Chapter 5

Adversarial Search

Adversarial search

- Search problems seen so far:
  - Single agent.
  - No interference from other agents and no competition.
- Game playing: Multi-agent environment.
  1. Cooperative games.
  2. Competitive games → adversarial search or games.

Types of games

	Deterministic	Chance
Perfect Information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect Information	Battleship	Bridge, Poker, Scrabble,

Adversarial search vs. search problems

- Unlike classic search problems, the outcome of the game depends not only on the action of the agent but also on the actions of the other agent(s).
- The action of the opponent agent are not known in advance → we need specify a move for every possible opponent reply.
- Time limits: game playing is limited by time → we need to approximate and <u>not</u> search for an optimal solution.

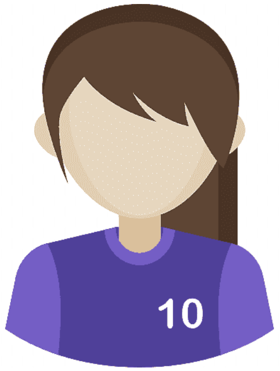
Planning ahead in a world that includes a hostile agent

- Games as search problems
- Idealization and simplification:
  - Two players
  - Alternate moves
    - MAX player
    - MIN player
  - Available information:
    - Perfect: chess, chequers, tic-tac-toe… (no chance, same knowledge for the two players)
    - Imperfect: poker, Stratego, bridge…

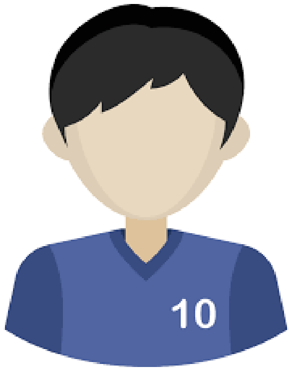
Setup

- At the end of the game, points are awarded to the winning player and penalties are given to the loser.
- We consider two-player games:
  - One agent tries to maximize the utility function, the other tries to minimize it.
  - As a convention our protagonist agent is a maximizer.

MAX



MIN



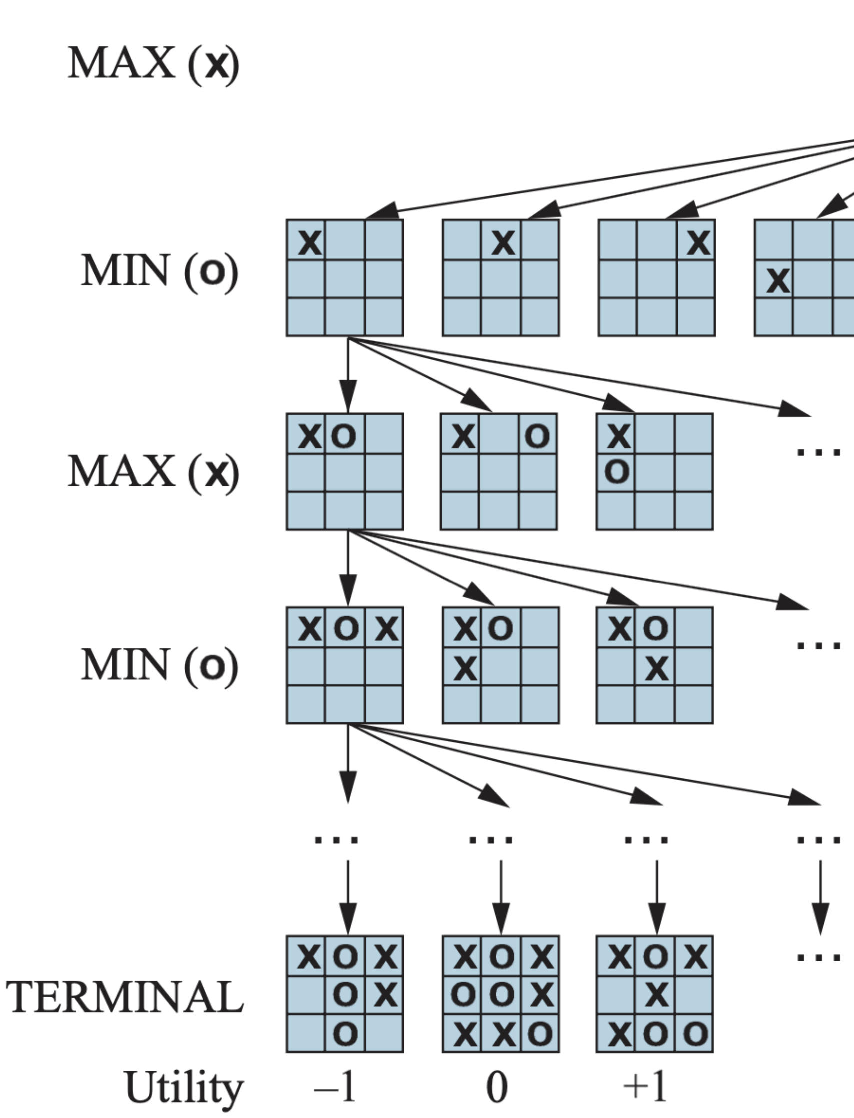
Problem Formulation

- Initial state: initial configuration of the game.
  - Example: initial board configuration in chess.
- Player: Defines which player has the turn in the state.
- Actions: set of legal moves from a state.
- Result: The **transition model**, which defines the result of a move
- Terminal test: decide if the game has finished.
- Utility function: produces a numerical value for (only) the terminal states.
    - Example: In chess, outcome = win/loss/draw, with values +1, 0, ½ respectively.

Game representation

- In the general case of a game with two players:
  - General state representation
  - Initial-state definition
  - Winning-state representation as:
    - Structure
    - Properties
    - Utility function
  - Definition of a set of operators

Example: game tree for tic-tac-toe



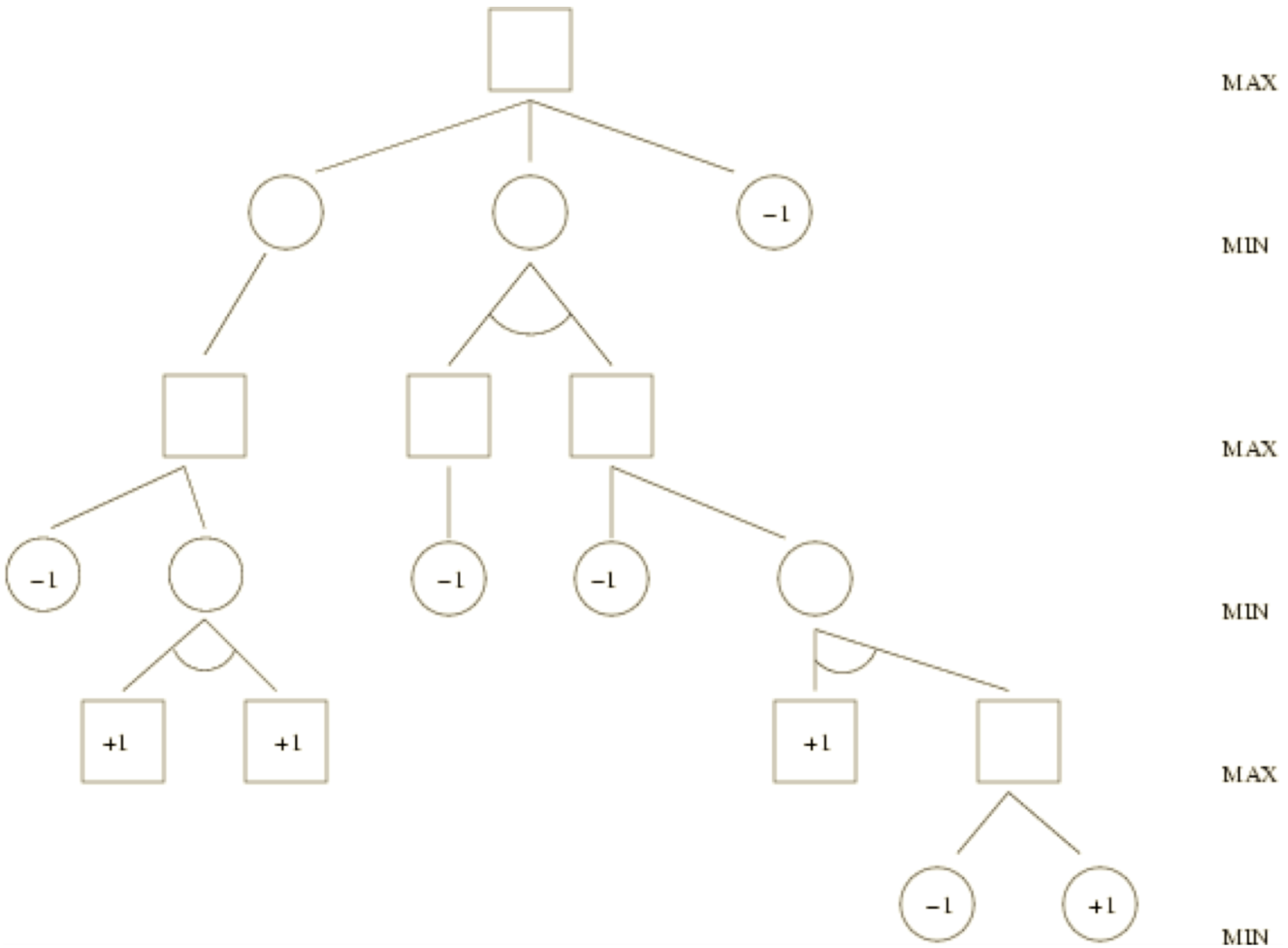
- 9! combinations, but actually less because of repetition
- MAX needs a **strategy** to move down the tree
  - What to do for every move MIN makes

- Is tic-tac-toe zero-sum?

