```
function MINIMAX-SEARCH(game, state) returns an action
   player ← game.TO-MOVE(state)
   value, move ← MAX-VALUE(game, state)
   return move

function MAX-VALUE(game, state) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow -\infty$
   for each a in game.ACTIONS(state) do
      v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
      if $v2 > v$ then
         v, move ← v2, a
   return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow +\infty$
   for each a in game.ACTIONS(state) do
      v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
      if $v2 < v$ then
         v, move ← v2, a
   return v, move
```

Figure 5.3 An algorithm for calculating the optimal move using minimax—the move that leads to a terminal state with maximum utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state and the move to get there.

```
function ALPHA-BETA-SEARCH(game, state) returns an action
   player ← game.TO-MOVE(state)
   value, move ← MAX-VALUE(game, state, $-\infty$, $+\infty$)
   return move

function MAX-VALUE(game, state, $\alpha$, $\beta$) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow -\infty$
   for each a in game.ACTIONS(state) do
      v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), $\alpha$, $\beta$)
      if $v2 > v$ then
         v, move ← v2, a
         $\alpha \leftarrow$ MAX($\alpha$, v)
      if $v \geq \beta$ then return v, move
   return v, move

function MIN-VALUE(game, state, $\alpha$, $\beta$) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   $v \leftarrow +\infty$
   for each a in game.ACTIONS(state) do
      v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), $\alpha$, $\beta$)
      if $v2 < v$ then
         v, move ← v2, a
         $\beta \leftarrow$ MIN($\beta$, v)
      if $v \leq \alpha$ then return v, move
   return v, move
```

Figure 5.7 The alpha–beta search algorithm. Notice that these functions are the same as the MINIMAX-SEARCH functions in Figure ??, except that we maintain bounds in the variables α and β, and use them to cut off search when a value is outside the bounds.