarrow \neg\,\mathrm{ask(shy)}$: 如果有其他消费者要求糖，他就不会要求糖.
- $\odot\,\mathrm{give(shy)} \Rightarrow \diamond\,\mathrm{ask(shy)}$: 如果之前得到了糖，则将来还会继续要求糖.

> **课后作业2-9**
>
> - 回忆在2.3节中讨论的真空吸尘器的例子，使用STRIPS的符号形式化表示这个Agent可提供的操作。

1. **Forward:**

$$\begin{array}{cc} & \mathrm{Forward}(x,y,d) \\ pre & \mathrm{In(x,y)} \wedge \mathrm{F\,acing(d)} \\ del & \mathrm{In(x,y)} \\ add & \mathrm{In}\left(\mathrm{x}',\mathrm{y}'\right) \end{array} \tag{4}$$

2. **Clean:**

$$\begin{array}{cc} & \mathrm{Clean(x,y)} \\ pre & \mathrm{In(x,y)} \wedge \mathrm{Dirt(x,y)} \\ del & \mathrm{Dirt(x,y)} \\ add & \emptyset \end{array} \tag{5}$$

3. **Turn:**

$$\begin{array}{cc} & \mathrm{Turn(d)} \\ pre & \mathrm{F\,acing(d)} \\ del & Facing(d) \\ add & Facing\left(d'\right) \end{array} \tag{6}$$

课后作业2-10

考虑用如下谓词描述的积木世界：

谓词	含义
On(x, y)	物体x在物体y之上
OnTable(x)	物体x在桌面上
Clear(x)	物体x上没有东西
Holding(x)	机械臂拿着物体x
ArmEmpty	机械臂为空

Agent关于积木A、B、C的初始信念B_0和意图i为：

$Beliefs\ B_0$	$Intention\ i$
$Clear(B)$	$Clear(A)$
$Clear(C)$	$Clear(B)$
$On(C,A)$	$On(B,C)$
$OnTable(A)$	$OnTable(A)$
$OnTable(B)$	$OnTable(C)$
$ArmEmpty$	$ArmEmpty$

Agent有一个动作集合:

$$Ac = \{Stack, Unstack, Pickup, PutDown\}$$

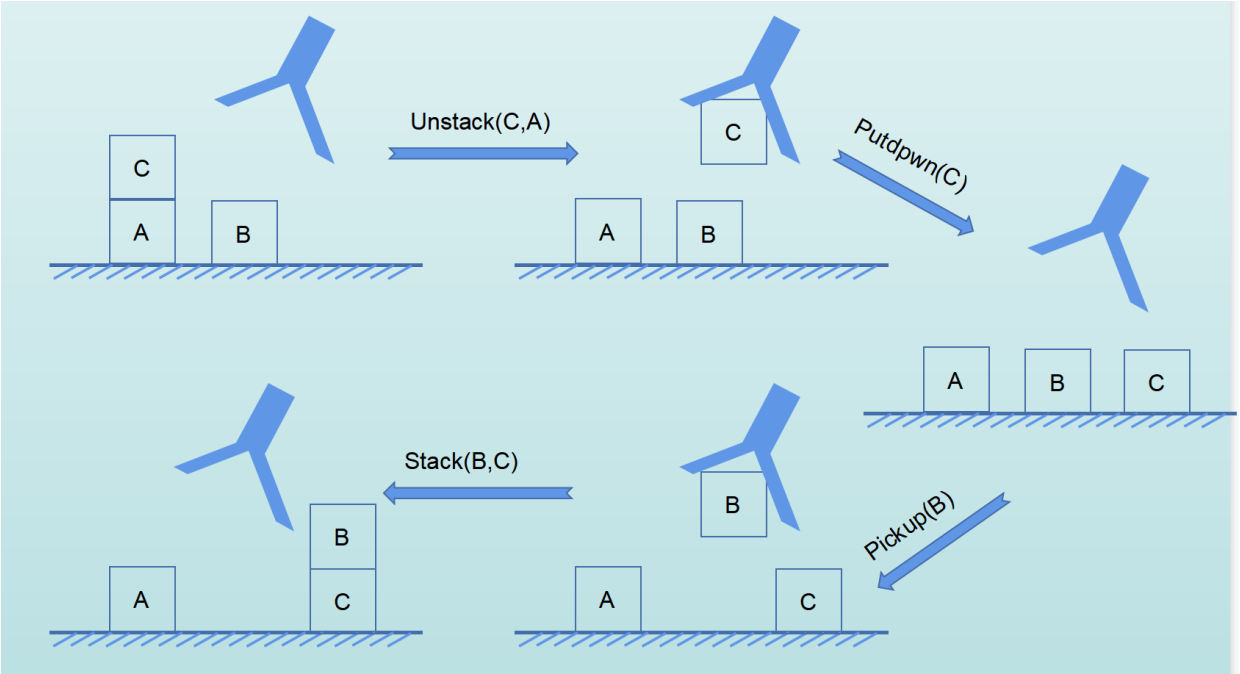
$Stack(x,y)$	
pre	$Clear(y)\ \&\ Holding(x)$
del	$Clear(y)\ \&\ Holding(x)$
add	$ArmEmpty\ \&\ On(x,y)$