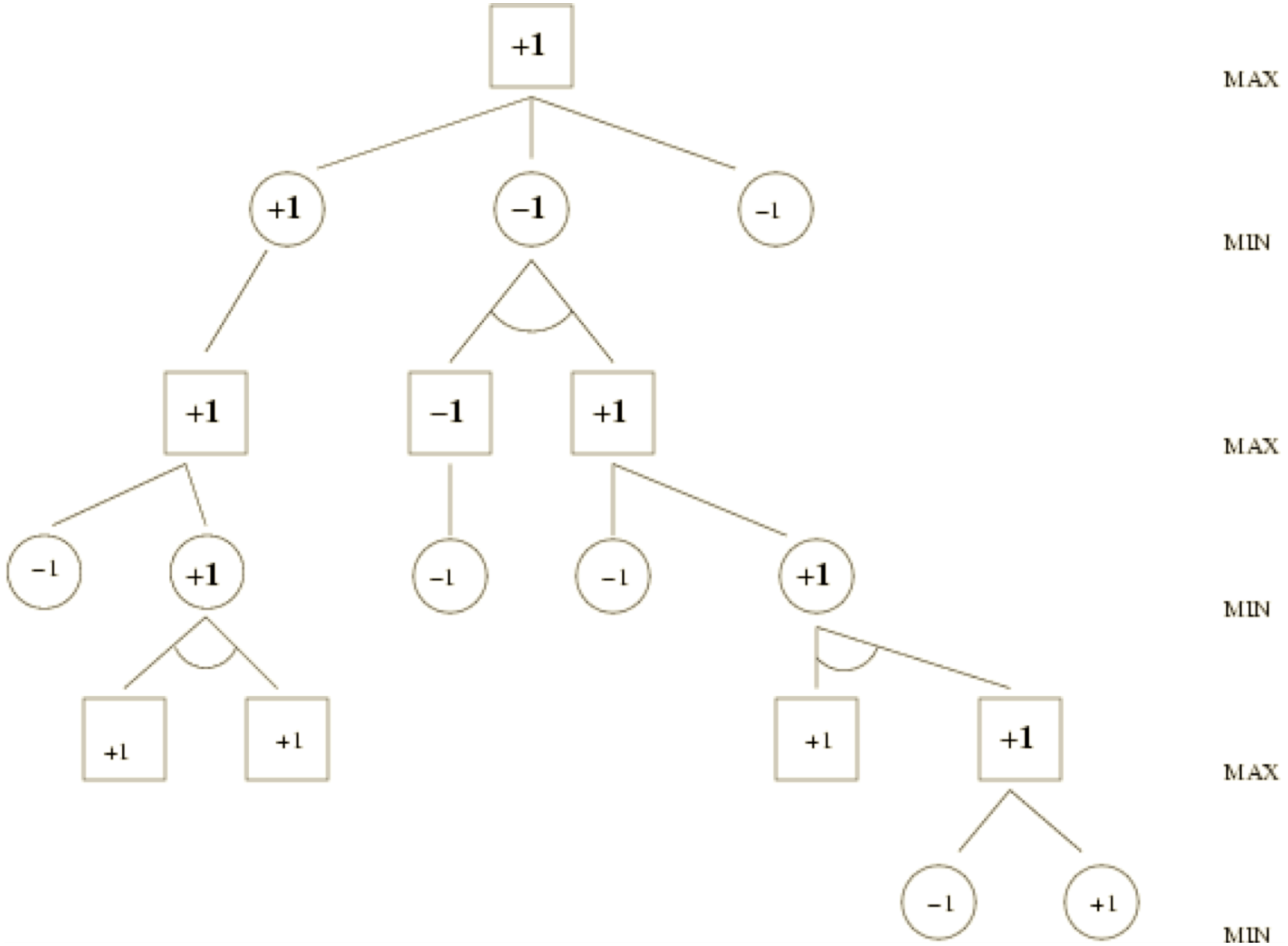
Search with an opponent

- **Trivial approximation**: generating the tree for all moves
- Terminal moves are tagged with a utility value, for example: “+1” or “-1” depending on if the winner is MAX or MIN.
- The goal is to find a path to a winning state.
- Even if a depth-first search would minimize memory space, in complex games this kind of search cannot be carried out.
- Even a simple game like tic-tac-toe is too complex to draw the entire game tree.

Search with an opponent



Search with an opponent

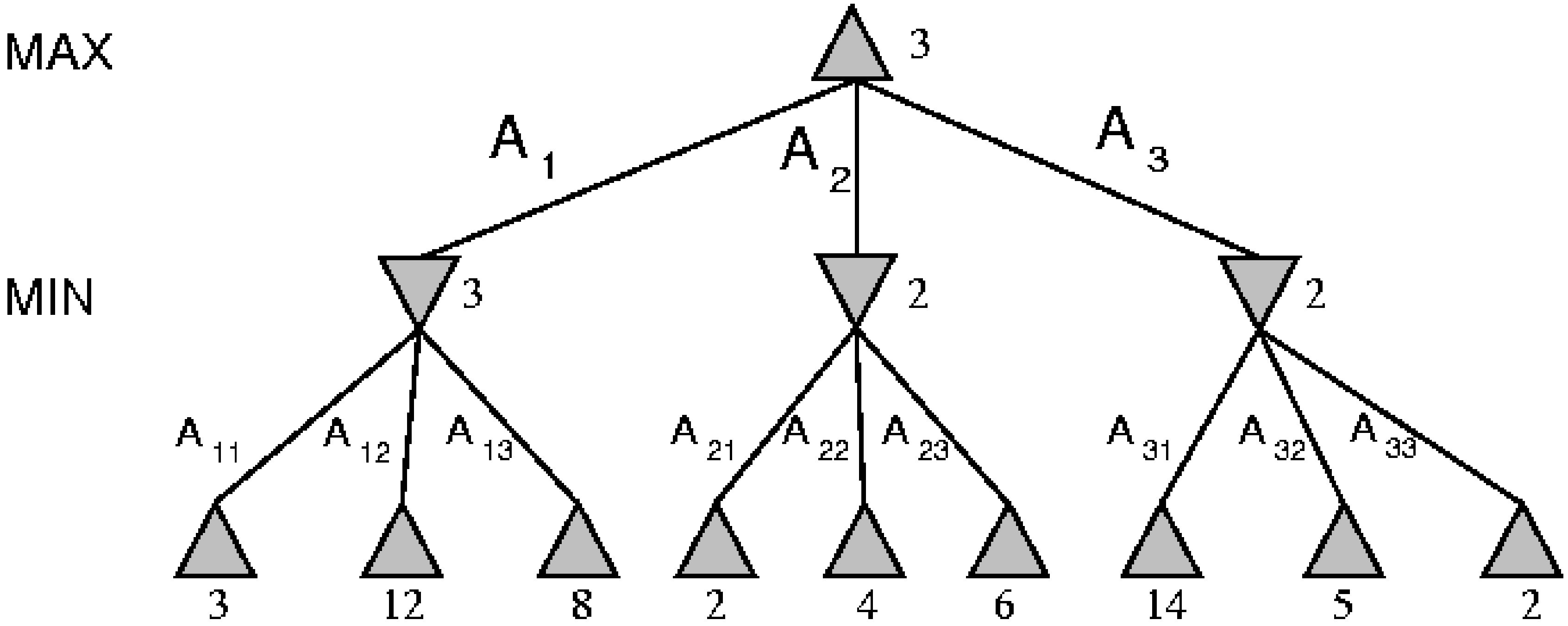


Search with an opponent

- **Heuristic approximation**: defining an **evaluation function** which indicates how close a state is from a winning (or losing) move
- This function includes domain information.
- It does not represent a cost or a distance in steps.
- Conventionally:
    - A winning move is represented by the value“+∞”.
    - A losing move is represented by the value “-∞”.
    - The algorithm searches with limited depth.

- Each new decision implies repeating part of the search.

