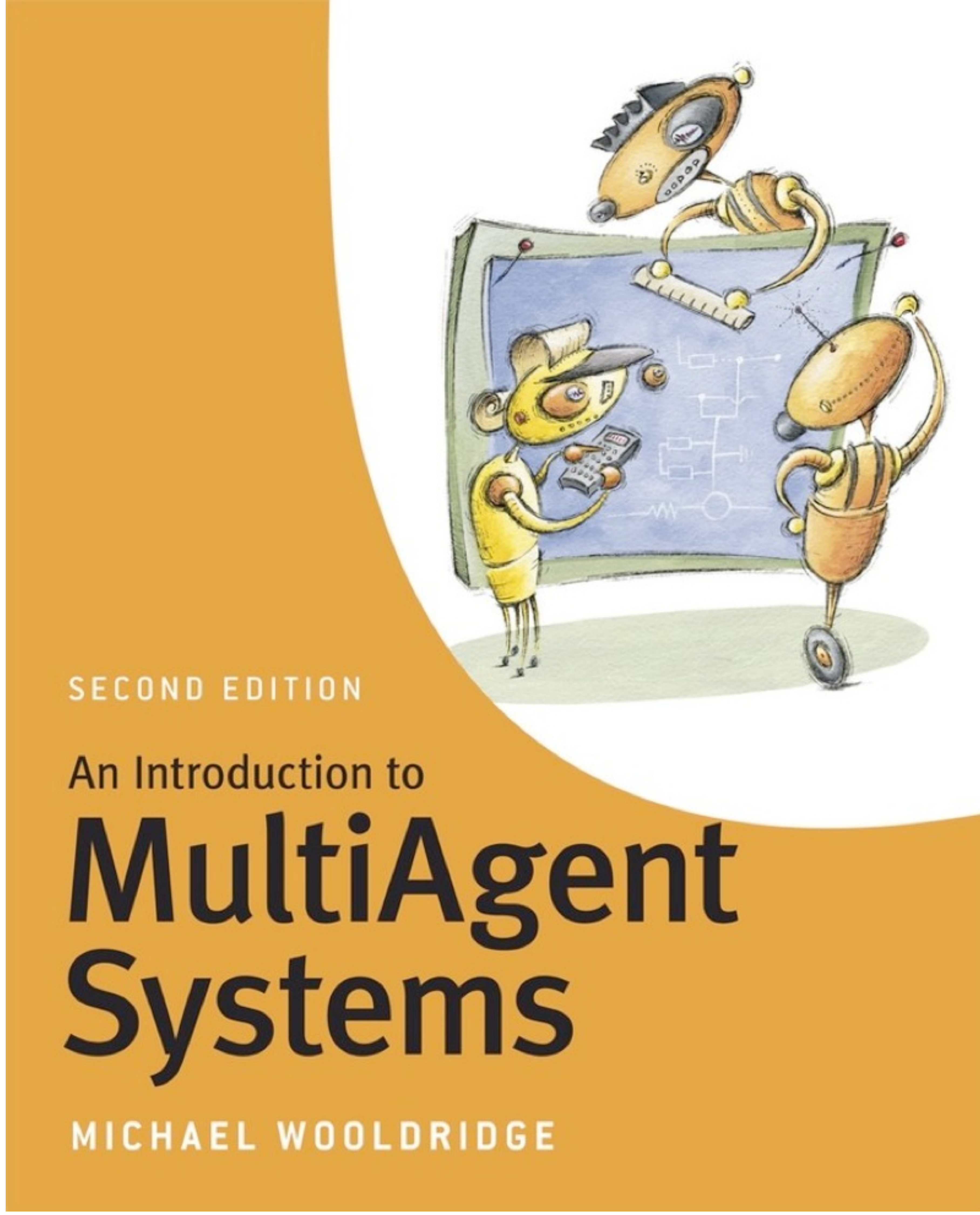
COMP310
Multi-Agent Systems

Chapter 3 - Deductive Reasoning Agents

Dr Terry R. Payne
Department of Computer Science
UNIVERSITY OF LIVERPOOL



Agent Architectures

- Pattie Maes (1991)

"... [A] particular methodology for building [agents]. It specifies how . . . the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions . . . and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology ..."

- Leslie Kaebling (1991)

"... [A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks ..."

Classes of Architecture

- 1956–present: Symbolic Reasoning Agents
  - Agents make decisions about what to do via ***symbol manipulation***.
  - Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do.
- 1985–present: Reactive Agents
  - Problems with symbolic reasoning led to a reaction against this
  - led to the ***reactive agents*** movement, 1985–present.
- 1990-present: Hybrid Agents
  - Hybrid architectures attempt to combine the best of reasoning and reactive architectures.

