南京大学
人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE, NANJING UNIVERSITY

第二部分： 智能自治Agent

章宗长

2023年3月14日

内容安排

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

面向Agent的程序设计

- 1990年，Yoav Shoham基于计算的社会性观点，提出了一个新的程序设计风格：面向Agent的程序设计

- 核心思想：用意识属性直接对Agent进行程序设计
  - 意识属性：如信念、愿望、意图等
  - 采用与人类一样的抽象机制表示复杂系统的特性

- Agent0：第一个实现面向Agent程序设计风格的语言

Agent0

- Agent0中的每个Agent有4部分：
  - 一组能力集合（Agent可以做的事情）
  - 初始信念集合
  - 初始承诺集合（Agent将要做的事情）
  - 承诺规则集合

决定Agent如何行动的关键部分

- 一条承诺规则包含：
  - 一个消息条件
  - 一个思维状态条件（mental condition）
  - 一个动作

- 当接收到一条消息，激活一条承诺规则的前提：
  - 该规则的消息条件与Agent接收到的消息匹配
  - 该规则的思维状态条件与Agent的信念匹配
- 如果一条承诺规则被激活，则Agent承诺要做的动作
- Agent的动作
  - 私有动作
    - 内部执行的子进程
  - 通信动作
    - 发送消息
- 消息
  - 请求（request）
    - 执行一个动作
  - 取消请求（non-request）
    - 阻止执行一个动作
  - 通知（inform）
    - 传送消息

Agent0的承诺规则的一个例子

- 如果从agent收到一条要求在时间time做动作action的消息，并且相信：
  - agent此时是朋友
  - 可以做动作action
  - 在时间time，没有对其他动作的承诺

  则承诺在时间time做动作action

```
COMMIT(
   ( agent, REQUEST, DO(time, action)
   ), ;;; msg condition
   ( B,
     [now, Friend agent] AND
     CAN(self, action) AND
     NOT [time, CMT(self, anyaction)]
   ), ;;; mental condition
   self,
   DO(time, action)
)
```

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**

并发MetateM & 时序逻辑

- 并发MetateM
  - 由Michael Fisher开发
  - 基于时序逻辑的多Agent编程语言
- 时序逻辑
  - 在经典逻辑的基础上增加了表示事件的时间顺序模态的连接词
  - 命题时序逻辑=命题逻辑+事件的时间顺序模态
  - 一阶时序逻辑=一阶逻辑+事件的时间顺序模态

MetateM Agent

- 并发MetateM系统中的Agent是并发执行的实体，可以通过消息广播互相通信
- 并发MetateM Agent有两个主要部件：
  - 接口：定义Agent如何与环境（即其他Agent）进行交互
  - 计算引擎：定义Agent如何采取行动
- 一个Agent的接口由3个部分组成：
  - Agent的标识（即Agent的名字）
  - 环境命题集：用来定义Agent可以接收的消息的符号集合
  - 组件命题集：用来定义Agent可以发送的消息的符号集合

- 例如，一个“堆栈”Agent的接口定义可以是

$$stack(pop, push)[popped, full]$$

Agent的标识　　环境命题　　组件命题

- 计算引擎
  - 直接执行Agent的说明，该说明用一组程序规则给出
  - 每条程序规则对应于一个时序逻辑公式
  - 每个时序逻辑公式描述了Agent的功能和行为
  - 程序规则集合的定义是用并发MetateM编写Agent程序的关键和核心

程序规则

- 形式：

$$\text{与过去有关的前件} \Longrightarrow \text{与现在和将来有关的后件}$$

- 时序逻辑公式，一些例子：

$$\Box \text{important(agents)}$$

现在和将来agents都是重要的

$$\Diamond \text{important(ConcurrentMetateM)}$$

将来某个时刻，并发MetateM是重要的

$\blacklozenge\, \text{important}(\text{Prolog})$

在过去某个时刻，Prolog是重要的

$(\neg \text{friends}(\text{us}))\, \mathcal{U}\, \text{apologise}(\text{you})$

直到你道歉以前，我们都不是朋友

$\bigcirc \text{apologise}(\text{you})$

明天（下一个状态），你道歉

$\boxdot\, \text{apologise}(\text{you}) \Rightarrow \bigcirc \text{friends}(\text{us})$

如果你昨天道歉了，那么我们明天将是朋友

$\text{friends}(\text{us})\, \mathcal{S}\, \text{apologise}(\text{you})$

从你道歉以来，我们一直是朋友

并发MetateM规则的时序连接符

操作符	含义
$\bigcirc\varphi$	φ“明天”为真
$\bullet\varphi$	φ“昨天”为真
$\Diamond\varphi$	未来某个时刻有φ
$\Box\varphi$	未来总是有φ
$\blacklozenge\varphi$	过去某个时刻有φ
$\blacksquare\varphi$	过去总是有φ
$\varphi\,\mathcal{U}\,\psi$	直到ψ以前φ为真
$\varphi\,\mathcal{S}\,\psi$	从ψ以来φ一直为真
$\varphi\,\mathcal{W}\,\psi$	除非ψ为真φ才为真
$\varphi\,\mathcal{Z}\,\psi$	除非ψ为真φ才为真

小结

- 作为定理证明的Agent
  - 内部状态（知识库），感知函数，状态转移函数，演绎规则
  - 动作选择函数是一个基于其知识的定理证明过程
  - 例子：真空吸尘器世界
  - 问题：动作选择的时效性、表达能力、易用性
- 面向Agent的程序设计
  - 核心思想：用意识属性直接对Agent进行程序设计
  - Agent0：第一个实现面向Agent程序设计风格的语言
- 并发MetateM
  - 基于时序逻辑的多Agent编程语言
  - 可以用于实现演绎推理Agent

内容安排

实用推理Agent

- **实用推理=慎思过程+目标手段推理**
- 目标手段推理
- 实现一个实用推理Agent
- 过程推理系统

什么是实用推理(practical reasoning)?