$UnStack(x,y)$	
pre	$On(x,y)\ \&\ Clear(x)\ \&\ ArmEmpty$
del	$On(x,y)\ \&\ ArmEmpty$
add	$Holding(x)\ \&\ Clear(y)$

$Pickup(x)$	
pre	$Clear(x)\ \&\ OnTable(x)\ \&\ ArmEmpty$
del	$OnTable(x)\ \&\ ArmEmpty$
add	$Holding(x)$

$PutDown(x)$	
pre	$Holding(x)$
del	$Holding(x)$
add	$OnTable(x)\ \&\ ArmEmpty\ \&\ Clear(x)$

给定初始信念B_0和意图i，计算一个规划π。画出该规划开始时的环境，以及每次执行了动作后的环境。



课后作业2-11

- 以下的伪代码为实用推理（BDI）Agent定义了一个控制回路：

```
1.
2.  B := B0;
3.  I := I0;
4.  while true do
5.      get next percept ρ;
6.      B := brf(B, ρ);
7.      D := options(B, I);
8.      I := filter(B, D, I);
9.      π := plan(B, I);
10.     while not (empty(π) or succeeded(I, B) or impossible(I, B)) do
11.         α := hd(π);
12.         execute(α);
13.         π := tail(π);
14.         get next percept ρ;
15.         B := brf(B, ρ);
16.         if reconsider(I, B) then
17.             D := options(B, I);
18.             I := filter(B, D, I);
19.         end-if
20.         if not sound(π, I, B) then
21.             π := plan(B, I)
22.         end-if
23.     end-while
24. end-while
```

参照此伪代码，解释以下组件的用途/作用：

a) 变量B、D和I
b) 感知ρ
c) brf(…)函数
d) options(…)函数
e) filter(…)函数
f) plan(…)函数
g) sound(…)函数
h) succeeded(…)函数和impossible(…)函数
i) reconsider(…)函数——在回答这部分问题的时候，你应该要明确说明此函数应具有的属性，以及可以假定其正常运行的情况。

(a) 变量 B，D 和 I:

- B代表 belief，即 Agent 所持有的信念集合，包含了 Agent 当前对世界的认知，代表环境
- D代表 desire，即 Agent 的愿望集合，表示 Agent 希望实现的状态或行为
- I代表 intention，即 Agent 当前的意图集合，表示 Agent 在当前时刻要实现的目标

(b) 感知 ρ:

感知是 Agent 从外部环境中收集信息的方式，获取当前时刻 Agent 所处的环境状态

(c) $\mathrm{brf}(\ldots)$函数:

$\mathrm{B} := \mathrm{brf}(\mathrm{B}, \rho)$: 信念修正函数，用于更新 Agent 的信念集合

(d) $\mathrm{options}(\ldots)$函数:

$D := \mathrm{options}(\mathrm{B}, \mathrm{I})$ ：选项生成函数，产生可能的选项或愿望集合 D

(e) $filter(\ldots)$ 函数:

$I := \mathrm{filter}(B, D, I)$: 过滤函数，用于从愿望集合 D 中过滤掉 Agent 在当前时刻难以完成的愿望，进而生成当前的意图集合 I，也即从竞争的选项中做出 "最佳" 的选择

(f) $\mathrm{plan}(\ldots)$ 函数:

$\pi := \mathrm{plan}(\mathrm{B}，\mathrm{I})$规划算法，用于根据 Agent 的信念集合 B 和当前的意图集合 I生成一个规划 π，以实现当前的意图 I

(g)$sound(\ldots)$ 函数:

$\mathrm{sound}(\pi，I，B)$ 函数是一个布尔函数，代表在给定 B 时，π 是实现 I 的正确的规划。用于判断 Agent 所制定的规划 π 是否合理且可行，以确保 Agent 能够在其执行规划的过程中不会违背其信念集合 B 和意图集合 I

(h) $succeeded(\ldots)$ 函数和 $impossible(\ldots)$ 函数:

$succeeded(\mathrm{I}，\mathrm{B})$函数和$impossible(I，B)$函数是用于判断当前的意图集合I是否是已经成功或者是不可能成功
$succeeded(I，B)$函数判断 Agent 在执行规划 π 后是否成功实现了其意图集合 I，如果成功则返回 True，否则返回 False.
$impossible(I，B)$ 函数判断 Agent 是否无法通过任何规划实现其意图集合 I，如果 无法实现则返回 True，否则返回 False.

