MAKING COMPLEX DECISIONS

function VALUE-ITERATION\(mdp, \epsilon\) returns a utility function

inputs: \(mdp\), an MDP with states \(S\), actions \(A(s)\), transition model \(P(s' | s, a)\), rewards \(R(s)\), discount \(\gamma\)
\(\epsilon\), the maximum error allowed in the utility of any state

local variables: \(U, U'\), vectors of utilities for states in \(S\), initially zero
\(\delta\), the maximum change in the utility of any state in an iteration

repeat
\(U \leftarrow U'\); \(\delta \leftarrow 0\)
for each state \(s\) in \(S\) do
\(U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U'[s']\)
if \(|U'[s] - U[s]| > \delta\) then \(\delta \leftarrow |U'[s] - U[s]|\)
until \(\delta < \epsilon(1 - \gamma)/\gamma\)
return \(U\)

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (??).
function `POLICY-ITERATION(mdp)` returns a policy
inputs: `mdp`, an MDP with states `S`, actions `A(s)`, transition model `P(s' | s, a)`
local variables: `U`, a vector of utilities for states in `S`, initially zero
                      `π`, a policy vector indexed by state, initially random
repeat
  `U ← POLICY-EVALUATION(π, U, mdp)`
  ` unchanged? ← true`
  for each state `s` in `S` do
    if \[ \max_{a \in A(s)} \sum_{s'} P(s' | s, a) \ U[s'] > \sum_{s'} P(s' | s, π[s]) \ U[s'] \] then do
      `π[s] ← \arg\max_{a \in A(s)} \sum_{s'} P(s' | s, a) \ U[s']`
      ` unchanged? ← false`
  until ` unchanged?`
return `π`

Figure 17.7  The policy iteration algorithm for calculating an optimal policy.

function `POMDP-VALUE-ITERATION(pomdp, ε)` returns a utility function
inputs: `pomdp`, a POMDP with states `S`, actions `A(s)`, transition model `P(s' | s, a)`,
          sensor model `P(e | s)`, rewards `R(s)`, discount `γ`
ε, the maximum error allowed in the utility of any state
local variables: `U`, `U'`, sets of plans `p` with associated utility vectors `α_p`

`U' ←` a set containing just the empty plan `[]`, with `α_{[]} (s) = R(s)`
repeat
  `U ← U'`
  `U' ←` the set of all plans consisting of an action and, for each possible next percept,
          a plan in `U` with utility vectors computed according to Equation (??)
  `U' ← REMOVE-DOMINATED-PLANS(U')`
until \( \text{MAX-DIFFERENCE}(U, U') < \epsilon(1 - \gamma)/\gamma \)
return `U`

Figure 17.9  A high-level sketch of the value iteration algorithm for POMDPs. The
`REMOVE-DOMINATED-PLANS` step and `MAX-DIFFERENCE` test are typically implemented as linear programs.