- 实用推理直接通过推理得到动作，是弄清楚做什么动作的过程：

> 实用推理是权衡不同观点间的矛盾，这些不同的观点来自Agent的愿望/评价/关心的问题以及Agent相信的事情。 ——Michael E. Bratman

- 将实用推理与理论推理区分：
  - 理论推理直接导致信念
  - 实用推理直接导致动作

实用推理的组成

- 人类的实用推理由两个过程组成：
  - **慎思过程**（deliberation）
    - 决定要实现的状态
    - 慎思过程的结果是**意图**（intentions）
  - **目标手段推理**（means-ends reasoning）
    - 决定如何实现这些状态
    - 手段-目的推理的结果是**规划**（plans）
- 例子：大学毕业，面临重要选择
  - **慎思过程**：决定把哪种职业作为目标的过程
  - **目标手段推理**：决定如何实现这一状态的过程

意图

- 意图可以刻画动作：

  我可能故意地把一个人推到火车下面，并且推他的意图是要杀了他。

- 意图可以刻画思维状态：

  今天早上我有意图要在下午把某个人推到火车下面。

- **在实用推理中，用意图刻画思维状态**
  - 未来的意图，即Agent有意图实现将来的某一状态

实用推理中的意图

- 意图在实用推理中起着重要作用：

1. 意图驱动目标手段推理

如果Agent形成了一个意图，它会投入资源来决定如何实现这一意图。

2. 意图约束未来慎思

Agent不会接受与当前意图不一致的选择。

3. 意图的持续性

没有足够的理由一般不会放弃一个意图。

4. 意图与信念密切相关

如果Agent形成了一个意图，那么它相信在合适的条件下，能实现这个意图。

如果Agent有意图实现某个状态，同时相信不能实现它，那么它是不理性的。

5. 不必想要意图的所有预期副作用

如果Agent相信□ψ 并且想要实现ϕ，那么它不一定也想要实现ψ。

想要看牙医，不意味着想要痛苦

6. 意图有比愿望（desire）更强的预动属性

我今天下午想打篮球的愿望会潜在地影响今天下午的活动。在确定做什么之前，它必然会和其他有关愿望发生竞争……

但是，一旦我的意图是下午打篮球，事情就确定了：一般不需要再继续从多个方面考虑。

等到下午的时候，一般只要继续执行这个意图就可以了。

（Bratman, 1990）

实用推理Agent

- 实用推理=慎思过程+目标手段推理
- **目标手段推理**
- 实现一个实用推理Agent
- 过程推理系统

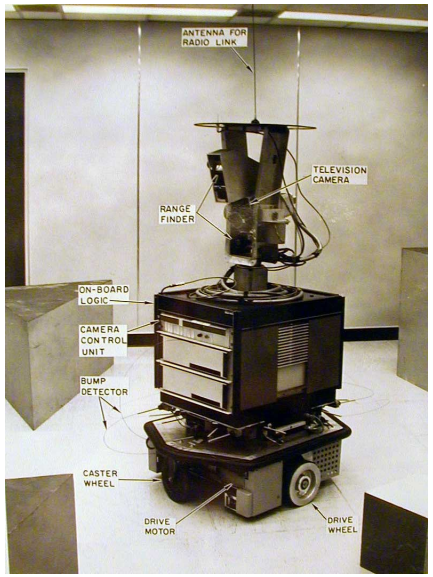
目标手段推理/规划（Planning）

- 规划是行动方案的设计，它将实现一些期望的目标
  - 规划器是接受下列输入表示的系统：
    - 目标、意图或者任务
    - 当前的环境状态——Agent的信念
    - Agent可以采取的动作

  - 作为输出，规划算法产生一个规划（plan）
- 本质上，这就是自动编程（automatic programming）

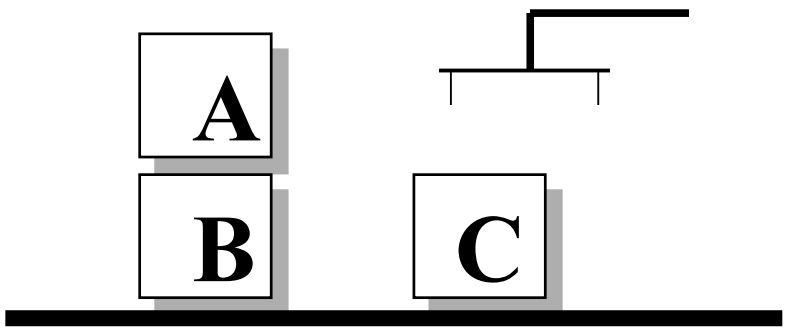
- 不需要直接告诉系统如何去做！
  - 让它自己想办法实现目标！

- STRIPS规划器
  - 第一个规划器：Stanford Research Institute Problem Solver
  - 由Richard Fikes和Nils Nilsson在1971年开发，用在机器人Shakey上
  - 两个基本部件
    - 世界模型：用一阶逻辑的集合表示
    - 动作模式：描述了规划Agent可以执行的所有动作的前提条件和执行效果