Minimax

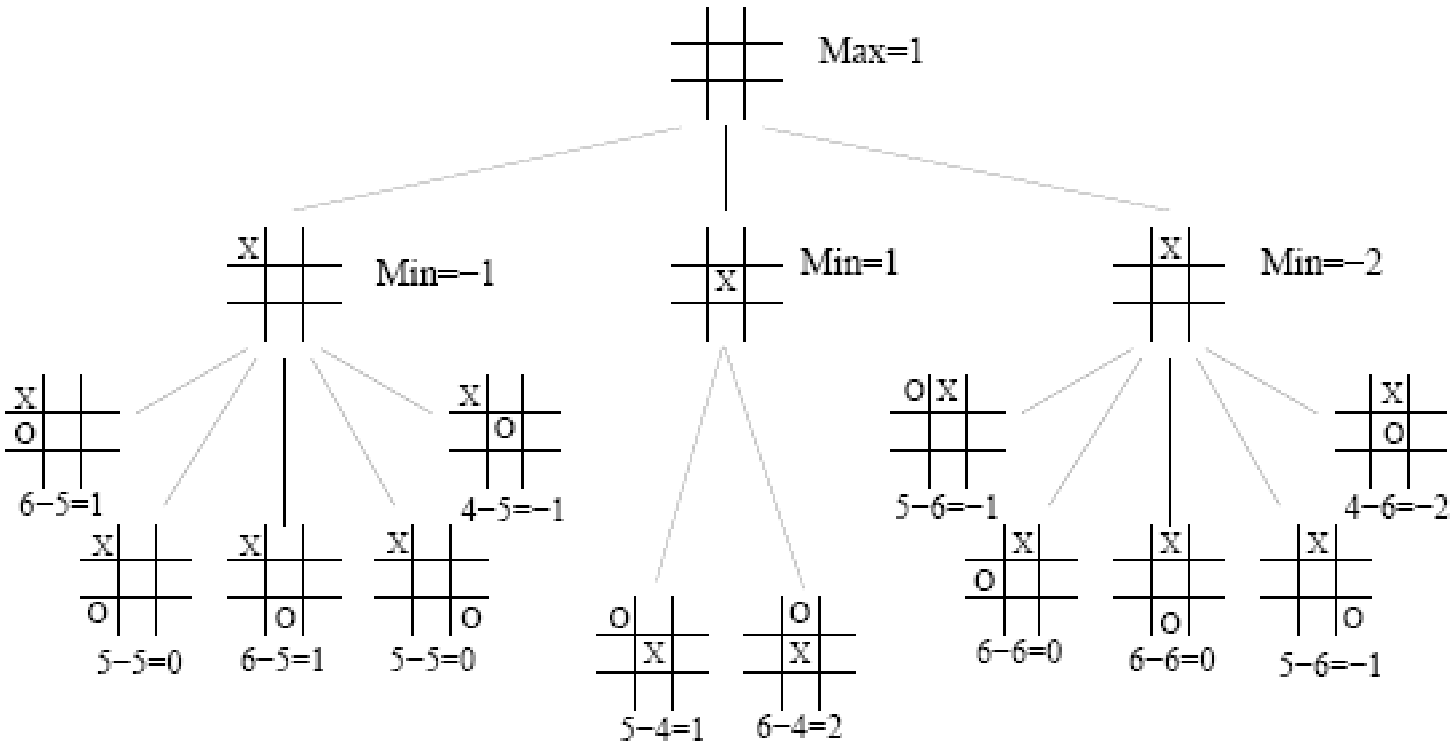


- Minimax-value(*n*): utility for MAX of being in state *n*, assuming both players are playing optimally =
    - Utility(*n*), if *n* is a terminal state
    - $\max_{s \in \text{Successors}(n)}$ Minimax-value(s), if *n* is a MAX state
    - $\min_{s \in \text{Successors}(n)}$ Minimax-value(s), if *n* is a MIN state

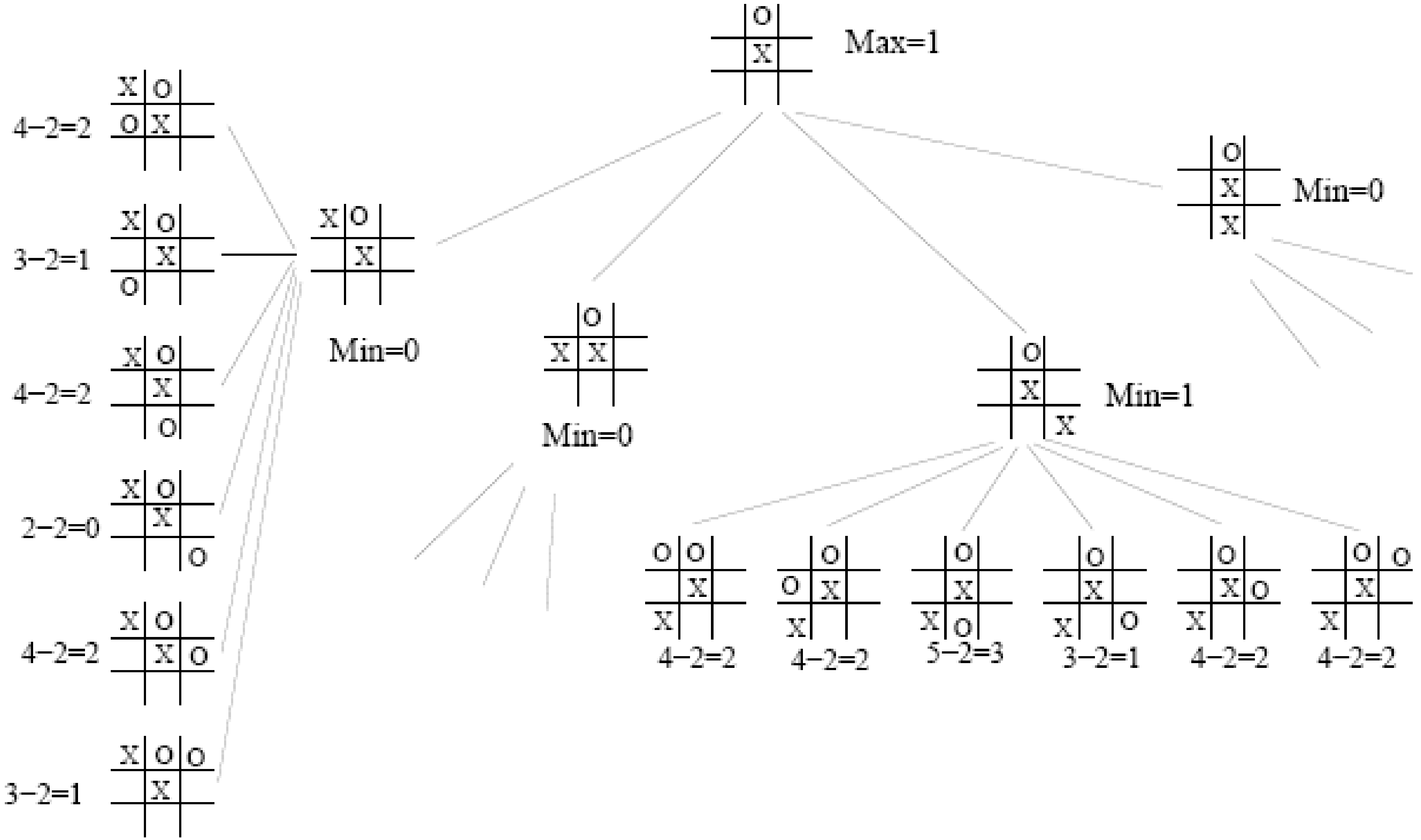
Example: tic-tac-toe

- e (evaluation function → integer) = number of available rows, columns, diagonals for MAX - number of available rows, columns, diagonals for MIN
- MAX plays with “X” and desires maximizing e.
- MIN plays with “0” and desires minimizing e.
- Symmetries are taken into account.
- A depth limit is used (2, in the example).