(i) $reconsider(\ldots)$ 函数:

$reconsider(I，B)$函数是一个布尔函数，用于在 Agent 执行规划过程中，根据环境 变化改变信念集合 B 的情况下，思考当前的信念集合 B 和意图集合I 间的关系，以判断是否需要调整 Agent 当前的愿望集合 D 和意图集合 I. 在其认为需要重新考虑时返回 True，不需要重新考虑时返回 False.

通过引入一个布尔函数 reconsider 决定是否重新考虑意图，就能很好地权衡重复考虑意图与否之间的平衡问题，解决专一承诺 Agent和坦率承诺 Agent的两难局面

> 课后作业2-12
>
> - 用Brooks的归类式结构的方法求解第2.3节描述的真空吸尘器的例子。

优先级依次从高到低:

1. if 当前网格位置有垃圾，then 打扫垃圾
2. if 当前位置在起点或者终点，then 顺时针转动 180° 掉头
3. if 当前位置位于上边界且面向边界，then 顺时针转动 90°，前进一格，再顺时针转动 90°
4. if 当前位置位于下边界且面向边界，then 逆时针转动 90°，前进一格，再逆时针转动 90°
5. if True，then 向前移动

课后作业2-13

- 用第2.3节描述的基于逻辑的方法求解探测火星的例子。

描述 Agent 内部状态的领域谓词：

- $\mathrm{In(x，y)}$：$Agent$位于$\mathrm{(x，y)}$位置;
- $\mathrm{Sample(x，y)}$：$\mathrm{(x，y)}$位置存在样本
- $Obstacle(x，y)$：$(x，y)$位置存在障碍物;
- $\mathrm{Base(x，y)}$：$\mathrm{(x，y)}$位置存在基地;
- $\mathrm{Crumb}(x，y)$：$(x，y)$位置存在碎屑;
- $Carry$：当前是否携带样本.

动作集合:

Turn: 改变方向

MoveRandomly: 随机移动

Drop: 放下样本

PickUp: 捡起样本;