机器人Shakey

积木世界



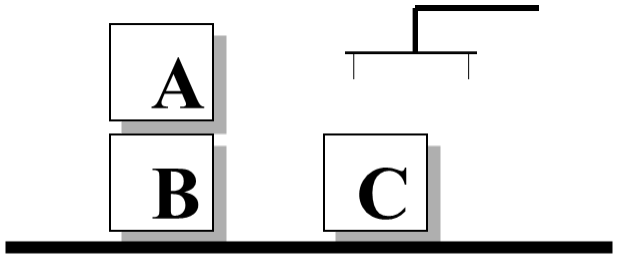
- 以积木世界为例介绍规划技术
  - 3个相同尺寸的积木（A，B和C），一个机械臂和一个桌面
  - 目的：为机械臂生成一个堆放积木的规划

- 描述积木世界的谓词：

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

- 使用封闭世界假设：
  - 未声明的任何内容为假

- 右图积木世界的谓词表示：
  - {Clear(A), On(A, B), OnTable(B),
    OnTable(C), Clear(C), ArmEmpty}



- 把目标也表示为一阶逻辑公式的集合：
  - {OnTable(A), OnTable(B), OnTable(C), ArmEmpty}

把所有的积木都放在桌面上

积木世界中的动作

- 使用STRIPS的形式来表示动作
- 每个动作有一个：
  - 名字：可能带有参数
  - 前提条件表：一些事实的表，动作执行前必须为真
  - 删除表：一些事实的表，动作执行后不再为真
  - 增加表：一些通过执行动作变成真的事实表
- 每个动作都可能包含变量

积木世界中的动作

Stack(x, y)

pre *Clear(y) ^ Holding(x)*

del *Clear(y) ^ Holding(x)*

add *ArmEmpty ^ On(x, y)*

当机械臂把手中的物体x放在物体y上时，产生**Stack动作**

Pickup(x)

pre *Clear(x) ^ OnTable(x) ^ ArmEmpty*

del *OnTable(x) ^ ArmEmpty)*

add *Holding(x)*

当机械臂从桌面上拿起物体x时，发生**Pickup动作**

UnStack(x, y)

pre *On(x,y) ^ Clear(x) ^ ArmEmpty*

del *On(x,y) ^ ArmEmpty*

Add *Holding(x) ^ Clear(y)*

当机械臂从另一个物体y上面拿起物体x时，发生**UnStack动作**

PutDown(x)

pre *Holding(x)*

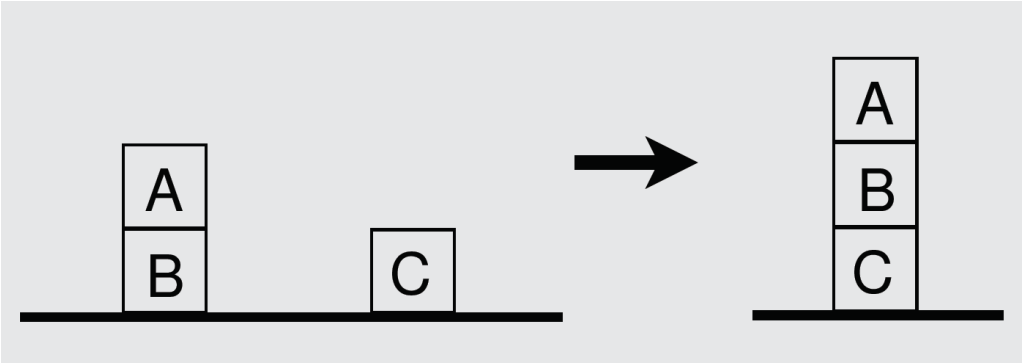
del *Holding(x)*

add *Clear(x) ^ OnTable(x) ^ ArmEmpty*

当机械臂把物体x放在桌面上时，发生**PutDown动作**

规划（Plan）

- 规划是什么？
  - 一系列动作（列表），其中变量替换为常量



UnStack(A,B)

Putdown(A)

Pickup(B)

Stack(B,C)

Pickup(A)

Stack(A,B)

规划问题的形式化表示

- 假设Agent可以执行固定的一组动作：

$$Ac = \{\alpha_1, \ldots, \alpha_n\}$$

- 对动作$\alpha \in Ac$的描述用三元组表示：

$$< P_\alpha, D_\alpha, A_\alpha >$$

  - P_α：刻画动作α的前提条件
  - D_α：刻画通过执行动作α（删除表）变为假的事实
  - A_α：刻画通过执行动作α（增加表）变为真的事实

- 规划问题可以用三元组表示：

$$< B_0, Ac, I >$$

  - B_0：Agent关于世界的初始状态的信念
  - Ac：动作的集合
  - I：要实现的目标（或者意图）

- 规划π是一个动作序列：$\pi = (\alpha_1, \ldots, \alpha_n)$
- 动作会改变世界的状态
  - 理性Agent在执行动作后会更新它的信念

$$B_0 \xrightarrow{\alpha_1} B_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} B_n$$

