

Intelligent Agents

Chapter 2

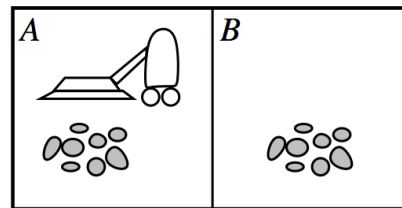
Agent functions and programs

- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is rational
- Aim: find a way to implement the rational agent function concisely

Table-lookup agent

- \input{algorithms/table-agent-algorithm}
- Drawbacks:
 - Huge table
 - Take a long time to build the table
 - No autonomy
 - Even with learning, need a long time to learn the table entries

Vacuum Agent Environment



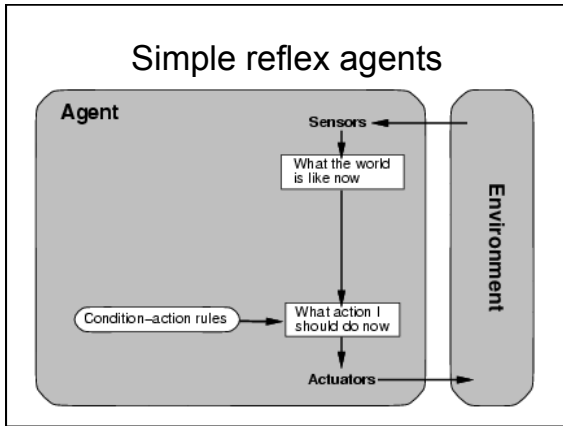
Reflex Vacuum agent program

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

Review of Some Agent types

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

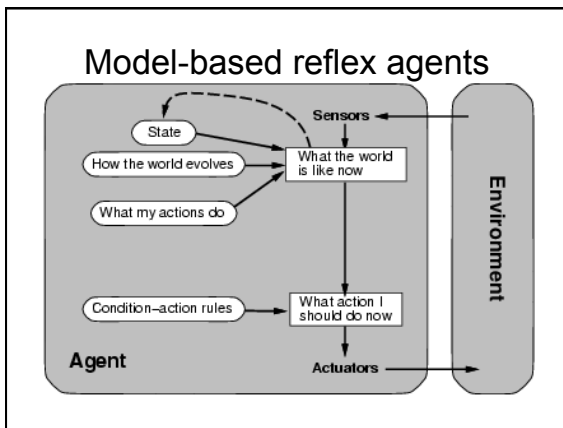


Simple reflex agents

```
function REFLEX-VACUUM-AGENT(location,status) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left

(setq joe (make-agent :name 'joe :body (make-agent-body)
                    :program (make-reflex-vacuum-agent-program)))

(defun make-reflex-vacuum-agent-program ()
  #'(lambda (percept)
      (let ((location (first percept)) (status (second percept)))
        (cond ((eq status 'dirty) 'Suck)
              ((eq location 'A) 'Right)
              ((eq location 'B) 'Left))))))
```



Model-based reflex agents

```
function REFLEX-VACUUM-AGENT(location,status) returns an action
  static: last-A, last-B, numbers, initially ∞
  if status = Dirty then ...

(defun make-reflex-vacuum-agent-with-state-program ()
  (let ((last-A infinity) (last-B infinity))
    #'(lambda (percept)
        (let ((location (first percept)) (status (second percept)))
          (incf last-A) (incf last-B)
          (cond ((eq status 'dirty)
                (if (eq location 'A) (setf last-A 0) (setf last-B 0))
                  'Suck)
                ((eq location 'A) (if (> last-B 3) 'Right 'NoOp))
                ((eq location 'B) (if (> last-A 3) 'Left 'NoOp))))))
```