GradientUp: 沿梯度上升方向行驶

GradientDown: 沿梯度下降方向行驶

DropCrumbs: 扔下一些碎屑

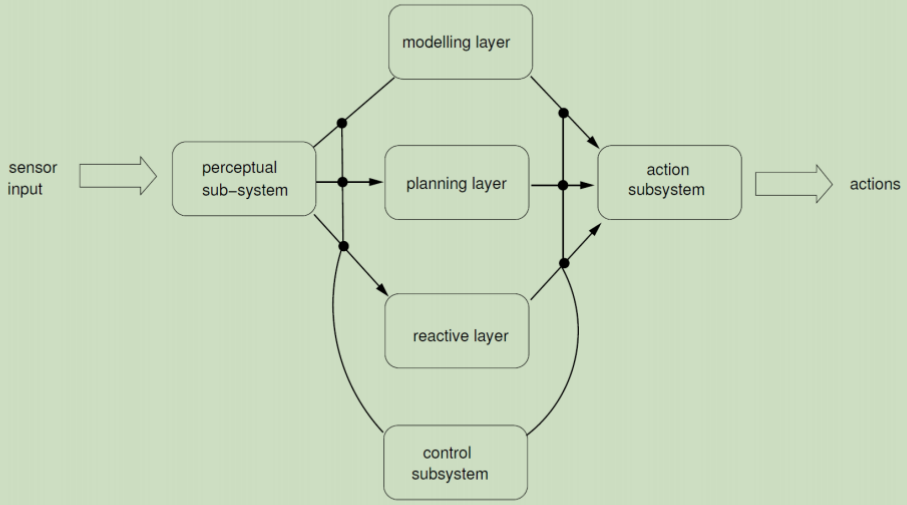
PickUpCrumb: 采集一个碎屑

演绎规则：

$$
\begin{gathered}
\mathrm{In}(x，y) \wedge \mathrm{Obstacle}(x，y) \rightarrow \mathrm{Do}(Turn) \\
\mathrm{In}(x，y) \wedge Carry \wedge \mathrm{Base(x，y)} \rightarrow Do(Drop) \\
\mathrm{In(x，y)} \wedge \mathrm{Carry} \wedge \neg \mathrm{Base(x，y)} \rightarrow \mathrm{Do}(DropCrumbs) \wedge Do(GradientUp) \\
\mathrm{In(x，y)} \wedge \mathrm{Sample(x，y)} \rightarrow \mathrm{Do(P\,ick\,Up)} \\
\mathrm{In}(x，y) \wedge \mathrm{Crumb(x，y)} \rightarrow \mathrm{Do(PickUpCrumb)} \wedge Do(GradientDown) \\
True \rightarrow Do(MoveRandomly)
\end{gathered}
\tag{7}
$$

课后作业2-14

- 下列示意图阐明了Touring机体系结构的关键子系统：



描述这个系统结构的各个部分，包括解释其中三个决策层如何实现产生反应式行为和预动行为的目标。

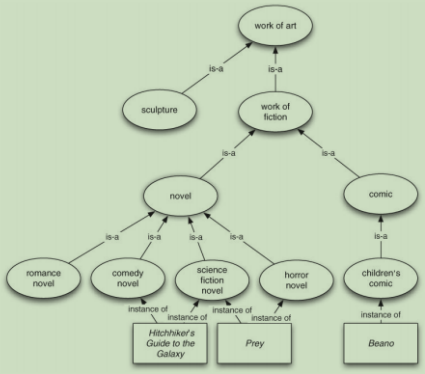
- 传感器输入 (Sensor input)：接收外部环境的信号， 并将其转化为数字信号
- 感知子系统 (Perceptual sub-system)：对传感器输入的数字信号进行处理和分析，提高对外部环境的理解和感知能力
- 模型层 (Modelling layer)：代表世界上各种各样的实体 (包括 Agent 本身和其他 Agent)， 可以预言 Agent 之间的冲突， 并产生需要完成的新目标来解决这些冲突。可以将新目标下传到规划层， 利用规划库来决定如何实现它们
- 规划层 (Planning layer)：完成 Agent 的预动行为，为了实现目标会尝试在规划库中找到一个合适的规划，并据此选择出要执行的动作
- 反应层 (Reactive layer)：反应层用情景-动作规则集实现，能对环境中发生的改变提供迅速的反应
- 控制子系统 (Control subsystem)：一个动作仲裁部件，按照控制规则集来实现，对三个决策层的输入和输出数据进行分析和控制， 可以使用各种控制算法和调节策略，实现对系统行为的优化和调整
- 执行子系统 (Action subsystem)：根据三个决策层得出来的结果，执行相应的动作和操作
- 行动 (Actions)：输出部分，代表了机器对外部环境所做出的响应

