CPT302 W2

Simple Agent

Agent 是一种 computer system，能够在某种环境中采取自主行动 (autonomous action)，以实现其委派的目标 (delegated goals)。

Agent 与其环境处于紧密耦合 (close-coupled，模块之间依赖程度高) 的持续交互中 (sense -> decide -> act -> sense -> decide ...)。

Properties of Environments

Accessible vs. inaccessible:

Accessible environment 是 agent 可以获得关于环境状态的完整、准确和最新的信息的环境。中等复杂的环境是 inaccessible 的，访问环境越方便，构建在其中运行的 agent 就越简单。

Deterministic vs. non-deterministic:

在 deterministic environment 中，任何行动都有一个被保证的效果 (对行为造成的状态没有不确定性)。物理世界可以被认为是 non-deterministic 的，非确定性环境对 agent 设计人员提出了更大的挑战。

Static vs. dynamic:

Static environment 保持不变，但 agent 执行的操作除外。Dynamic environment 有其他进程在其上运行，并且以 agent 无法控制的方式变化。物理世界是一个高度动态的环境。

Discrete vs. continuous:

如果一个环境中有固定的，有限数量的行动和感知，那么它就是离散的。国际象棋游戏是一个离散的环境，而汽车驾驶是一个连续的环境。

Autonomy

Autonomy as a spectrum:

- Humans: freedom with respect to beliefs, goals and actions
- Software services: no freedom, they do what they are told

Adjustable autonomy:

当满足某些条件时，决策权从 agent 转移到一个人身上：

- 当 agent 相信人类会做出更好的决定时
- 当环境存在一定程度的不确定性时
- 当决策可能造成伤害时
- 当 agent 缺乏自己做出决定的能力时

Decisions and Actions

Agent 具有一组可用操作（修改环境的能力）。

Agent 的行动具有相关的先决条件，这些先决条件决定了可以应用这些行动的可能情况。

代理的关键问题是决定执行哪个操作以最好地满足其（设计）目标。

Examples of (Simple) Agents

Control systems:

- 任何控制系统都可以被视为代理。一个简单的例子：thermostat (goal: to maintain certain temperature, actions: heat on or off)

Software daemons:

- 监视软件环境并执行操作以对其进行修改的任何软件。例如：anti-virus (or anti-spam) software

Intelligent Agents

Behavior

Intelligent agent（应）表现出 3 种类型的行为：

- Reactivity：能够感知他们的环境，并（快速）响应变化，以满足他们的目标
- Pro-activeness：能够采取主动（目标导向的行为）以满足他们的目标

- Social ability：能够与其他代理（可能还有人类）进行交互，以满足他们的目标

在 reactivity 和 pro-activeness 之间实现适当的平衡是很重要且困难的：

- Agent 应系统地实现其目标，例如，通过构建和遵循复杂的计划。但是，它们不应该盲目地遵循这些计划，如果很明显计划不会起作用或当目标不再有效时，agent 应该能够对新情况做出反应。但是，agent 不应该为了不失去总体目标而不断做出反应。

Social ability 是通过以下方式与其他 agent 交互的能力：

- cooperation - 作为一个团队共同努力，实现共同的目标
- coordination - 管理行动之间的相互依赖关系
- negotiation - 就共同关心的事项达成协议的能力

Agents and Objects

Agent 和 object (对象) 是一个东西吗?

- Object 封装某些状态，object 通过 message passing 进行通信，object 具有处理此状态的操作相对应的 method。
- agent 体现了比 object 更强的自治概念，特别是它们决定是否根据另一个 agent 的请求执行操作 (object 根据要求执行此操作；agent 这样做是因为它们想这样做，或因为利益)
- Agent 可执行 flexible behavior (reactive, proactive, social)，这是标准对象模型不能处理的
- 多智能体系统本质上是多线程的，假定每个代理程序至少具有一个控制线程

Agents as Intentional Systems

Intentional System：一个实体，可以预测其行为，通过归因于信念，欲望和理性敏锐度 (比如人类)。

计算领域最重要的发展是基于新的抽象（过程抽象，抽象数据类型，对象）。随着软件系统变得越来越复杂，我们需要更强大的抽象和隐喻来解释它们的操作（低级解释变得不切实际）。