Symbolic Reasoning Agents

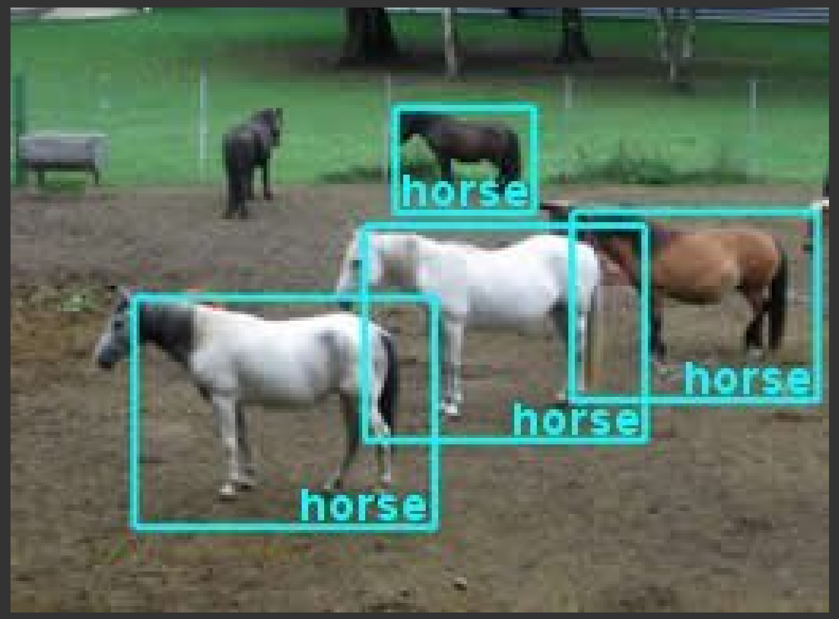
- The classical approach to building agents is to view them as a particular type of knowledge-based system, and bring all the associated methodologies of such systems to bear.
  - This paradigm is known as symbolic AI.

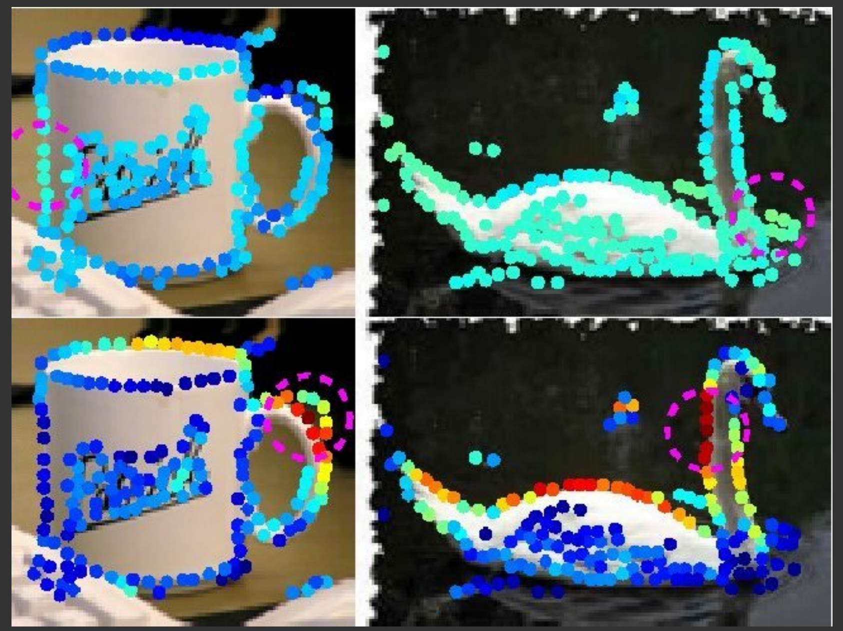
- We define a deliberative agent or agent architecture to be one that:
  - contains an explicitly represented, symbolic model of the world;
  - makes decisions (for example about what actions to perform) via symbolic reasoning.

Two issues

The Transduction Problem

Identifying objects is hard!!!