规划的有效性

- 对于问题$< B_0, Ac, I >$，称一个规划π是可接受的（acceptable），当且仅当：

$$\text{对于所有 } 1 \le j \le n,\ B_{j-1} \models P_{\alpha j}$$

第j步的动作α_j的前提条件在前一个信念B_{j-1}中是可满足的

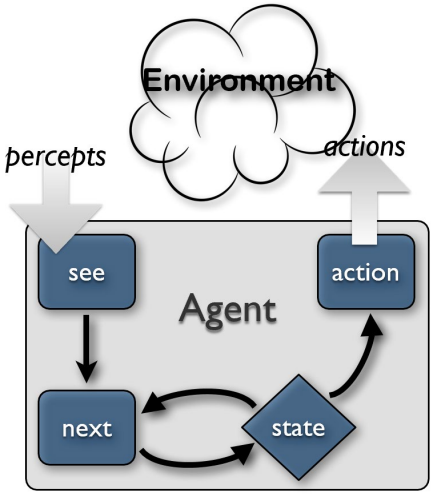
- 一个规划π是正确的（correct），当且仅当：
  - 它是可接受的
  - $B_n \models I$，即目标在由规划产生的最后的环境状态中

实用推理Agent

- 实用推理=慎思过程+目标手段推理
- 目标手段推理
- **实现一个实用推理Agent**
- 过程推理系统

实现一个实用推理Agent

- 第一步：




```
Agent Control Loop Version 1
1.  while true
2.       observe the world;
3.       update internal world model;
4.       deliberate about what intention
           to achieve next;
5.       use means-ends reasoning to get
           a plan for the intention;
6.       execute the plan
7.  end while
```

- 感知函数 $see : E \rightarrow Per$
- 下一个状态函数

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

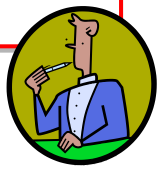
信念修正函数

$$brf : \mathbf{2}^{Bel} \times Per \rightarrow \mathbf{2}^{Bel}$$

- 使算法更正式些：

```
Agent Control Loop Version 2
1.  B := B_0; /* initial beliefs */
2.  while true do
3.      get next percept ρ;
4.      B := brf(B, ρ);
5.      I := deliberate(B);
6.      π := plan(B, I);
7.      execute(π)
8.  end while
```

怎样实现这个慎思函数呢？



- $I \subseteq Int$，Int为所有意图的集合
- $brf()$：信念修正函数
- $plan()$：规划算法，输出一个规划
- execute(π)：依次执行规划π中每个动作的函数

慎思过程

- Agent如何慎思？
  - 首先尝试了解可以使用的**选项**（options）
  - 在它们之间**选择**，并**承诺**（commit）其中一些
- 选择的选项就是意图
- 慎思函数可以分解为两个不同的功能组件：
  - 选项生成（ option generation ）　　得到愿望集
  - 过滤（ filtering ）　　得到意图集

选项生成 & 过滤

- 选项生成函数

$$options : 2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$$

  - 接受Agent当前的信念和意图，以此为基础产生可能的选项或愿望的集合

- 过滤函数

$$filter : 2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$$

  - 从竞争的选项中做出“最佳”的选择，供Agent做出承诺

信念/愿望/意图（BDI）模型

信念修正函数

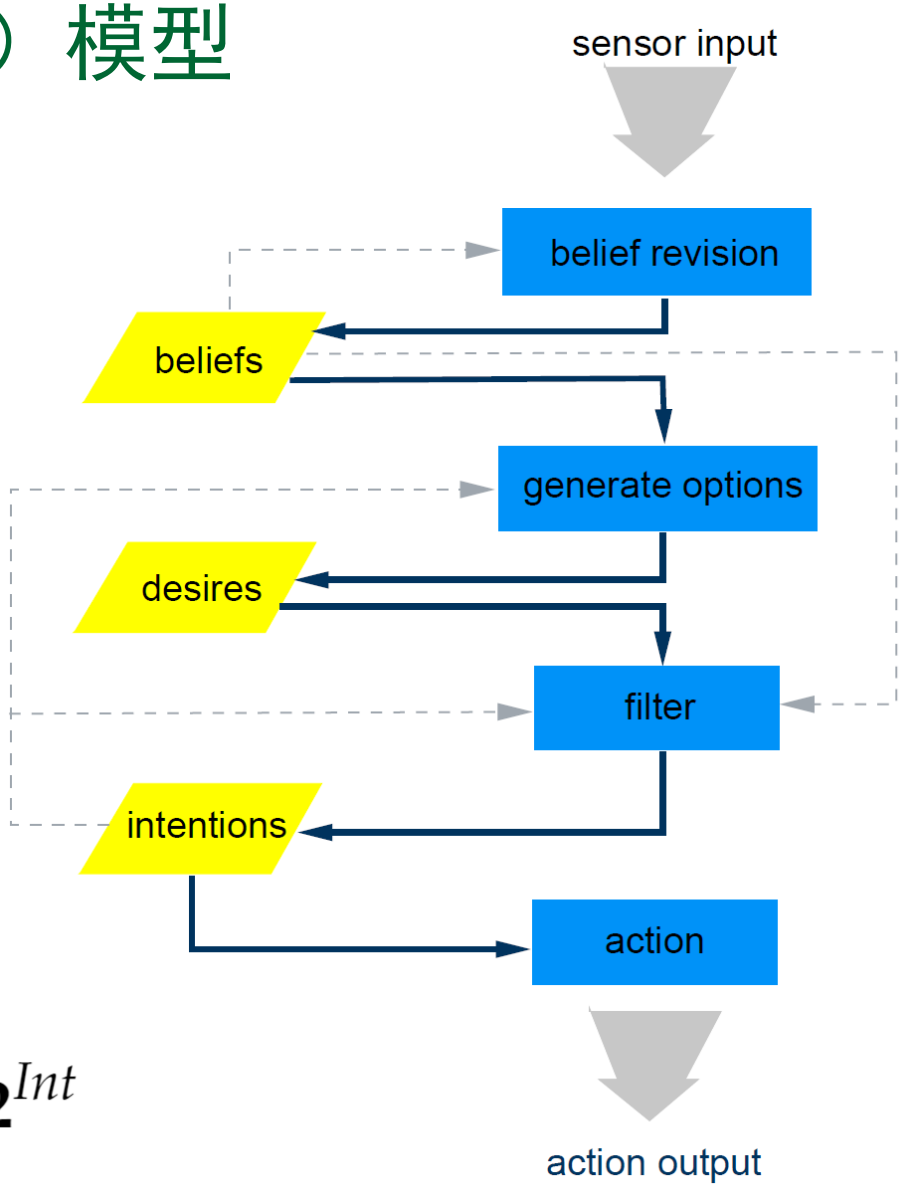
$$brf : 2^{Bel} \times Per \rightarrow 2^{Bel}$$