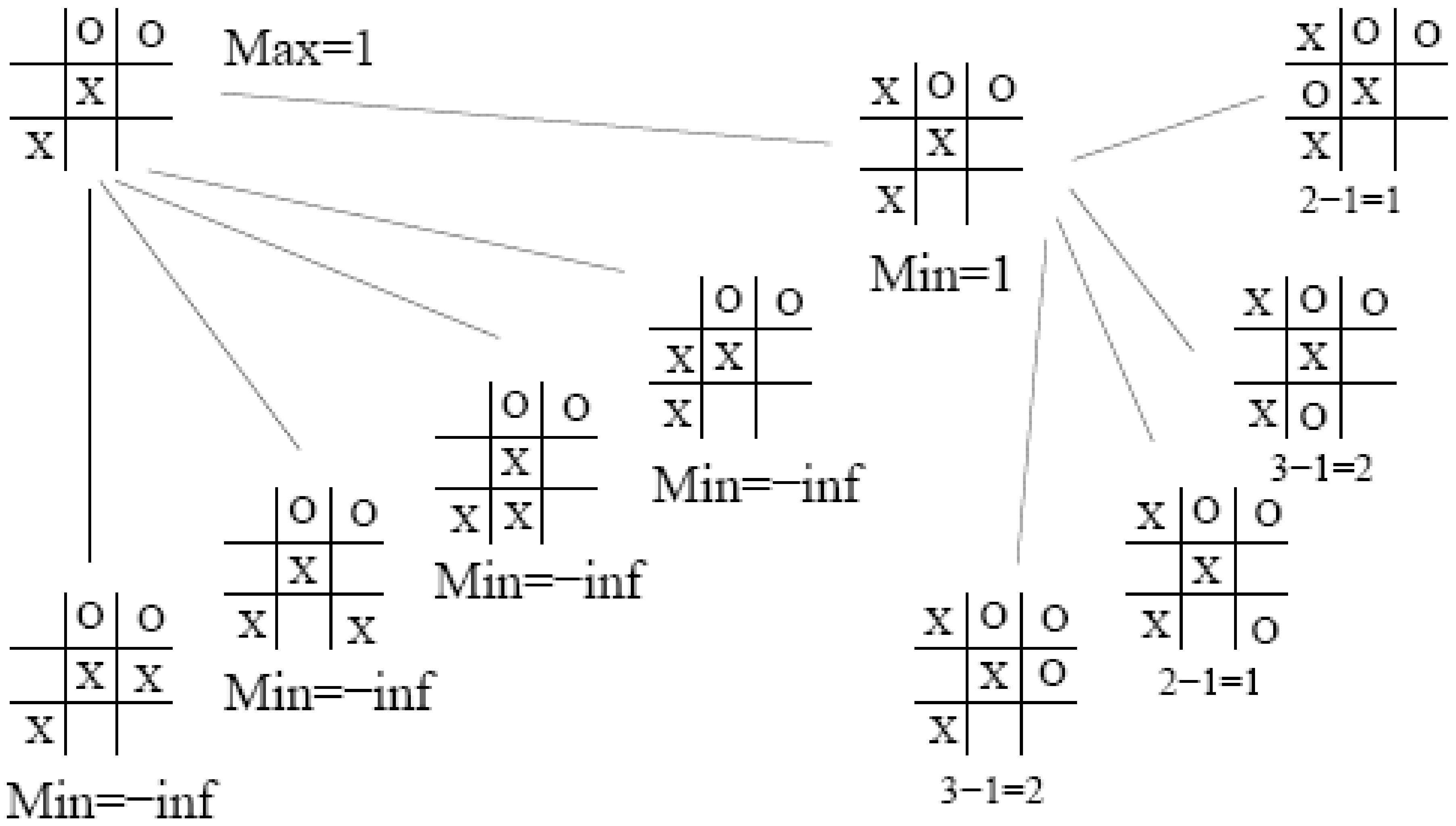
Example: tic-tac-toe



Example: tic-tac-toe



Example: tic-tac-toe



Minimax algorithm

- **Idea**: choose move to position with highest minimax value = best achievable payoff against best play
- This definition of optimal play for MAX assumes that MIN also plays optimally—it maximizes the *worst-case* outcome for MAX
- What if MIN does not play optimally? Then MAX will do even better

The minimax algorithm

- The **minimax algorithm** computes the minimax decision from the current state.
- It uses a simple recursive computation of the minimax values of each successor state:
    - directly implementing the defining equations.
- The recursion proceeds all the way down to the leaves of the tree.
- Then the minimax values are **backed up** through the tree as the recursion unwinds.

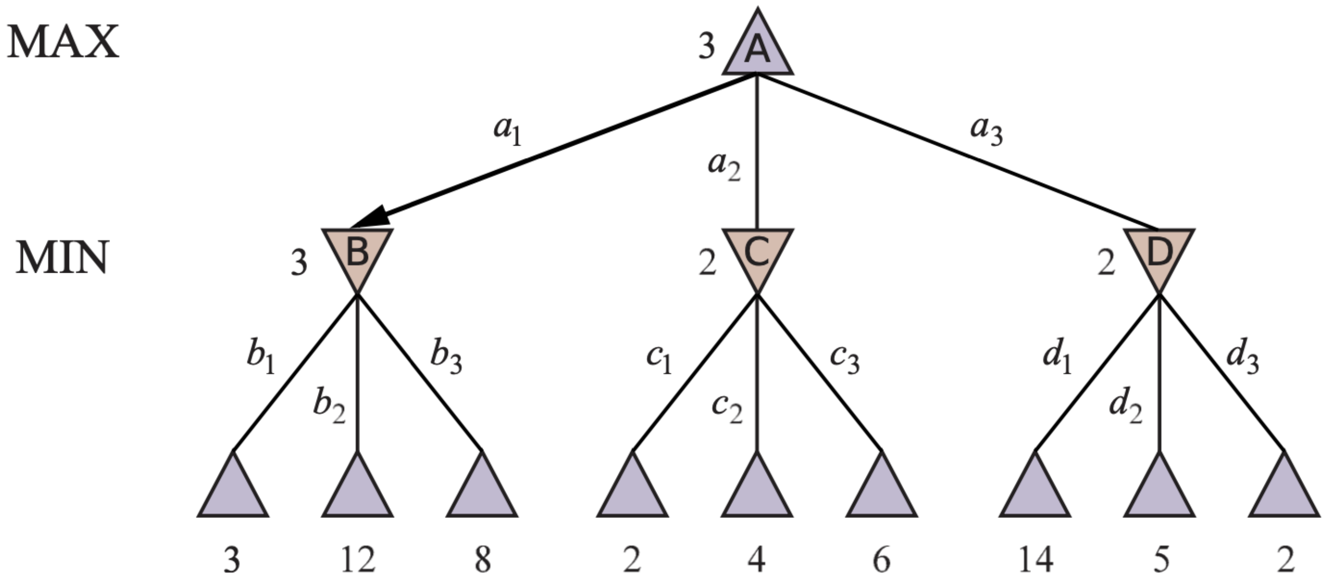
20

Zero-sum games

- **Zero-sum game**: defined as one where the total payoff to all players is the same for every instance of the game.
  - Total payoff $\neq 0$
- Example: Chess is zero-sum because every game has a payoff of either 0 + 1, 1 + 0 or ½ + ½
- "Constant-sum" would have been a better term, but zero-sum is traditional

Simple game tree

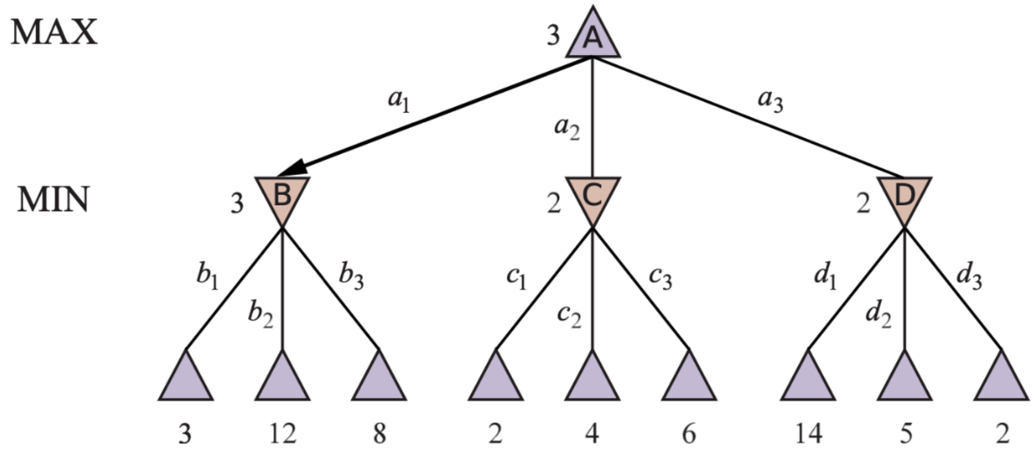
- Given a game tree, the optimal strategy can be determined from the minimax value of each node
- $MINIMAX(n)$: Utility (for MAX) of being in the corresponding state
  - We assume that both players play optimally from there to the end of the game



Simple game tree

$$MINIMAX(s) =$$

$$\begin{cases} Utility(s) & if\ Terminal - Test(s) \\ Max_{a \in Actions(s)} MINIMAX(Result(s,a)) & if\ Player(s) = MAX \\ Min_{a \in Actions(s)} MINIMAX(Result(s,a)) & if\ Platyer(s) = MIN \end{cases}$$



Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

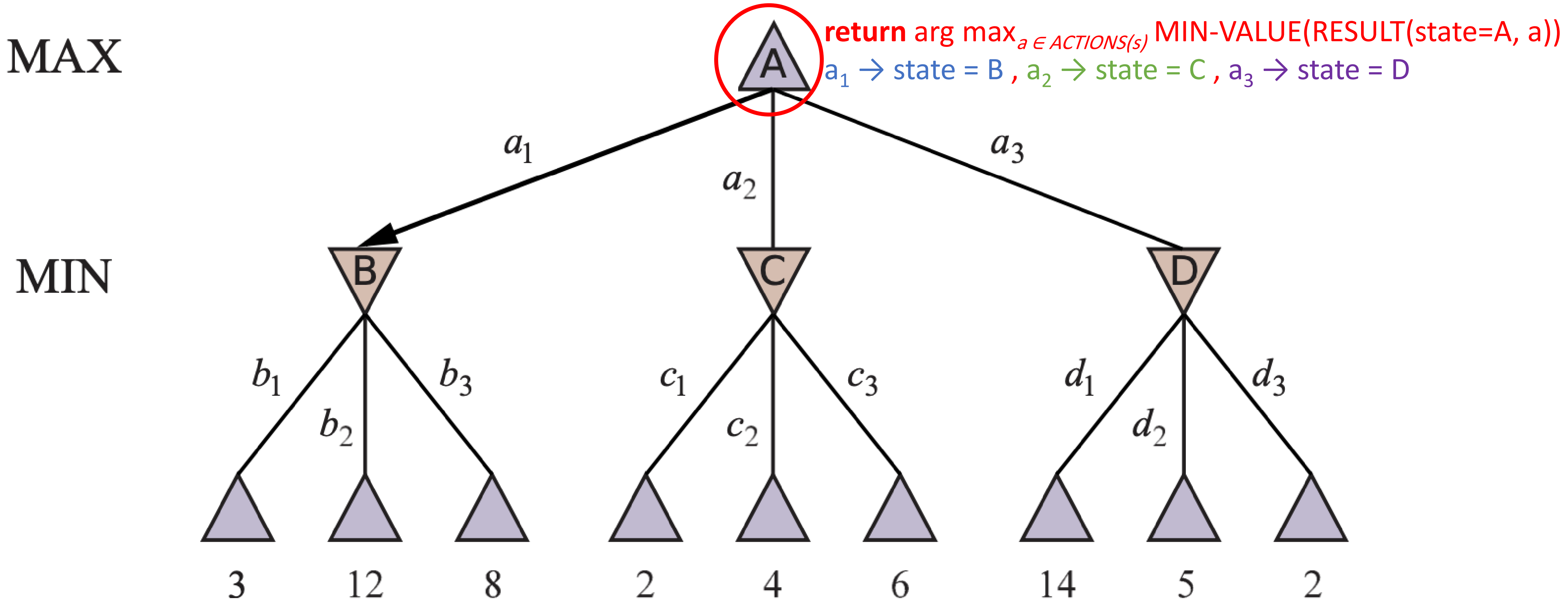
function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*s*, *a*)))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(*s*, *a*)))
  return v