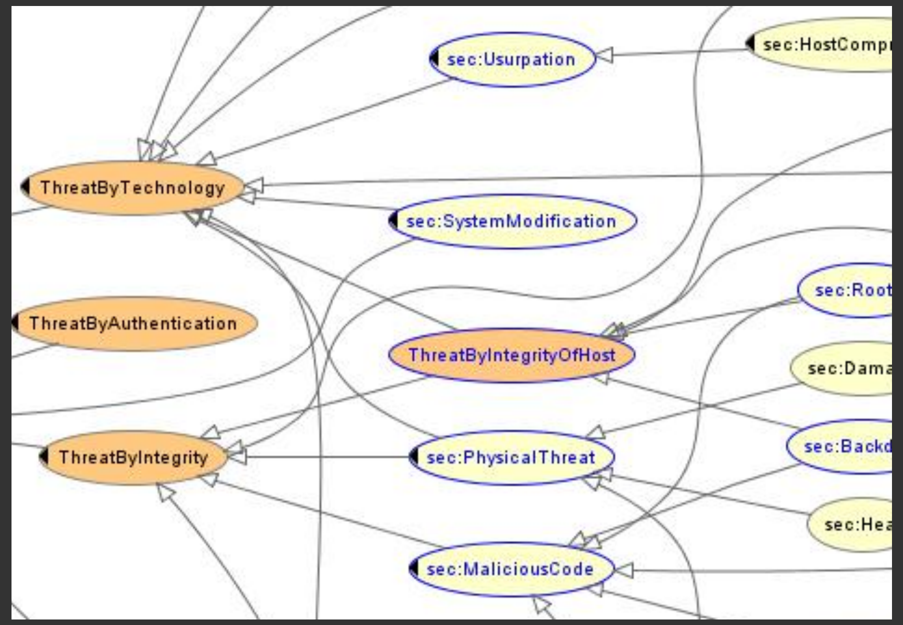


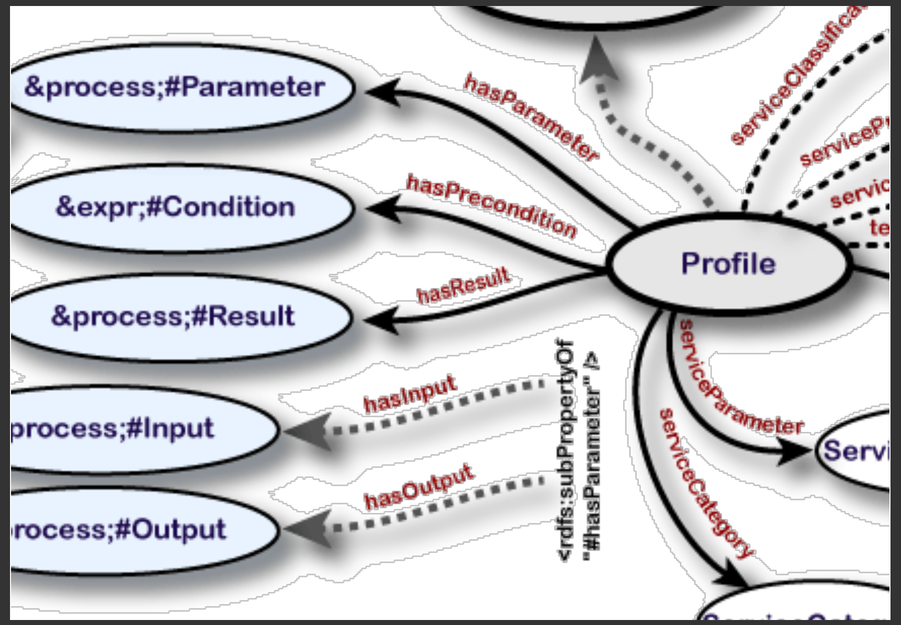
The transduction problem is that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.

This has led onto research into vision, speech understanding, learning…

The Representation/Reasoning Problem

Representing objects is harder!





How to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

This has led onto research into knowledge representation, automated reasoning, planning…

Most researchers accept that neither problem is anywhere near solved.

The representation / reasoning problem

- The underlying problem with knowledge representation/ reasoning lies with the complexity of symbol manipulation algorithms.
  - In general many (most) search-based symbol manipulation algorithms of interest are ***highly intractable***.
  - Hard to find ***compact representations***.
- Because of these problems, some researchers have looked to alternative techniques for building agents; we look at these later.

Deductive Reasoning Agents

- How can an agent decide what to do using theorem proving?
  - Basic idea is to use logic to encode a theory stating the best action to perform in any given situation.

- Let:
  - ρ be this theory (typically a set of rules);
  - Δ be a logical database that describes the current state of the world;
  - Ac be the set of actions the agent can perform;
  - $\Delta \vdash_{\rho} \varphi$ means that φ can be proved from Δ using ρ.

Deductive Reasoning Agents

- How does this fit into the abstract description we talked about last time?
  - The perception function is as before:

$$see : E \rightarrow Per$$

  - of course, this is (much) easier said than done.

- The next state function revises the database Δ :

$$next : \Delta \times Per \rightarrow \Delta$$

- And the action function?
  - Well a possible action function is on the next slide.

Action Function

```
for each $\alpha \in Ac$ do          /* try to find an action explicitly prescribed */
    if $\Delta \vdash_\rho Do(\alpha)$ then
        return $\alpha$
    end-if
end-for

for each $\alpha \in Ac$ do          /* try to find an action not excluded */
    if $\Delta \not\vdash_\rho \neg Do(\alpha)$ then
        return $\alpha$
    end-if
end-for

return null                          /* no action found */
```

An example: The Vacuum World

The Vacuum World

The goal is for the robot to clear up all the dirt.