选项产生函数

$$options : 2^{Bel} \times 2^{Int} \rightarrow 2^{Des}$$

过滤函数

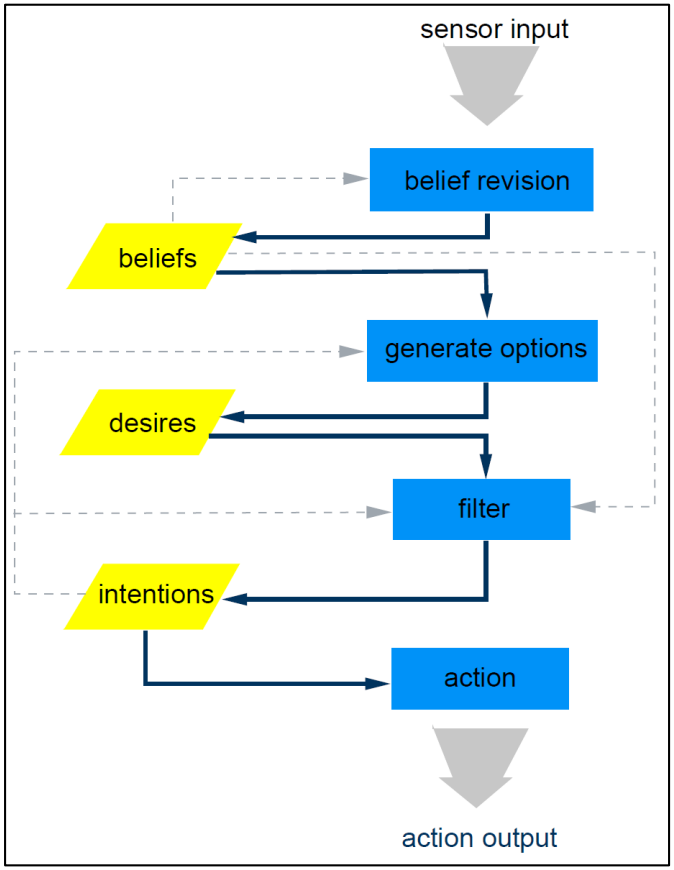
$$filter : 2^{Bel} \times 2^{Des} \times 2^{Int} \rightarrow 2^{Int}$$



盲目承诺Agent

Agent Control Loop Version 3

1. $B := B_0$;
2. $I := I_0$;
3. while true do
4. get next percept ρ;
5. $B := brf(B, \rho)$;
6. $D := options(B, I)$;
7. $I := filter(B, D, I)$;
8. $\pi := plan(B, I)$;
9. $execute(\pi)$
10. end while



- 存在什么问题？

对手段和目标都盲目承诺

承诺策略

- Agent用来决定什么时候更新意图和规划的机制
- Agent对目标和手段都有承诺
- 盲目承诺（也称狂热承诺）
  - Agent将一直维持一个意图，直到它相信这个意图真的已经实现了为止

- 如何修改呢？

如果规划有误，则重新规划

仅对目标盲目承诺的Agent

- 当规划π中没有动作时，$empty(\pi)$为真
- $hd(\pi)$返回规划π中第一个动作
- $tail(\pi)$返回除去规划头部的剩余所有规划
- $sound(\pi, I, B)$代表在给定B时，π是实现I的正确的规划

```
Agent Control Loop Version 4

1.  B := B_0;
2.  I := I_0;
3.  while true do
4.      get next percept ρ;
5.      B := brf(B, ρ);
6.      D := options(B, I);
7.      I := filter(B, D, I);
8.      π := plan(B, I);
9.      while not empty(π) do
10.         α := hd(π);
11.         execute(α);
12.         π := tail(π);
13.         get next percept ρ;
14.         B := brf(B, ρ);
15.         if not sound(π, I, B) then
16.             π := plan(B, I)
17.         end-if
18.     end-while
19. end-while
```

仅对目标盲目承诺的Agent（续）

- 对手段不再盲目承诺
  - 每执行一个动作后，会修正信念B（第14行）
  - 并确保在给定B时，π是实现I的正确的规划（第15行）
  - 如果π不再是正确的规划，则通过规划算法重新产生新的π（第16行）
- 对目标（意图）仍然盲目承诺
  - 不停下来考虑它的意图是否合适

```
Agent Control Loop Version 4

1.  B := B_0;
2.  I := I_0;
3.  while true do
4.      get next percept ρ;
5.      B := brf(B, ρ);
6.      D := options(B, I);
7.      I := filter(B, D, I);
8.      π := plan(B, I);
9.      while not empty(π) do
10.         α := hd(π);
11.         execute(α);
12.         π := tail(π);
13.         get next percept ρ;
14.         B := brf(B, ρ);
15.         if not sound(π, I, B) then
16.             π := plan(B, I)
17.         end-if
18.     end-while
19. end-while
```