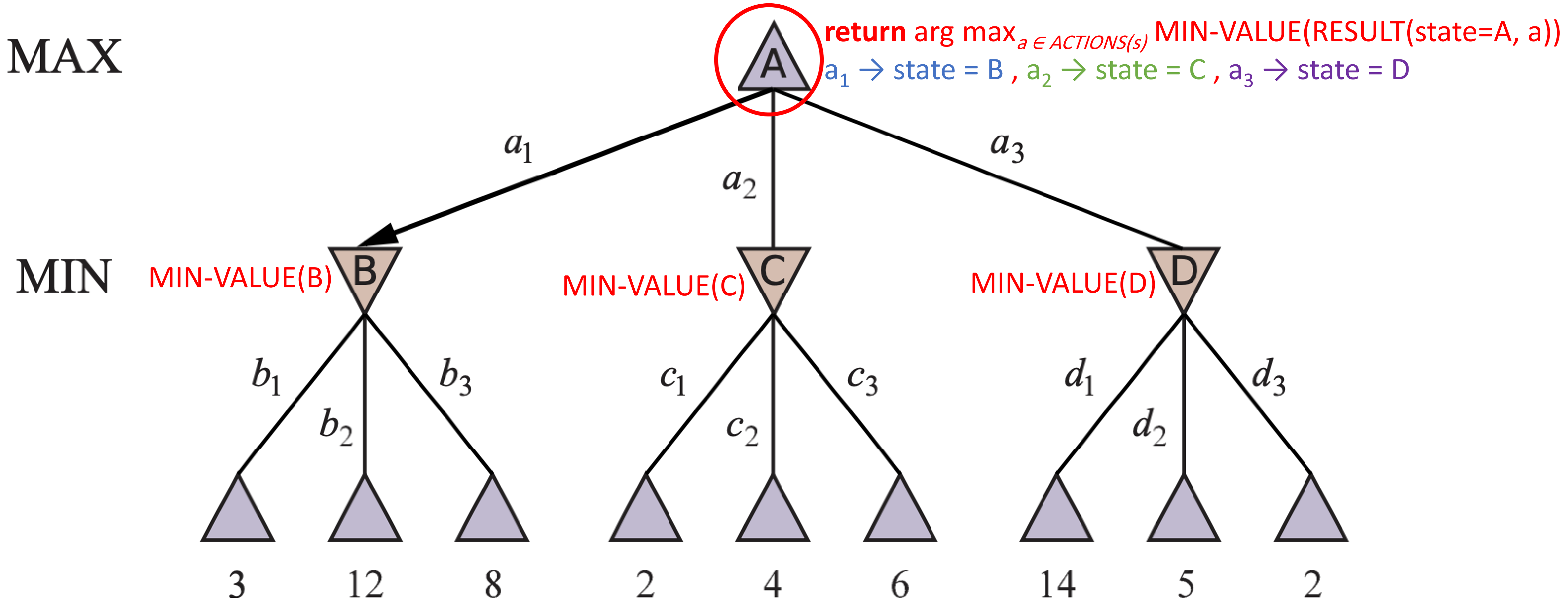
function MINIMAX-DECISION(*state*) **returns** *an action*
return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