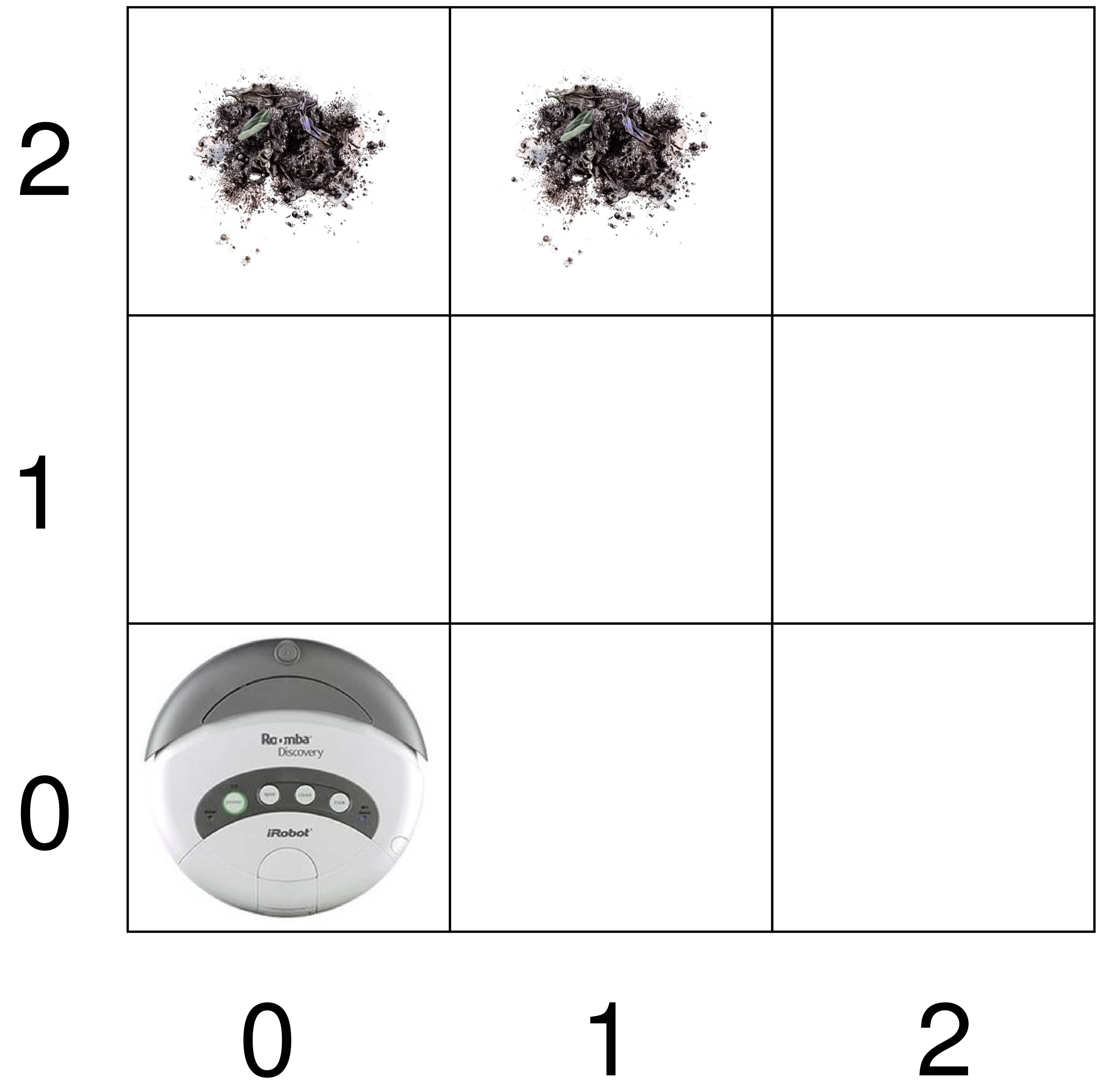
Uses 3 domain predicates in this exercise:

In(x,y)	agent is at *(x,y)*
Dirt(x,y)	there is dirt at *(x,y)*
Facing(d)	the agent is facing direction *d*