专一承诺Agent

- 修改
  - 停下来确定意图是否已经实现了或者意图是否已经不可能实现
- 特点：在满足下列条件之一时重新考虑意图
  - 已经完全执行了一个规划来实现当前意图
  - 相信已经实现了当前意图
  - 相信当前意图不再可能实现

专一承诺（Single Minded Commitment）

```
Agent Control Loop Version 5

1.  B := B_0;
2.  I := I_0;
3.  while true do
4.      get next percept ρ;
5.      B := brf(B, ρ);
6.      D := options(B, I);
7.      I := filter(B, D, I);
8.      π := plan(B, I);
9.      while not(empty(π)
                  or succeeded(I, B)
                  or impossible(I, B)) do
10.         α := hd(π);
11.         execute(α);
12.         π := tail(π);
13.         get next percept ρ;
14.         B := brf(B, ρ);
15.         if not sound(π, I, B) then
16.             π := plan(B, I)
17.         end-if
18.     end-while
19. end-while
```

坦率承诺Agent

- 专一承诺Agent存在的问题
  - 当前意图可能不再是“最佳”的，因为在规划和动作执行的过程中，环境可能发生变化
- 修改
  - 在每次执行一个动作后，都会有慎思过程来重新计算意图

坦率承诺

（Open Minded Commitment）

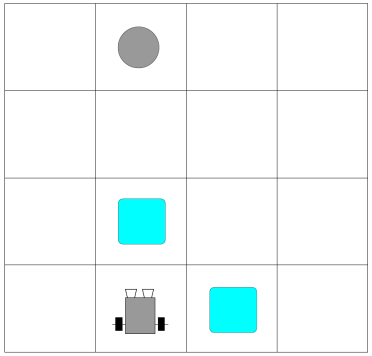
```
Agent Control Loop Version 6

1.  B := B_0;
2.  I := I_0;
3.  while true do
4.      get next percept ρ;
5.      B := brf(B, ρ);
6.      D := options(B, I);
7.      I := filter(B, D, I);
8.      π := plan(B, I);
9.      while not(empty(π)
                  or succeeded(I, B)
                  or impossible(I, B)) do
10.         α := hd(π);
11.         execute(α);
12.         π := tail(π);
13.         get next percept ρ;
14.         B := brf(B, ρ);
15.         D := options(B, I);
16.         I := filter(B, D, I);
17.         if not sound(π, I, B) then
18.             π := plan(B, I)
19.         end-if
20.     end-while
21. end-while
```

使用意图重考虑的Agent

- 两难的局面
  - 专一承诺Agent：不停下来充分地重新考虑它的意图，可能这些意图不再是最佳的
  - 坦率承诺Agent：不断地重复考虑它的意图，将会没有足够的时间实现这些意图
- 解决方案
  - 引入一个布尔函数reconsider决定是否重新考虑意图

```
Agent Control Loop Version 7

1.  B := B_0;
2.  I := I_0;
3.  while true do
4.      get next percept ρ;
5.      B := brf(B, ρ);
6.      D := options(B, I);
7.      I := filter(B, D, I);
8.      π := plan(B, I);
9.      while not(empty(π)
                  or succeeded(I, B)
                  or impossible(I, B)) do
10.         α := hd(π);
11.         execute(α);
12.         π := tail(π);
13.         get next percept ρ;
14.         B := brf(B, ρ);
15.         if reconsider(I, B) then
16.             D := options(B, I);
17.             I := filter(B, D, I);
18.         end-if
19.         if not sound(π, I, B) then
12.             π := plan(B, I)
21.         end-if
22.     end-while
23. end-while
```

重新考虑最佳意图



- Kinny和Georgeff在1991年通过瓦片世界的实验研究了重新考虑意图策略的有效性：

Kinny, D. and Georgeff, M. Commitment and effectiveness of situated agents. In IJCAI, pages 82-88, Sydney, Australia, 1991.

- 两种不同类型的重新考虑策略：
  - 鲁莽的（bold）Agent：在当前规划被全部执行以前，从不停下来重新考虑它们的意图
  - 谨慎的（cautious）Agent：在执行每个动作后，都停下来重新考虑

重新考虑最佳意图（续）

在一个Agent控制回路/循环中世界变化的次数

- 环境的动态性以世界变化率γ表示
- 如果γ低（世界变化不快），鲁莽的Agent比谨慎的Agent做得更好
  - 谨慎的Agent要花费时间重新考虑它们的承诺，而鲁莽的Agent则忙于努力实现自己的意图
- 如果γ高（世界变化频繁），谨慎的Agent胜过鲁莽的Agent
  - 谨慎的Agent能够发现意图什么时候消失，并且能够在偶然的运气和新的机会出现时，利用它们