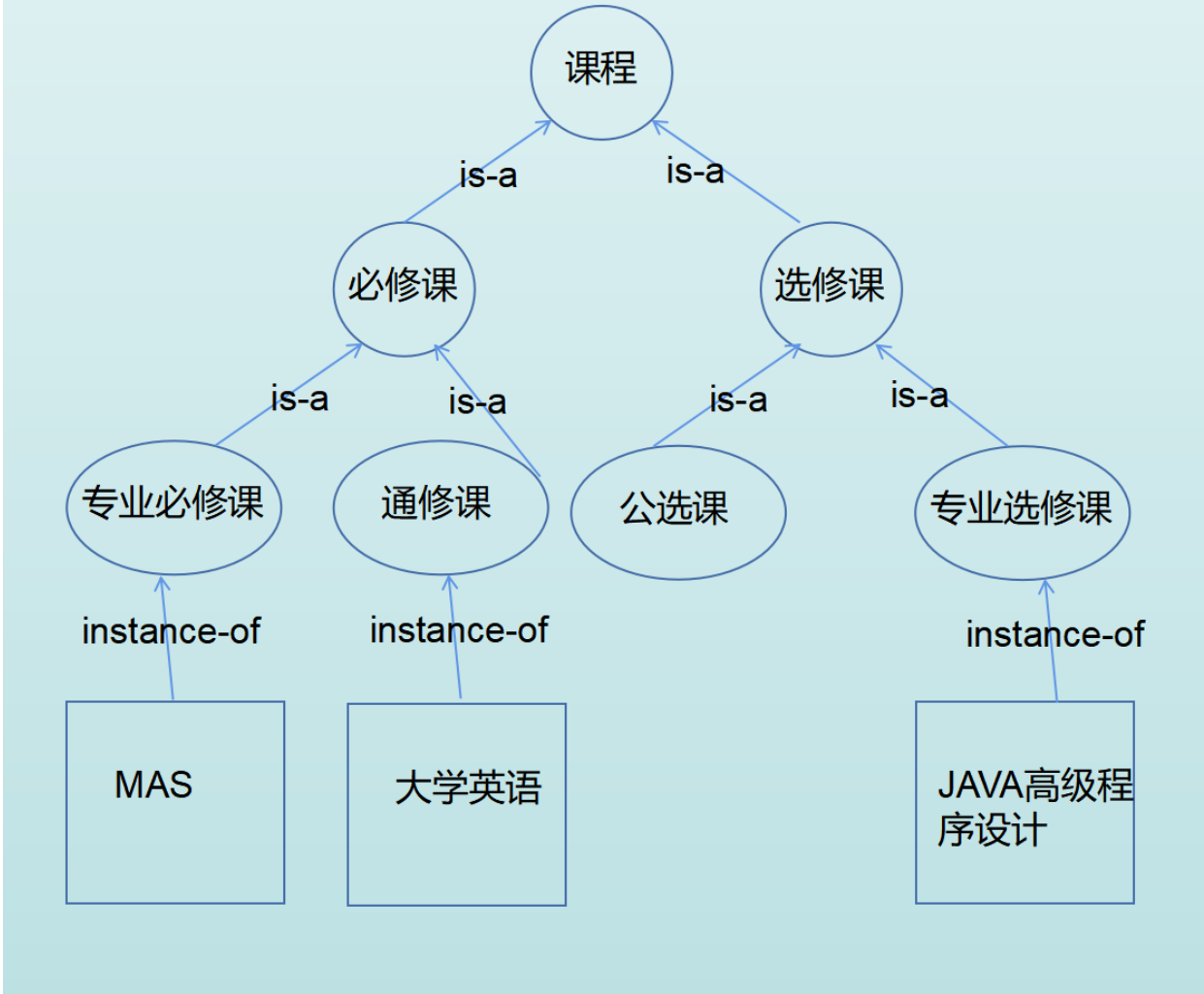
反应式行为的实现：根据来自感知子系统的输入，实现快速响应和高效执行，以应对突发的环境变化和紧急任务，实现即时的基本的反应动作。

预动行为的实现：模型层将新目标下传到规划层，利用规划库来决定如何实现它们。规划层考虑机器人的长期目标，环境的变化和任务的需求，以实现预动行为。

课后作业3-1

- 使用课上所学的本体设计方法构建一个本体。使用类似课件中的**框图表示类别层次和实例关系，并描述相关的属性和至少2个可能的属性约束**（可用自然语言表达）。应满足的需求有：
  - 课程，可以分为必修课与选修课，其中必修课分为专业必修课和通修课，选修课分为专业选修课和公选课。课程的属性包括课程编号、授课老师、选课人数等等。
  - 《多智能体系统》是专业必修课
  - 《大学英语》是通修课
  - 《Java高级程序设计》是专业选修课
- 内容可根据上述需求进行补充。属性约束请自行设计，约束符合常理即可。





属性：

- 课程编号，开设时间，上课教室，开设院系，选课人数：外在属性
- 授课教师：关系

属性约束：

- 课程编号，开设时间，上课教室，开设院系，选课人数：唯一，且格式规范，势约束，类别约束
- 授课教师：有范围约束，需要是老师类中的成员

> ## 课后作业3-2
>
> - 利用课后作业3-1中设计的本体，使用KQML语言描述一段对话：
>   - A向B查询（evaluate）《多智能体系统》课程的选课人数情况。
>   - B告知A《多智能体系统》选课人数为30人。
> - 需要描述清楚每条消息的语用词和参数，你可以使用自然语言表达消息内容。

1、查询

```
(:evaluate
    :sender A
    :receiver B
    :language KIF
    :ontology 课程本体
    :reply-with q1
    :content (val (选课人数 多智能体系统)))
```

2、回复

```
(:reply
    :sender B
    :receiver A
    :language KIF
    :ontology 课程本体
    :in-reply-to q1
    :content (= (选课人数 多智能体系统) (scalar 30)))
```