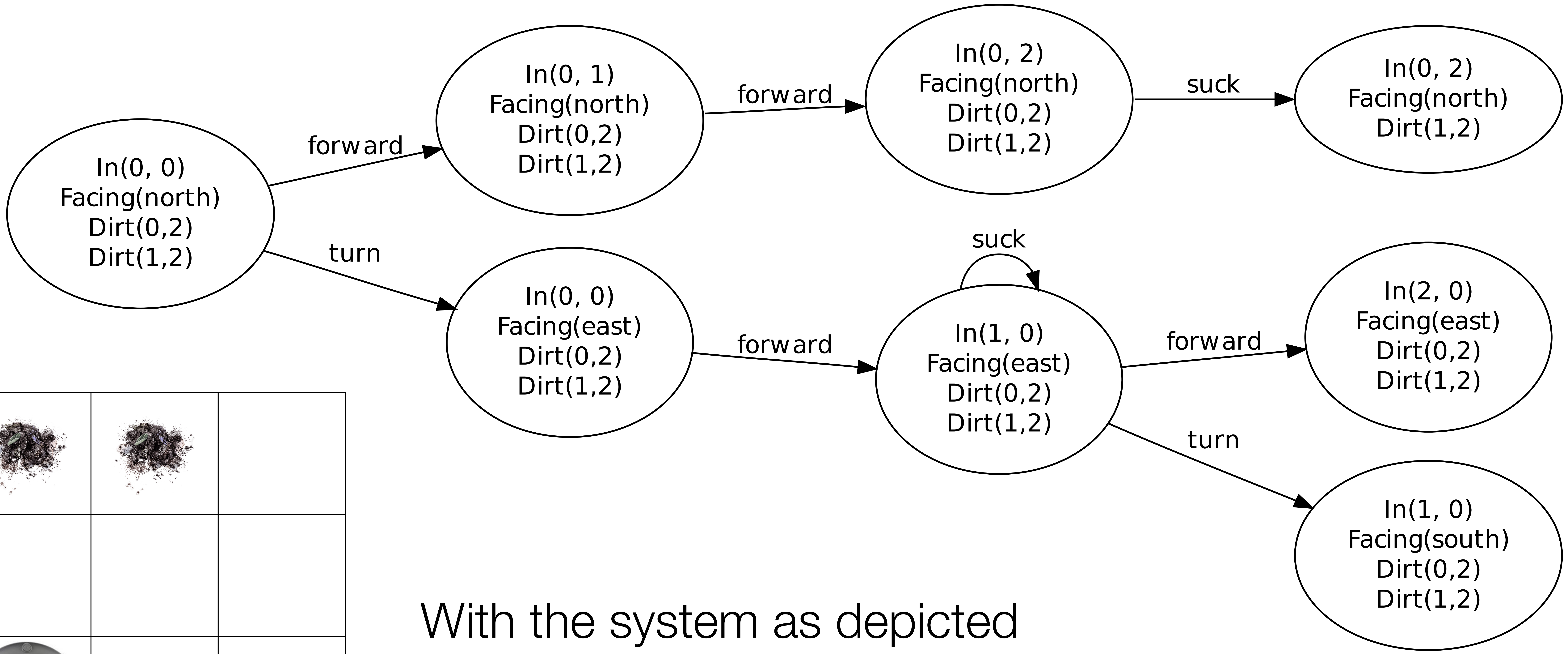
Possible Actions:

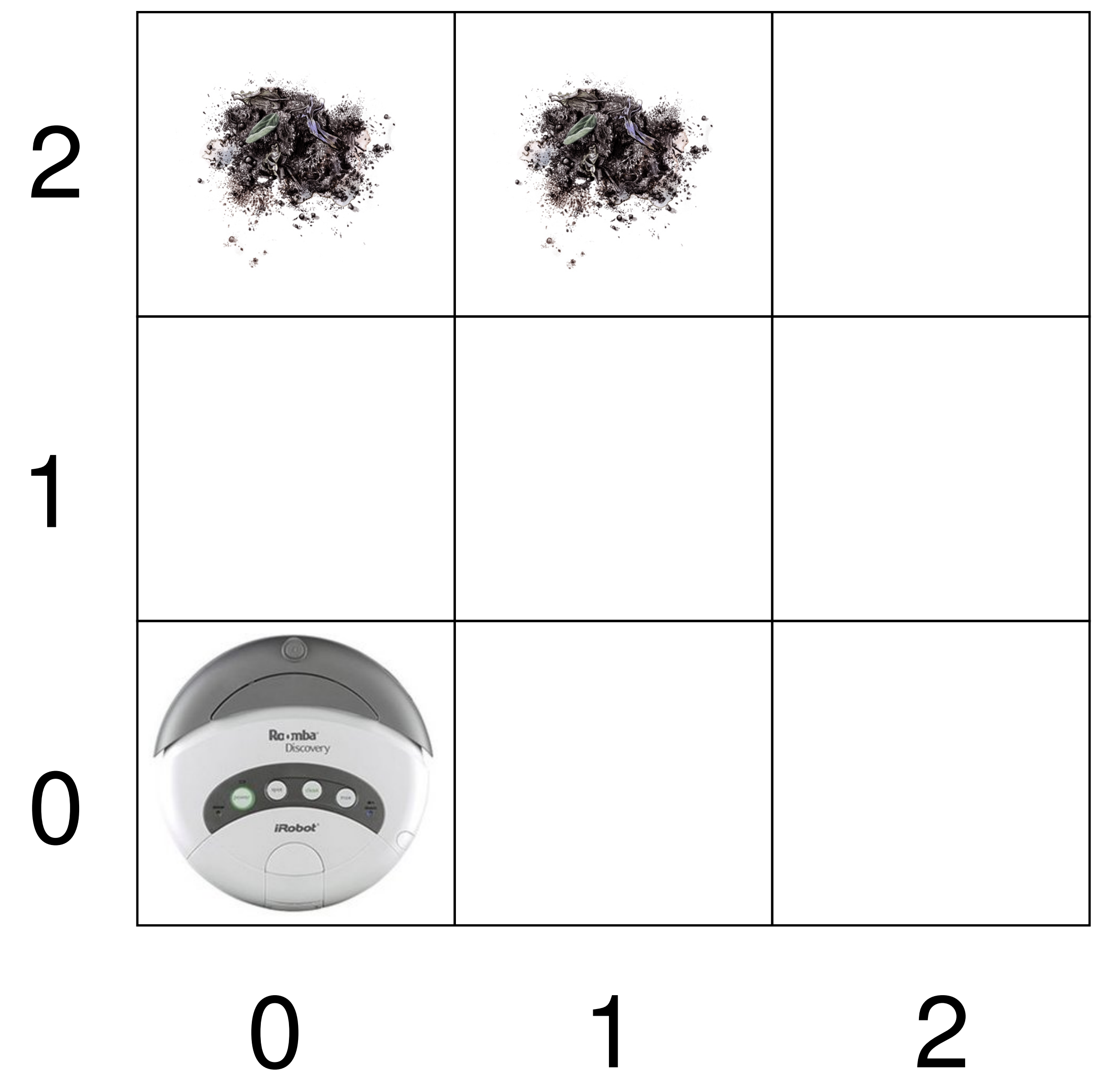
Ac = {turn, forward, suck}

Note: *turn* means "turn right"



The Vacuum World


In(0, 0) Facing(north) Dirt(0,2) Dirt(1,2)
forward
In(0, 1) Facing(north) Dirt(0,2) Dirt(1,2)
forward
In(0, 2) Facing(north) Dirt(0,2) Dirt(1,2)
suck
In(0, 2) Facing(north) Dirt(1,2)
turn
In(0, 0) Facing(east) Dirt(0,2) Dirt(1,2)
forward
In(1, 0) Facing(east) Dirt(0,2) Dirt(1,2)
suck
forward
In(2, 0) Facing(east) Dirt(0,2) Dirt(1,2)
turn
In(1, 0) Facing(south) Dirt(0,2) Dirt(1,2)



With the system as depicted above, here are some possible ways that the system might run.

The Vacuum World

- Rules ρ for determining what to do:

$$\begin{array}{rcl} In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) & \longrightarrow & Do(forward) \\ In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) & \longrightarrow & Do(forward) \\ In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) & \longrightarrow & Do(turn) \\ In(0,2) \wedge Facing(east) & \longrightarrow & Do(forward) \end{array}$$

  - ... and so on!

- Using these rules (+ other obvious ones), starting at *(0, 0)* the robot will clear up dirt.