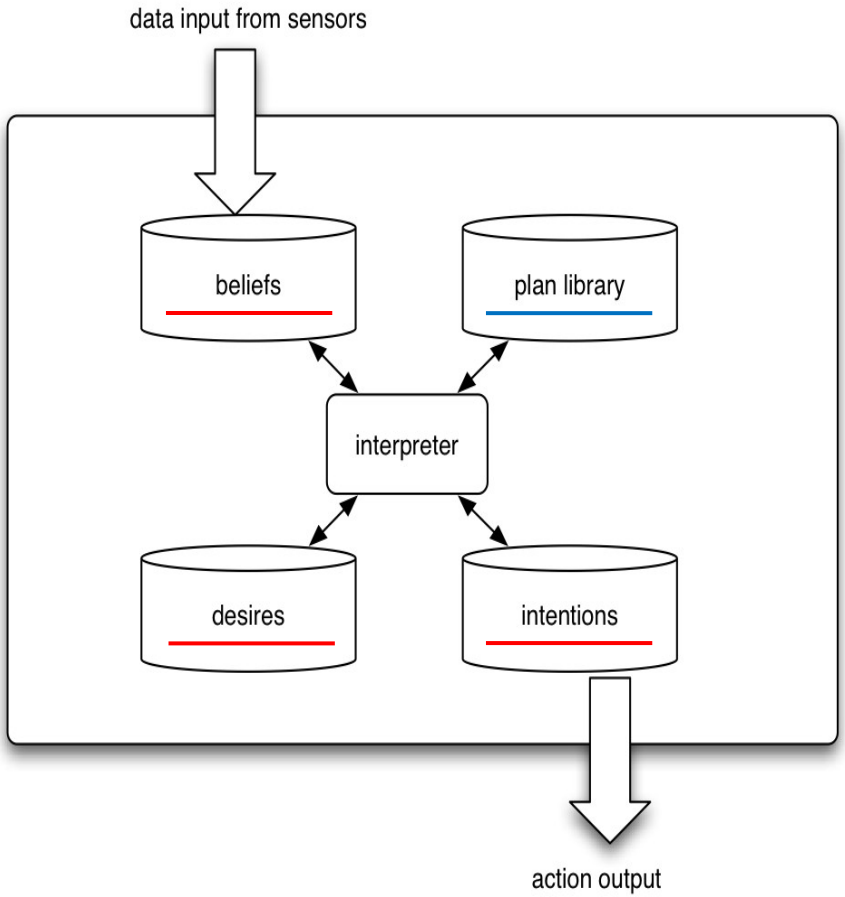
实用推理Agent

- 实用推理=慎思过程+目标手段推理
- 目标手段推理
- 实现一个实用推理Agent
- **过程推理系统**

过程推理系统（Procedure Reasoning System, PRS）

- 过程推理系统（PRS）：由斯坦福大学研究所的Michael Georgeff等研发
- 最早使用BDI体系结构来开发软件Agent并成功用于各种应用的系统之一
- 不同版本的PRS
  - 澳大利亚人工智能研究所的DMARS
  - 密歇根大学用C++实现的PRS系统UMPRS
  - 利用Java程序设计语言实现的JAM

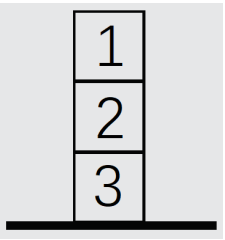
PRS系统的结构



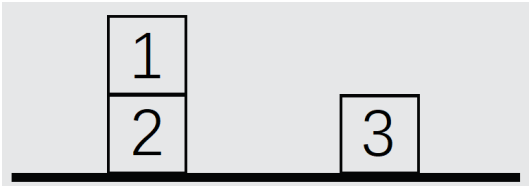
- 采用BDI体系结构
- 拥有一个预先编译好的规划库
  - 规划库拥有一个或者多个规划
  - 这些规划是事先由Agent程序员手工构造的
- 每一个规划有下列元素
  - 目标——规划的后件
  - 上下文——规划的前件（前提条件）
  - 内容——规划的“方法”部分，即要执行的动作序列

PRS系统实例：JAM系统

- Agent的顶层目标
  - 实现目标blocks_stacked



- Agent的初始信念
  - 在FACTS部分给出



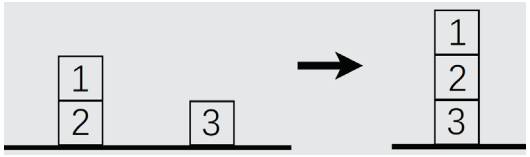
GOALS:
ACHIEVE blocks_stacked;

FACTS:
// Block1 on Block2 initially so need
//to clear Block2 before stacking.

FACT ON "Block1" "Block2";
FACT ON "Block2" "Table";
FACT ON "Block3" "Table";
FACT CLEAR "Block1";
FACT CLEAR "Block3";
FACT CLEAR "Table";
FACT initialized "False";

