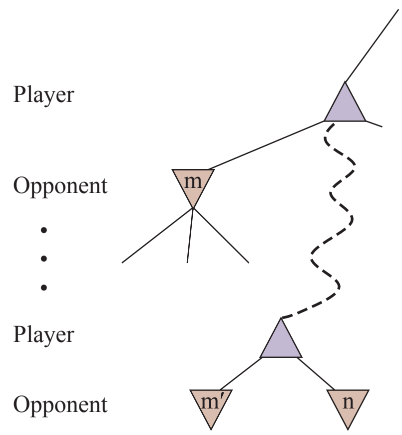

Figure 5.6 The general case for alpha–beta pruning. If m or m' is better than n for Player, we will never get to n in play.

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

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   v, move ← −∞, null
   for each a in game.ACTIONS(state) do
      v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
      if v2 > v then
         v, move ← v2, a
         α ← MAX(α, v)
      if v ≥ β then return v, move
   return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
   if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
   v, move ← +∞, null
   for each a in game.ACTIONS(state) do
      v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
      if v2 < v then
         v, move ← v2, a
         β ← MIN(β, v)
      if v ≤ α 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 5.3, 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.