Agent（作为 Intentional System）代表了一种进一步和更强大的抽象，用于描述，解释和预测复杂系统的行为。

Abstract Architectures for Agents

假设环境可以处于有限集合 E 中的任何一个离散状态：

$$E = \{e, e^{'}, \ldots\}$$

假定 agent 具有一组可能的操作，这些操作可以转换环境的状态：

$$Ac = \{\alpha, \alpha^{'}, \ldots\}$$

Agent 在环境中的 run r 是一系列交错的环境状态和操作：

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

let：

$\mathscr{R}$ be the set of all such possible finite sequences (over E and Ac)

$\mathscr{R}^{Ac}$ be the subset of these runs that end with an action

$\mathscr{R}^{E}$ be the subset of these runs that end with an environment state

注：$\mathscr{R}$ 是 e 和 α 所有可能E的有限排列；$\mathscr{R}^{Ac}$ 是以 α 结尾的 run；$\mathscr{R}^{E}$ 是以 e 结尾的 run；

Environment

A ***state transformer function*** represents behavior of the environment：

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

注：2^{E} 代表 E 中所有元素的组合情况；比如 E = (1, 2, 3)，那么 2^{E} 会是长度为 8 的集合，即 ((), 1, 2, 3, (1, 2), (1, 3), (2, 3), (1, 2, 3))。

- 环境是：
  - history dependent - 当前状态在某种程度上是由之前的操作决定的
  - non-deterministic - 执行操作的结果存在不确定性
- 如果 $\tau(r) = \varnothing$，r 没有可能的继承状态 (successor states)，所以我们说 run has ended (game over)

An environment Env is a triple

$$Env = \ < E, e_0, \tau >$$

where E is a set of environment states, $e_0 \in E$ is initial state, and τ is state transformer function.

Agents and Systems

Agent is a function that maps runs to actions:

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

使 $\mathscr{AG}$ 表示所有 agent 的集合。Agent 根据其迄今为止见证的系统历史记录决定要执行的操作。

一个 system 包含一对代理和环境。任何系统都与一组可能的 runs 相关联，我们通过 $\mathscr{R}(Ag, Env)$ 表示 the set of runs of agent Ag in environment Env。

让 $\mathscr{R}(Ag, Env)$ 仅包含已结束的 runs (terminated or finite runs).

正式地，一个序列：

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

表示环境 $Env =< E, e_0, \tau >$ 中 agent Ag 的 run，如果：

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

注：τ 代表的是当前操作下 (考虑过去历史)，所有可能的后续状态 (把 run 转换为状态的集合)；Ag 代表的是当前状态下 (考虑过去历史)，所有可能的操作 (把 run 转换为操作的集合)。

$\alpha_0 = Ag(e_0)$ 代表在状态 e_0下，可能的操作为 α_0；$e_u \in \tau(e_0, \alpha_0, \cdots, \alpha_{u-1})$ 代表经过 α_{u-1} 的操作，可能会得到一系列状态，e_u 是其中一种。