PRS系统实例：JAM系统（续）

- 顶层规划
  - 实现目标blocks_stacked



CONTEXT为空（即为真），这个规划可以直接执行

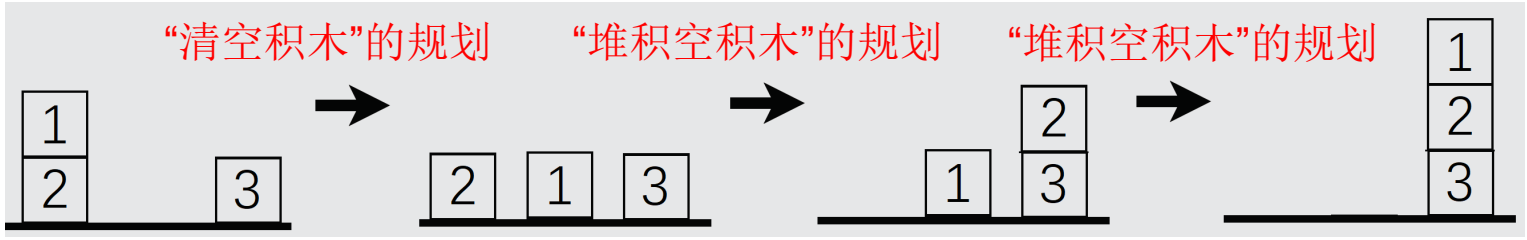
BODY由一些指令和目标构成

在执行开始时，会把要实现的意图推入**意图堆栈**

```
Plan: {
NAME: "Top-level plan"
DOCUMENTATION:
"Establish Block1 on Block2 on Block3."
GOAL:
ACHIEVE blocks_stacked;
CONTEXT:
BODY:
EXECUTE print "Goal: Blk1 on Blk2 on Blk3 on Table.\n";
EXECUTE print "World Model at start is:\n";
EXECUTE printWorldModel;

EXECUTE print "ACHIEVEing Block3 on Table.\n";
ACHIEVE ON "Block3" "Table";

EXECUTE print "ACHIEVEing Block2 on Block3.\n";
ACHIEVE ON "Block2" "Block3";

EXECUTE print "ACHIEVEing Block1 on Block2.\n";
ACHIEVE ON "Block1" "Block2";

EXECUTE print "World Model at end is:\n";
EXECUTE printWorldModel;
}
```

Agent的一个FACT

PRS系统实例：JAM系统（续）



- “清空积木”的规划

EFFECTS 决定了 Agent 在成功执行了BODY中的所有指令后，应该做些什么

给出了规划失败时，Agent要执行的动作

```
Plan: {
NAME: "Clear a block"
GOAL:
ACHIEVE CLEAR $OBJ;
CONTEXT:
FACT ON $OBJ2 $OBJ;
BODY:
EXECUTE print "Clear " $OBJ2 " from on top of " $OBJ "\n";
EXECUTE print "Move " $OBJ2 " to table.\n";
ACHIEVE ON $OBJ2 "Table";
EFFECTS:
EXECUTE print "Clear: Retract ON " $OBJ2 " " $OBJ "\n";
RETRACT ON $OBJ2 $OBJ;
FAILURE:
EXECUTE print "\n\nClearing block " $OBJ " failed!\n\n";
}
```

PRS系统实例：JAM系统（续）

- “堆积空积木”的规划

这个规划使用了**效用**，将其用于**慎思过程**

```
Plan: {
NAME: "Stack blocks that are already clear"
GOAL:
ACHIEVE ON $OBJ1 $OBJ2;
CONTEXT:
BODY:
EXECUTE print "Making sure " $OBJ1 " is clear\n";
ACHIEVE CLEAR $OBJ1;
EXECUTE print "Making sure " $OBJ2 " is clear.\n";
ACHIEVE CLEAR $OBJ2;
EXECUTE print "Moving " $OBJ1 " on top of " $OBJ2 ".\n";
PERFORM move $OBJ1 $OBJ2;
UTILITY: 10;
FAILURE:
EXECUTE print "\n\nStack blocks failed!\n\n";
}
```

- 在PRS系统中，慎思过程是从不同的规划中选择的过程

- Agent如何慎思
- 1. 通过使用元级规划来实现，可以在执行时修改Agent的意图
- 2. 可以对规划使用效用，只需要选择效用最大的规划

小结

- 实用推理=慎思过程+目标手段推理
  - 实用推理：直接通过推理得到动作
  - 慎思过程：决定要实现的状态，结果是意图集
  - 目标手段推理：决定如何实现这些状态，结果是规划
- 目标手段推理
  - 规划，STRIPS规划器，规划问题的形式化表示
  - 例子：积木世界
- 实现一个实用推理Agent
  - BDI体系结构，承诺策略，意图重考虑
- 过程推理系统
  - 例子：用Java实现的JAM系统

课后作业2-8

- 考虑下图中的并发MetateM程序，解释Agent在这个系统中的行为。

提示：有5个Agent：

- SnowWhite是资源的提供者。可以把她想象成白雪公主，手中有糖
- eager, greedy, courteous, shy是资源的消费者。可以把他们想象成不同类型的小矮人

$$
\begin{aligned}
&SnowWhite(ask)[give]: \\
&\bullet ask(x) \Rightarrow \Diamond give(x) \\
&give(x) \wedge give(y) \Rightarrow (x = y) \\
&eager(give)[ask]: \\
&\text{start} \Rightarrow ask(eager) \\
&\bullet give(eager) \Rightarrow ask(eager) \\
&greedy(give)[ask]: \\
&\text{start} \Rightarrow \Box ask(greedy) \\
&courteous(give)[ask]: \\
&((\neg ask(courteous)\ S\ give(eager)) \wedge \\
&(\neg ask(courteous)\ S\ give(greedy))) \Rightarrow ask(courteous) \\
&shy(give)[ask]: \\
&\text{start} \Rightarrow \Diamond ask(shy) \\
&\bullet ask(x) \Rightarrow \neg ask(shy) \\
&\bullet give(shy) \Rightarrow \Diamond ask(shy)
\end{aligned}
$$

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。

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 := B_0;
3.  I := I_0;
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。

编程作业1

- 阅读论文：

  Kinny, D. and Georgeff, M. Commitment and effectiveness of situated agents. In IJCAI, pages 82-88, Sydney, Australia, 1991.

  复现论文中的实验结果。

  提交代码（用Python或者C++实现 ）和实验报告。

  截止时间为：**2023年4月11日**