Minimax algorithm



function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

Minimax algorithm

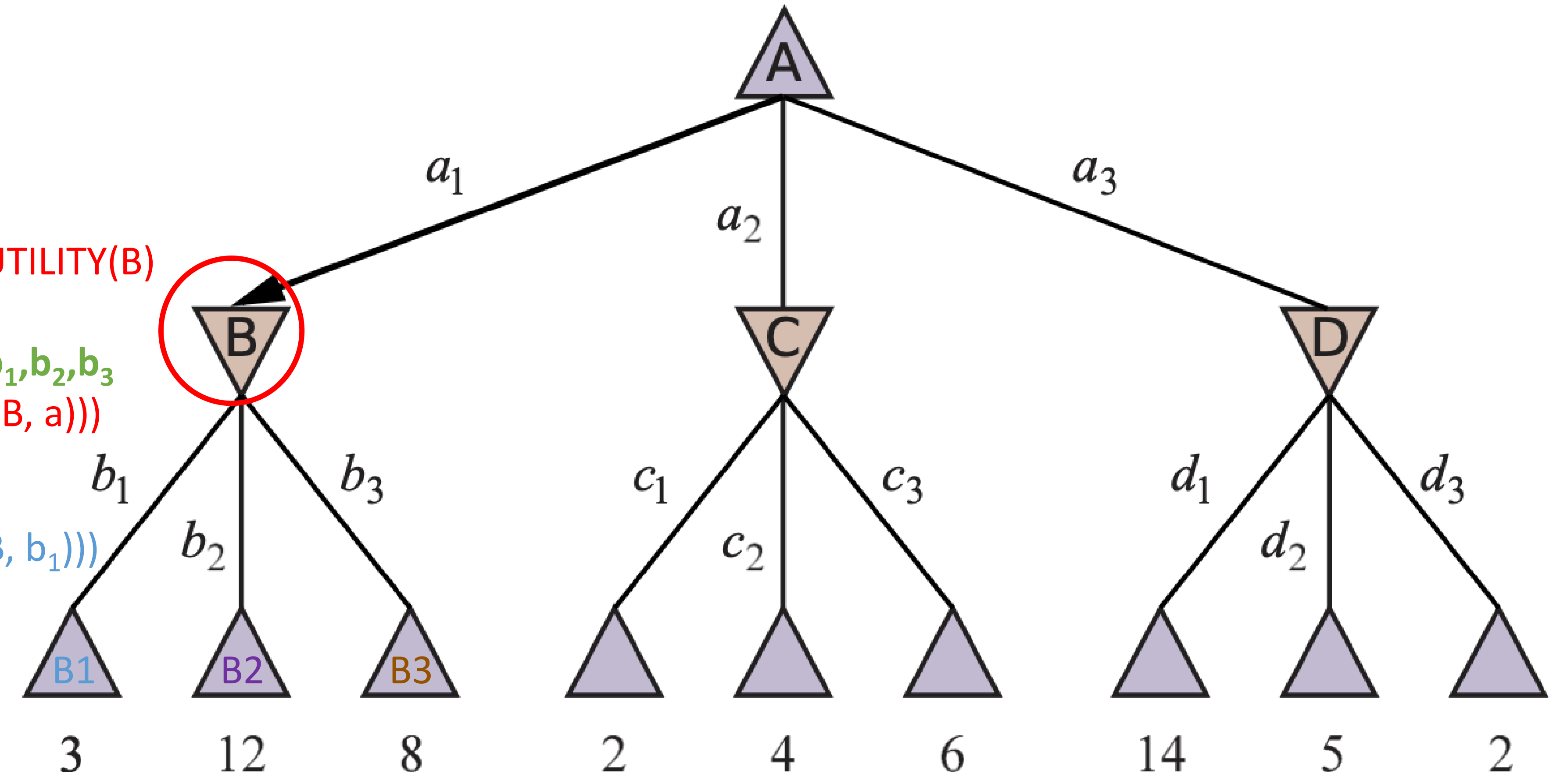


Minimax algorithm

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

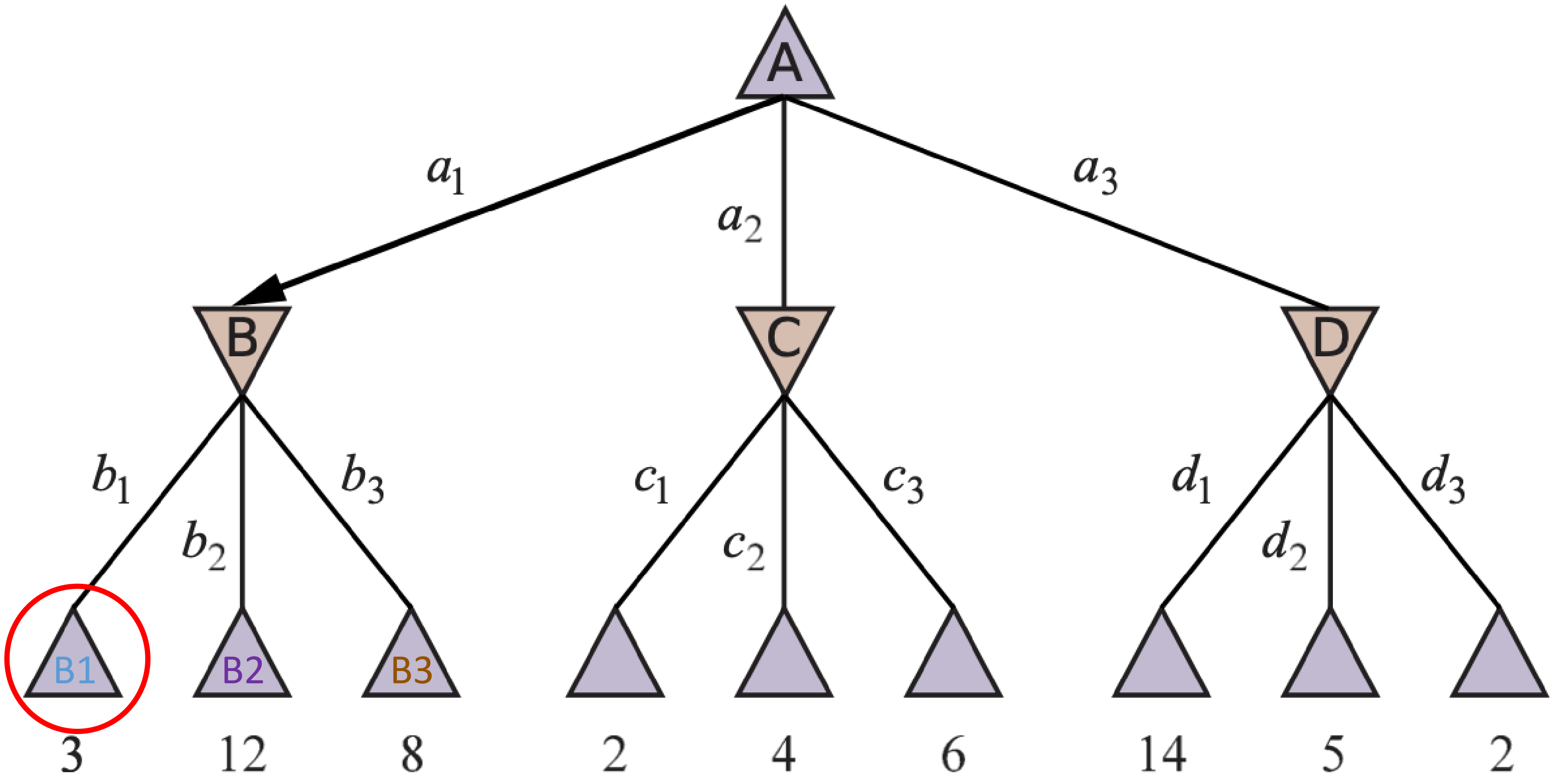
if TERMINAL-TEST(B) **then return** UTILITY(B)
$v \leftarrow \infty$
for each a **in** ACTIONS(B) **do** //a= b_1,b_2,b_3
  $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B, a)))
return v

$v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B, b_1)))
$v \leftarrow$ MIN(v, MAX-VALUE(B1))
$v \leftarrow$ MIN(v, MAX-VALUE(B2))
$v \leftarrow$ MIN(v, MAX-VALUE(B3))



Minimax algorithm

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v



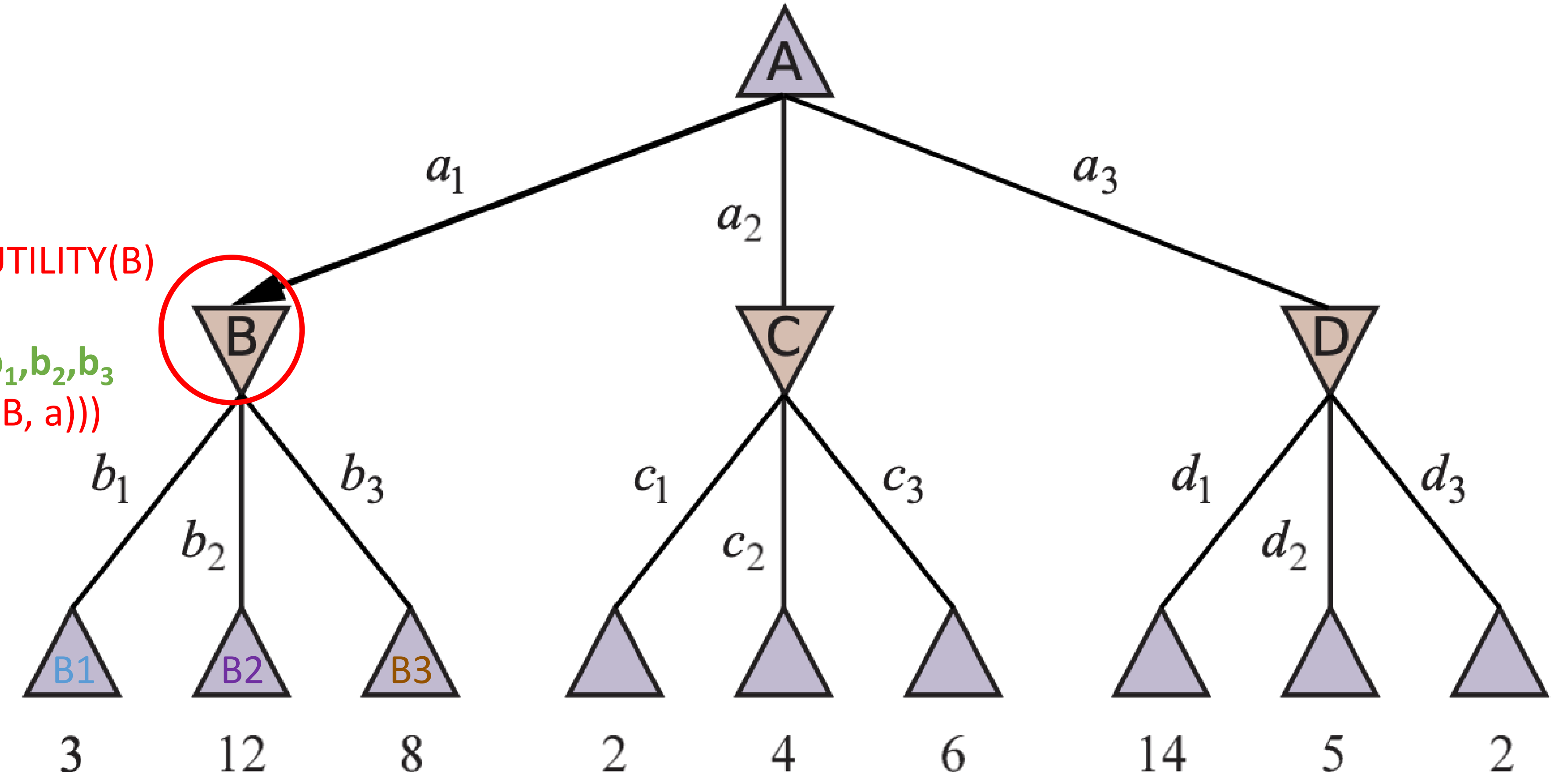
if TERMINAL-TEST(B1) **then**
return UTILITY(B1) = 3

Minimax algorithm

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

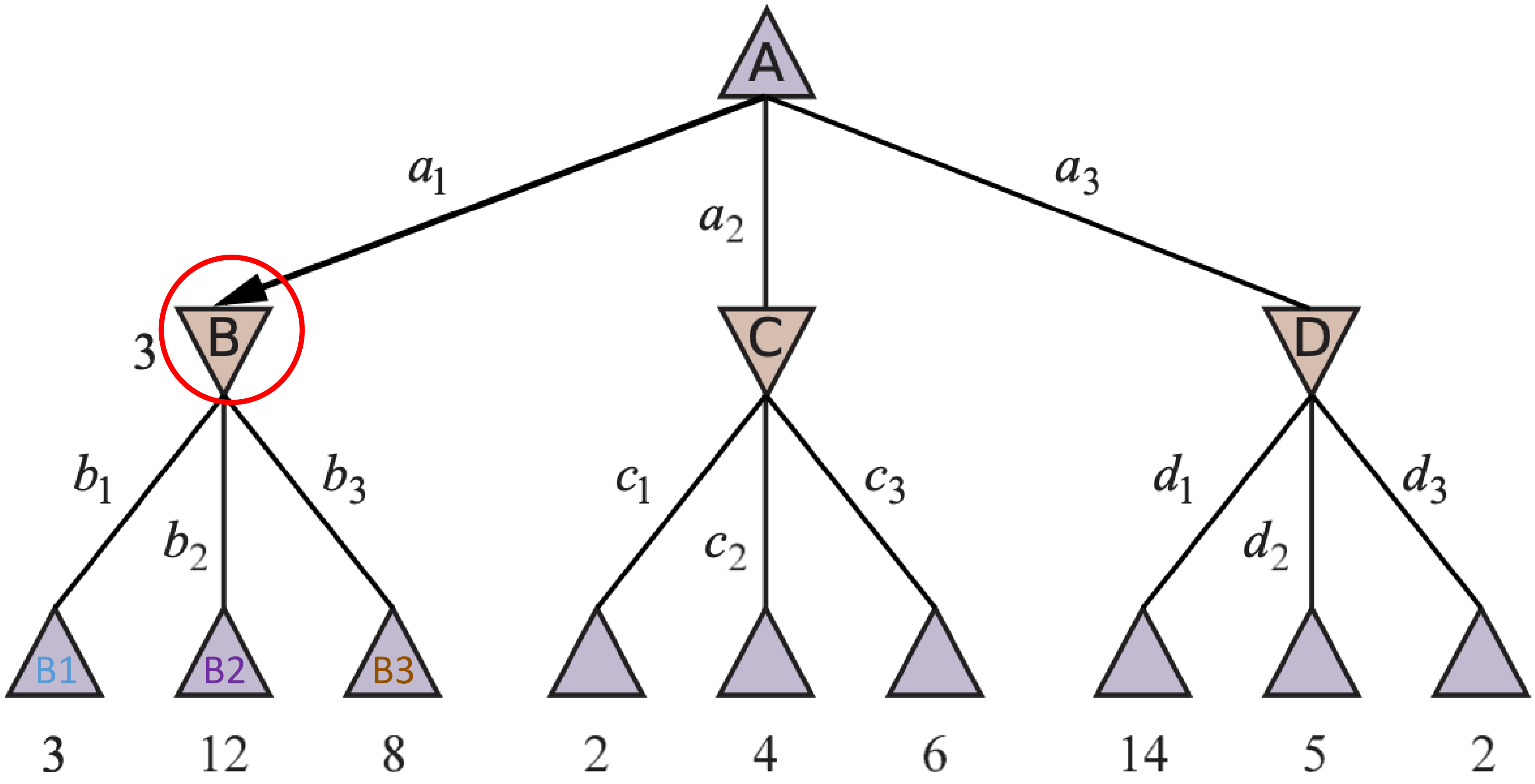
if TERMINAL-TEST(B) **then return** UTILITY(B)
$v \leftarrow \infty$
for each a **in** ACTIONS(B) **do** //a= b_1,b_2,b_3
  $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B, a)))
return v