课后作业3-3

- 利用课后作业3-1中设计的本体，使用KQML语言描述一段对话：
  - A向B查询《机器学习导论》课程的所有信息。
  - B告知A《机器学习导论》的课程编号为30000150。
  - B告知A《机器学习导论》的授课老师为周志华老师。
  - B告知A《机器学习导论》选课人数为300人。
  - B告知A当前通信流结束。

- 需要描述清楚每条消息的语用词和参数，你可以使用自然语言表达消息内容。

```
(stream-about
    :sender A:receiver B
    :language KIF:ontology 课程本体
    :reply-with q1
    :content 机器学习导论)
(tell
    :sender B:receiver A
    :in-reply-to q1
    :content (= (课程编号 机器学习导论) (scalar 30000150)))
(tell
    :sender B:receiver A
    :in-reply-to q1
    :content (= (授课老师 机器学习导论) 周志华))
(tell
    :sender B:receiver A
    :in-reply-to q1
    :content (= (选课人数 机器学习导论) (scalar 300)))
(eos
    :sender B:receiver A
    :in-reply-to q1)
```

课后作业3-4

- 使用自然语言，描述下面两段KQML消息的含义

```
(a) (ask-if
      :sender      A
      :receiver    B
      :language    OWL
      :ontology    pizza
      :reply-with  q1
      :content ( (margherita isa Pizza)
                 (margherita hasTopping mozzarella) )
    )

(b) (tell
      :sender      A
      :receiver    B
      :language    OWL
      :ontology    pizza
      :reply-with  q1
      :content (not (hawaiian isa ItalianPizza))
    )
```

(a)

发送者 A 向接收者 B 发送了一个 ask-one 类型的消息, 消息内容是用 OWL 语言来描述的, 其本体为 pizza。该消息的标识为 q1, 内容是询问 margherita 是否是一种 Pizza，以及 margherita 的顶部是否有 mozzarella

(b)

发送者 A 向接收者 B 发送了一个 tell 类型的消息, 消息内容是用 OWL 语 言来描述的, 其本体为 pizza。该消息的标识为 q1，内容意思为 hawaiian 不是一种 ItalianPizza,

课后作业3-5

- 使用KQML语言，描述如何实现合同网协议。

合同网包括识别、通知、竞标、授予、实现五个阶段，是通过任务共享实现有效合作的高级协议

识别阶段

要求所有参与者提供身份证明

```
(send-all
     (assert
     :content (list :provide-identity)
         :receiver :all
         :language :kqml))
```

通知阶段

所有参与者已经提供了身份证明。

使用tell语用词告知所有参与者一个竞标任务

```
(send-all
    ( assert
        :content (list :all-participants-identified)
        :language :kqml))

(send-all
    ( tell
        :content (list :bid-task "task-id")
        :receiver :all
        :language :kqml))
```

竞标阶段

参与者可以使用request语用词请求竞标任务的详细信息

使用tell语用词告知所有参与者一个合同的详细信息

参与者可以使用ask-all语用词询问其他参与者是否已经提交竞标

```
(send
  ( request
    :content (list :request-task-info "task-id")
    :receiver :task-manager
    :language :kqml))

(receive
  ((:tell
     :content (list :provide-contract-info "contract-id")
     :sender :task-manager
     :language :kqml))
   (send-all
     ( tell
       :content (list :provide-contract-info "contract-id")
       :receiver :all
       :language :kqml)))

(send-all
  ( ask-all
    :content (list :bid-submitted-ask "task-id")
    :receiver :all
    :language :kqml))
```

授予阶段

竞标结束后，使用tell语用词告知所有参与者竞标结果

如果有参与者中标，则使用assert语用词陈述一个事实，即“竞标任务已授予给中标者”，并告知中标者合同的详细信息

```
(receive
  ((:tell
     :content (list :bid-result "task-id" "winner-id")
```

```
     :language :kqml))
   (send-all
     ( tell
       :content (list :bid-result "task-id" "winner-id")
       :receiver :all
       :language :kqml)))

(if (equal "winner-id" "my-identity")
  (progn
    (send
      ( assert
        :content (list :contract-awarded "contract-id")
        :language :kqml))
    (send
      ( tell
        :content (list :provide-contract-info "contract-id")
        :receiver "winner-id"
        :language :kqml))))
```

实现阶段

当中标者同意合同后，使用tell语用词告知所有参与者该合同已经实现

```
(receive
  ((:tell
     :content (list :contract-implemented "contract-id")
     :language :kqml))
   (send-all
     ( tell
       :content (list :contract-implemented "contract-id")
       :receiver :all
       :language :kqml))))
```