因此，上面的条件是：run 的起始条件必须是环境的起始条件，run 中其他的状态 (和操作) 都必须是上一步的操作 (和状态) 得到的。

Purely Reactive Agents

这些 agent 决定做什么时不参考历史，它们的决策完全基于现在。

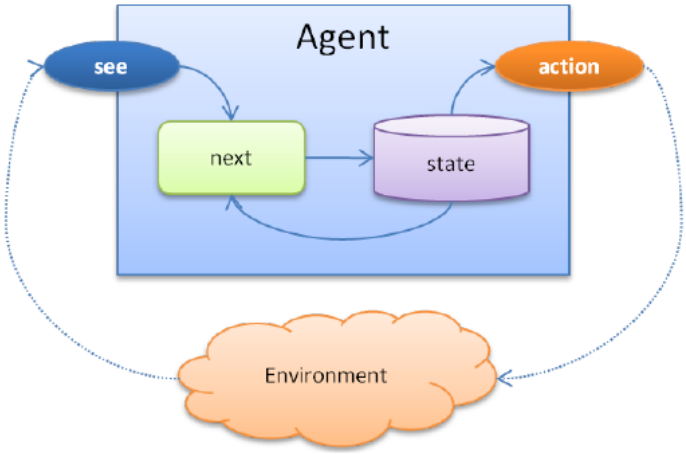
形式上，一个 purely reactive agent 可以表示为：

$$Ag : E \rightarrow Ac$$

注：这里不再使用 run $\mathscr{R}^E$，而是仅依靠当前的状态。

Agents with State

为了构建真实的 agent，抽象模型需要被分解为子系统 (数据和控制) 来重新构建。这些子系统构成了 agent architecture。



此类 agent 具有一些数据结构（状态），用于记录有关环境状态和历史记录。

设 I 是 agent 的所有内部状态的集合。

感知函数 see 表示智能体从其环境中获取信息并将其转换为感知输入的能力。它被定义为：$see : E \to Per$。

Action-selection 函数被定义为：$action : I \to Ac$。

接下来引入了一个额外的函数，用于将内部状态和感知映射到新的内部状态：$next : I \times Per \to I$

1. Agent starts in some initial internal state i_0
2. Observe environment state and generate a percept $see(e)$
3. Update the internal state via the *next* function – set the state to $i_1 = next(i_0, see(e))$
4. Select action via the *action* function – select $action(i_1)$
5. Perform the selected action
6. Go to 2

Tasks for Agents

我们构建代理以便为我们执行任务，任务必须由我们指定。但是我们想告诉代理商该做什么，而不告诉它们该怎么做。

一种方法是将 utilities (效用) 与单个状态相关联 - 代理的任务是达到 utililty 最大化的状态。

Task specification (任务说明) 是一个函数：

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

此函数将每个环境状态和一个实数相关联。

确定代理在特定环境中的整体效用：

- pessimistic approach (悲观方法) - run 中最差状态的效用
- optimistic approach (乐观方法) - run 中最佳状态的效用
- run 中所有状态的效用的总和
- run 中所有状态的效用的平均

缺点：difficult to specify a long term view when assigning utilities to individual (isolated) states (如果 state 是孤立的，该 state 的效用就是 agent 的整体效用，这很 “短视”).

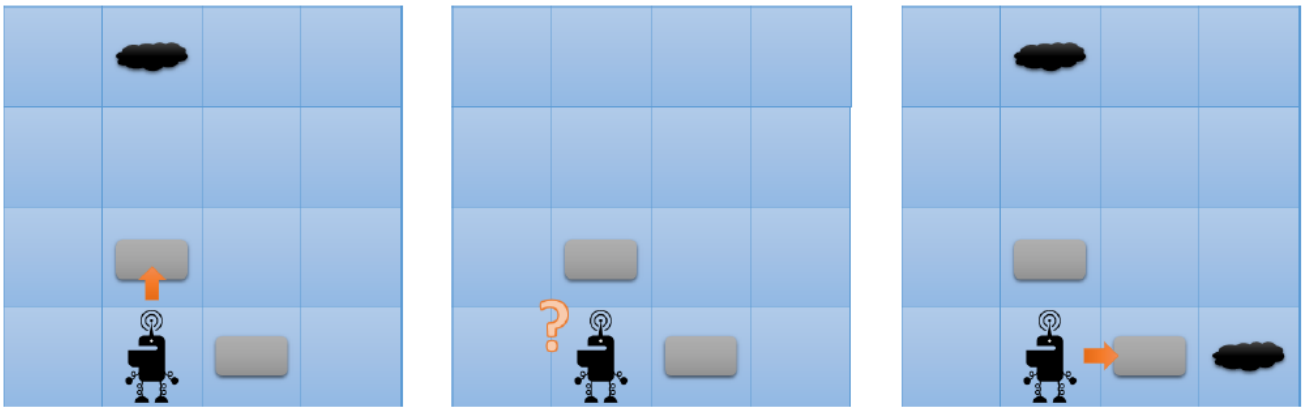
另一种可能性：不将 utility 分配给单个状态，而是分配给 run。

$$u : \mathscr{R} \to \mathbb{R}$$

This approach focuses on a long term view.

Utility in Tileworld

Tileworld 模拟二维网格环境，其中存在代理、瓷砖、障碍物和孔。代理可以向上、向下、向左或向右四个方向移动，如果它位于瓷砖旁边，它可以推动瓷砖。孔必须由代理用瓷砖填补，代理通过把瓷砖推到孔所在的位置来得分，其目的是尽可能多地填补孔。Tileworld 随着孔的随机出现和消失而变化。



代理的性能是通过运行 Tileworld 并计算代理填补的孔的数量和所需的时间来衡量的。代理在某些特定的 run 中的性能被削弱为:

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