$v \leftarrow$ MIN(∞, 3)
$v \leftarrow$ MIN(3, 12)
$v \leftarrow$ MIN(3, 8)



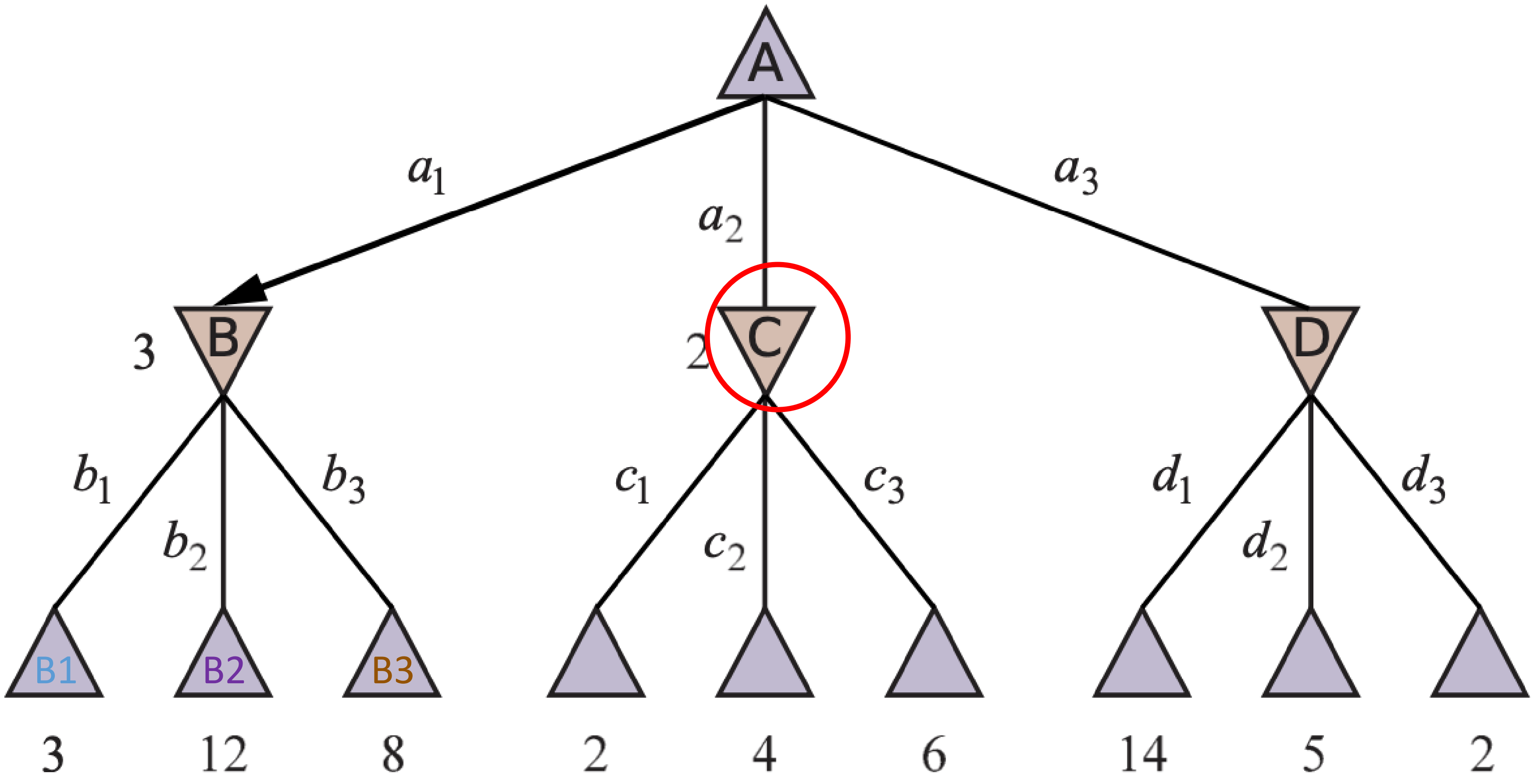
Minimax algorithm

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v



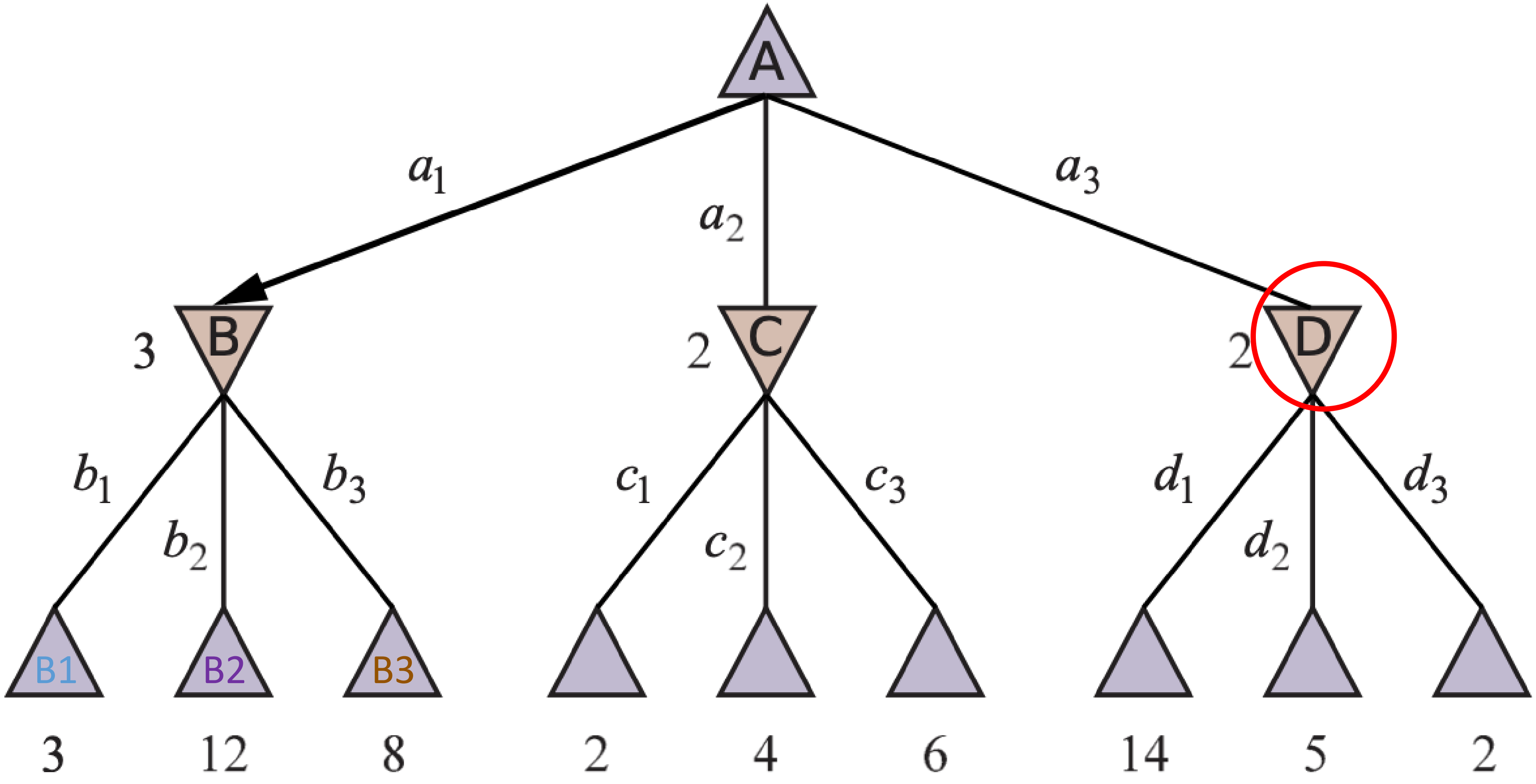
Minimax algorithm

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v



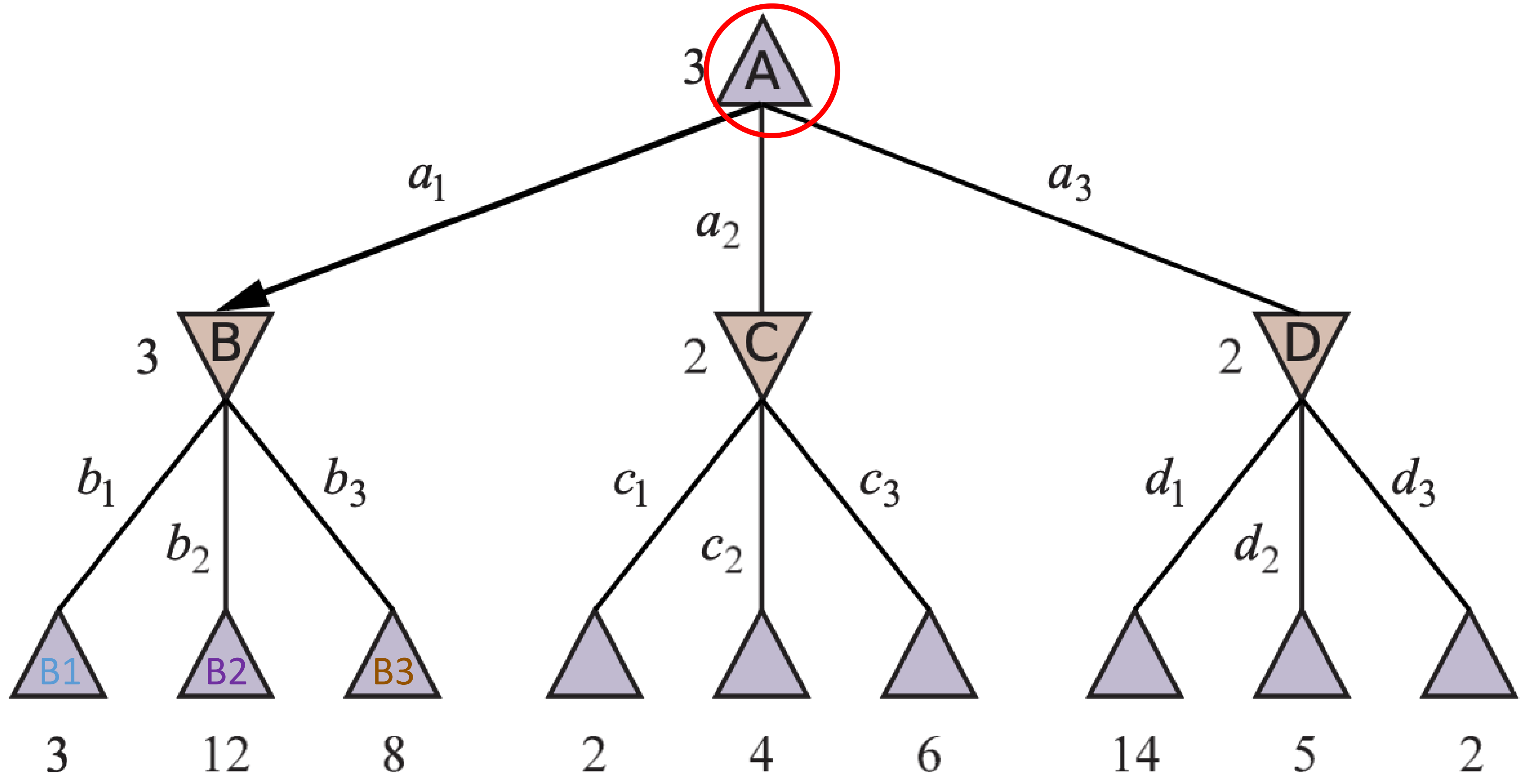
Minimax algorithm

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v



$$\textbf{function } \text{MINIMAX-DECISION}(state) \textbf{ returns } \textit{an action}$$
$$\quad \textbf{return } \arg\max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(state, a))$$

Minimax algorithm



Properties of minimax

- Minimax performs a complete DFS exploration of the game tree
- Completeness: Yes (if tree is finite)
- Optimality: Yes (against an optimal opponent)
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

Example: Chess: $b \approx 35$, $m \approx 100$ for “reasonable” games

➢ Exact solution infeasible

➢ Use **alpha–beta pruning**

The minimax algorithm: problems