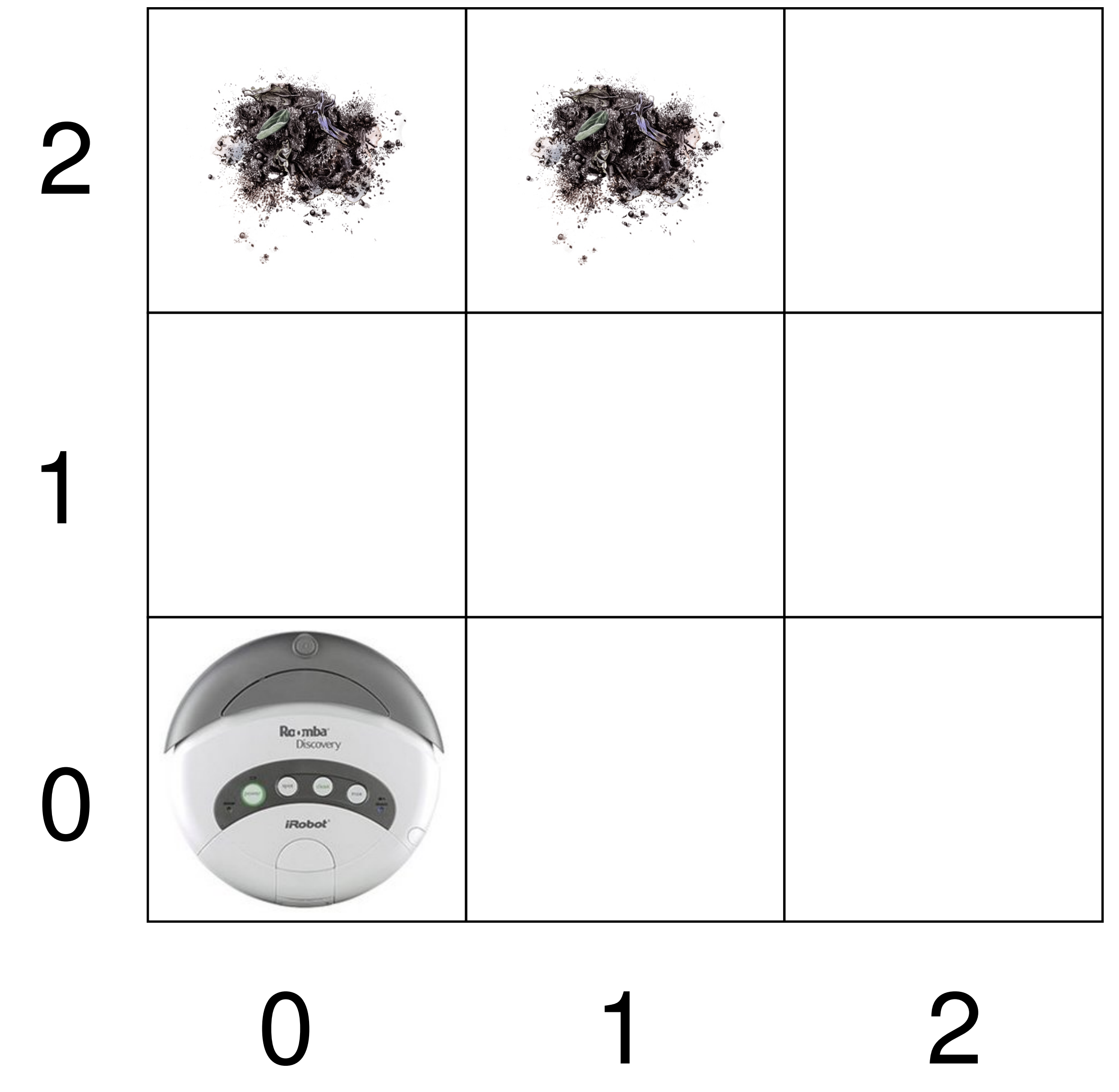
Uses 3 domain predicates in this exercise:

In(x,y)	agent is at *(x,y)*
Dirt(x,y)	there is dirt at *(x,y)*
Facing(d)	the agent is facing direction *d*

Possible Actions:

Ac = {turn, forward, suck}

Note: *turn* means "turn right"



The Vacuum World

- Problems:
  - how to convert video camera input to *Dirt(0, 1)*?
  - decision making assumes a static environment:
    - *calculative rationality*.
  - decision making using first-order logic is ***undecidable***!

- Typical solutions:
  - weaken the logic;
  - use symbolic, non-logical representations;
  - shift the emphasis of reasoning from ***run time*** to ***design time***.

Agent-oriented programming

- Yoav Shoham introduced "agent-oriented programming" in 1990:

"... new programming paradigm, based on a societal view of computation ..."

- The key idea:
  - directly programming agents in terms of intentional notions
    - like belief, desire, and intention
    - Adopts the same abstraction as humans
  - Resulted in the Agent0 programming language

Agent0

- AGENT0 is implemented as an extension to LISP.
  - Each agent in AGENT0 has 4 components:
    - a set of ***capabilities*** (things the agent can do);
    - a set of ***initial beliefs***;
    - a set of ***initial commitments*** (things the agent will do); and
    - a set of ***commitment rules***.
- The key component, which determines how the agent acts, is the commitment rule set.
  - Each commitment rule contains
    - a ***message condition***;
    - a ***mental condition***; and
    - an ***action***.

Agent0 Decision Cycle

On each decision cycle . . .

- The message condition is matched against the messages the agent has received;
    - The mental condition is matched against the beliefs of the agent.
    - If the rule fires, then the agent becomes committed to the action (the action gets added to the agents commitment set).

Actions may be . . .

- Private
    - An externally executed computation
- Communicative
    - Sending messages

Messages are constrained to be one of three types . . .

