function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s,a)$,
    sensor model $P(e \mid s)$, rewards $R(s,a,s')$, 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

  $U' \leftarrow$ a set containing all one-step plans $[a]$, with $\alpha_{[a]}(s) = \sum_{s'} P(s' \mid s,a)\, R(s,a,s')$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ 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 (17.18)
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $\leq \epsilon(1-\gamma)/\gamma$
  return U

Figure 17.16 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.

Given such a utility function, an executable policy can be extracted by looking at which hyperplane is optimal at any given belief state b and executing the first action of the corresponding plan. In Figure 17.15(d), the corresponding optimal policy is still the same as for depth-1 plans: *Stay* when $b(B) > 0.5$ and *Go* otherwise.

In practice, the value iteration algorithm in Figure 17.16 is hopelessly inefficient for larger problems—even the 4×3 POMDP is too hard. The main reason is that given n undominated conditional plans at level d, the algorithm constructs $|A| \cdot n^{|E|}$ conditional plans at level $d+1$ before eliminating the dominated ones. With the four-bit sensor, $|E|$ is 16, and n can be in the hundreds, so this is hopeless.

Since this algorithm was developed in the 1970s, there have been several advances, including more efficient forms of value iteration and various kinds of policy iteration algorithms. Some of these are discussed in the notes at the end of the chapter. For general POMDPs, however, finding optimal policies is very difficult (PSPACE-hard, in fact—that is, very hard indeed). The next section describes a different, approximate method for solving POMDPs, one based on look-ahead search.

17.5.2 Online algorithms for POMDPs

The basic design for an online POMDP agent is straightforward: it starts with some prior belief state; it chooses an action based on some deliberation process centered on its current belief state; after acting, it receives an observation and updates its belief state using a filtering algorithm; and the process repeats.

One obvious choice for the deliberation process is the expectimax algorithm from Section 17.2.4, except with belief states rather than physical states as the decision nodes in the tree. The chance nodes in the POMDP tree have branches labeled by possible observations and leading to the next belief state, with transition probabilities given by Equation (17.17). A fragment of the belief-state expectimax tree for the 4×3 POMDP is shown in Figure 17.17.