```
function ANGELIC-SEARCH(problem, hierarchy, initialPlan) returns a solution or fail
  frontier ← a FIFO queue with initialPlan as the only element
  while true do
    if IS-EMPTY?(frontier) then return fail
    plan ← POP(frontier)        // chooses the shallowest node in frontier
    if REACH⁺(problem.INITIAL, plan) intersects problem.GOAL then
      if plan is primitive then return plan        // REACH⁺ is exact for primitive plans
      guaranteed ← REACH⁻(problem.INITIAL, plan) ∩ problem.GOAL
      if guaranteed≠{ } and MAKING-PROGRESS(plan, initialPlan) then
        finalState ← any element of guaranteed
        return DECOMPOSE(hierarchy, problem.INITIAL, plan, finalState)
      hla ← some HLA in plan
      prefix,suffix ← the action subsequences before and after hla in plan
      outcome ← RESULT(problem.INITIAL, prefix)
      for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        add APPEND(prefix, sequence, suffix) to frontier

function DECOMPOSE(hierarchy, s_0, plan, s_f) returns a solution
  solution ← an empty plan
  while plan is not empty do
    action ← REMOVE-LAST(plan)
    s_i ← a state in REACH⁻(s_0, plan) such that s_f ∈ REACH⁻(s_i, action)
    problem ← a problem with INITIAL = s_i and GOAL = s_f
    solution ← APPEND(ANGELIC-SEARCH(problem, hierarchy, action), solution)
    s_f ← s_i
  return solution
```

Figure 11.11 A hierarchical planning algorithm that uses angelic semantics to identify and commit to high-level plans that work while avoiding high-level plans that don't. The predicate MAKING-PROGRESS checks to make sure that we aren't stuck in an infinite regression of refinements. At top level, call ANGELIC-SEARCH with $[Act]$ as the *initialPlan*.

The ability to commit to or reject high-level plans can give ANGELIC-SEARCH a significant computational advantage over HIERARCHICAL-SEARCH, which in turn may have a large advantage over plain old BREADTH-FIRST-SEARCH. Consider, for example, cleaning up a large vacuum world consisting of an arrangement of rooms connected by narrow corridors, where each room is a $w \times h$ rectangle of squares. It makes sense to have an HLA for *Navigate* (as shown in Figure 11.7) and one for *CleanWholeRoom*. (Cleaning the room could be implemented with the repeated application of another HLA to clean each row.) Since there are five primitive actions, the cost for BREADTH-FIRST-SEARCH grows as 5^d, where d is the length of the shortest solution (roughly twice the total number of squares); the algorithm cannot manage even two 3×3 rooms. HIERARCHICAL-SEARCH is more efficient, but still suffers from exponential growth because it tries all ways of cleaning that are consistent with the hierarchy. ANGELIC-SEARCH scales approximately linearly in the number of squares—it commits to a good high-level sequence of room-cleaning and navigation steps and prunes away the other options.