- requests
    - To commit to action
- unrequests
    - To refrain from action
- Informs
    - Which pass on information

Commitment Rules

- This rule may be paraphrased as follows:
  - if I receive a message from *agent* which requests me to do *action* at *time*, and I believe that:
    - *agent* is currently a friend;
    - I can do the *action*;
    - at *time*, I am not committed to doing any other *action*,
  - then commit to doing *action* at *time*.

A commitment Rule

```
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)
)
```

PLACA

- A more refined implementation was developed by Becky Thomas, for her 1993 doctoral thesis.
- Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0
  - the inability of agents to plan, and communicate requests for action via high-level goals.
- Agents in PLACA are programmed in much the same way as in AGENT0, in terms of mental change rules.

PLACA

- A more refined implementation was developed by Becky Thomas, for her 1993 doctoral thesis.

- Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0
  - the inability of agents to ***plan***, and ***communicate requests*** for action via high-level goals.

- Agents in PLACA are programmed in much the same way as in AGENT0, in terms of mental change rules.

PLACA: Mental Change Rule

- If:
  - someone *asks* you to xerox something *x* at time *t* and you can, and you don't believe that they're a *VIP*, or that you're supposed to be *shelving* books
- Then:
  - *adopt* the intention to *xerox* it by *5pm*, and
  - *inform* them of your newly adopted intention.

A PLACA Mental Change Rule

```
(((self ?agent REQUEST (?t (xeroxed ?x)))
 (AND (CAN-ACHIEVE (?t xeroxed ?x)))
      (NOT (BEL (*now* shelving)))
      (NOT (BEL (*now* (vip ?agent))))
 ((ADOPT (INTEND (5pm (xeroxed ?x)))))
 ((?agent self INFORM
       (*now* (INTEND (5pm (xeroxed ?x)))))))
```

Concurrent MetateM

- Concurrent METATEM is a multi-agent language, developed by Michael Fisher
  - Each agent is programmed by giving it a ***temporal logic*** specification of the behaviour it should exhibit.
    - These specifications are executed directly in order to generate the behaviour of the agent.

- Temporal logic is classical logic augmented by ***modal operators*** for describing how the truth of propositions changes over time.
  - Think of the world as being a number of discrete states.
  - There is a single past history, but a number of possible futures
    - all the possible ways that the world might develop.

MetateM Agents

- A Concurrent MetateM system contains a number of agents (objects)
  - Each object has 3 attributes:
    - a ***name***
    - an ***interface***
    - a ***MetateM program***
  - An agent's interface contains two sets:
    - messages the agent will accept;
    - messages the agent may send.

For example, a 'stack' object's interface:

stack(pop, push)[popped, stackfull]

{pop, push} = ***messages received***

{popped, stackfull} = ***messages sent***

MetateM

- The root of the MetateM concept is Gabbay's separation theorem:
  - Any arbitrary temporal logic formula can be rewritten in a logically equivalent past $\Rightarrow$ future form.

- Execution proceeds by a process of continually matching rules against a "history", and firing those rules whose antecedents are satisfied.
  - The instantiated future-time consequents become commitments which must subsequently be satisfied.

Examples

$\Box\,\text{important}(\text{agents})$	*means "it is now, and will always be true that agents are important"*
$\Diamond\,\text{important}(\text{ConcurrentMetateM})$	*means "sometime in the future, ConcurrentMetateM will be important"*
$◈\ \text{important}(\text{Prolog})$	*means "sometime in the past it was true that Prolog was important"*
$(\neg\text{friends}(\text{us}))\,\mathcal{U}\,\text{apologise}(\text{you})$	*means "we are not friends until you apologise"*
$\bigcirc\,\text{apologise}(\text{you})$	*means "tomorrow (in the next state), you apologise"*
$◉\ \text{apologise}(\text{you}) \Rightarrow \bigcirc\,\text{friends}(\text{us})$	*means "if you apologised yesterday, then tomorrow we will be friends"*
$\text{friends}(\text{us})\,\mathcal{S}\,\text{apologise}(\text{you})$	*means "we have been friends since you apologised"*

Summary

- This chapter has focussed on Agent Architectures and general approaches to programming an agent.
  - We defined the notion of symbolic reasoning agents, and discussed...
    - ...how can deductive reasoning be achieved through the use of logic; and
    - ...the Transduction and Representation Problems
  - We introduced the concept of Agent Oriented Programming, and looked at examples of AOP languages, including:
    - Agent0 and PLACA
    - Concurrent MetateM and temporal logic

- In the next chapter, we will consider the merits of practical reasoning agents.

Class Reading (Chapter 3):

"Agent Oriented Programming", Yoav Shoham. Artificial Intelligence Journal 60(1), March 1993. pp51-92.

This paper introduced agent-oriented programming and throughout the late 90ies was one of the most cited articles in the agent community. One of the main points was the notion of using mental states, and introduced the programming language Agent0.