- For real games the time cost of minimax is totally impractical, but this algorithm serves as the basis:
  - for the mathematical analysis of games and
  - for more practical algorithms
- Problem with minimax search:
  - The number of game states it has to examine is exponential in the number of moves.
- Unfortunately, the exponent can’t be eliminated, but it can be cut in half.

35

Alpha-beta pruning

- It is possible to compute the correct minimax decision without looking at every node in the game tree.
- Alpha-beta pruning allows to eliminate large parts of the tree from consideration, without influencing the final decision.

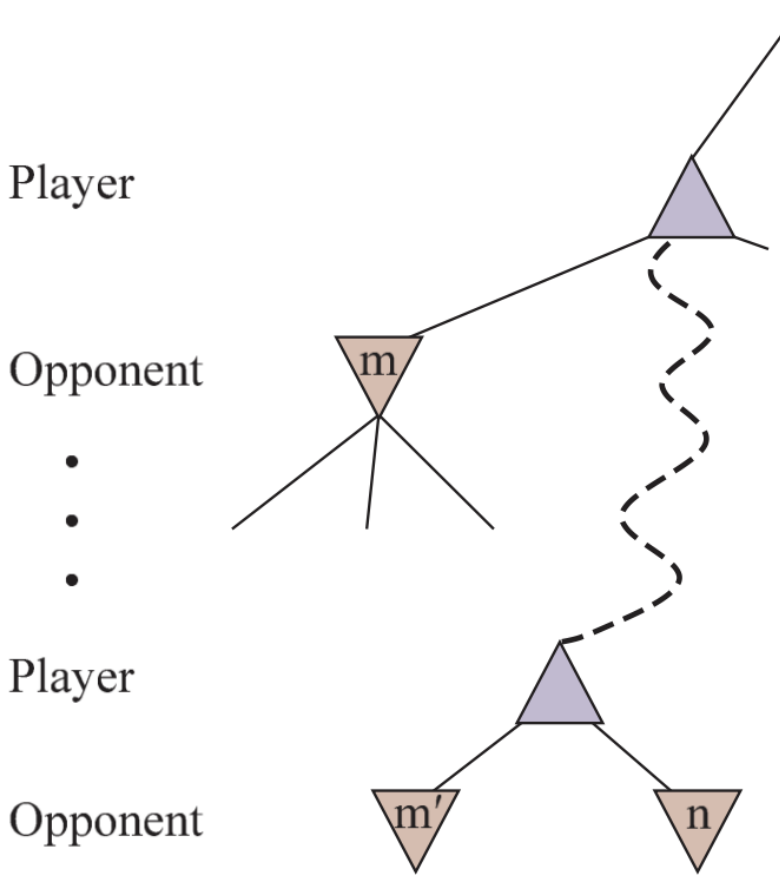
36

Why is it called $\alpha - \beta$?

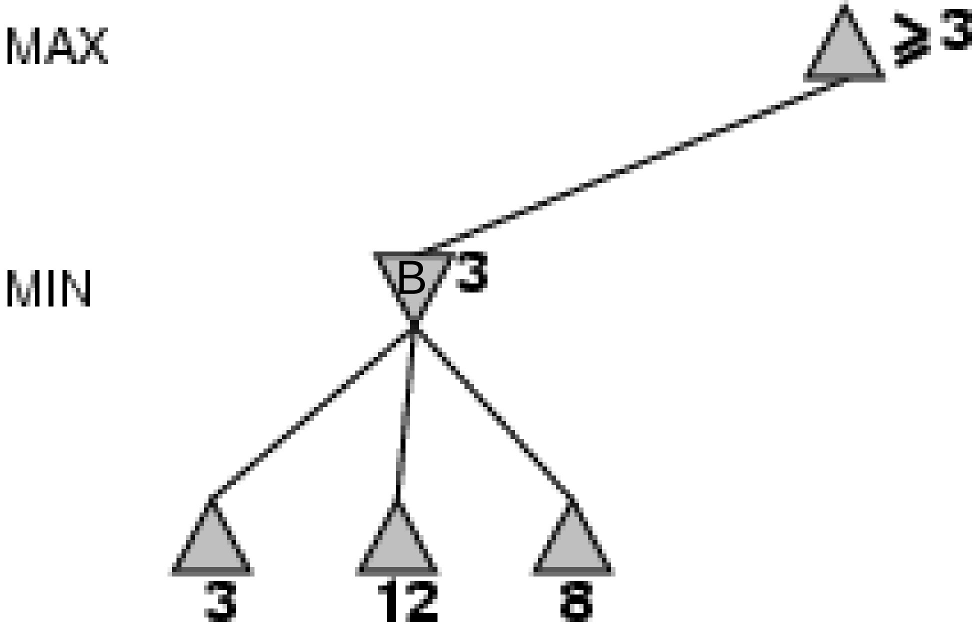
- α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Pruning in $\alpha - \beta$

- Consider node n that the player (MAX) can move to
- If m is a better node higher up in the tree, then n will never be reached
- MIN applies the same ideas in their turn



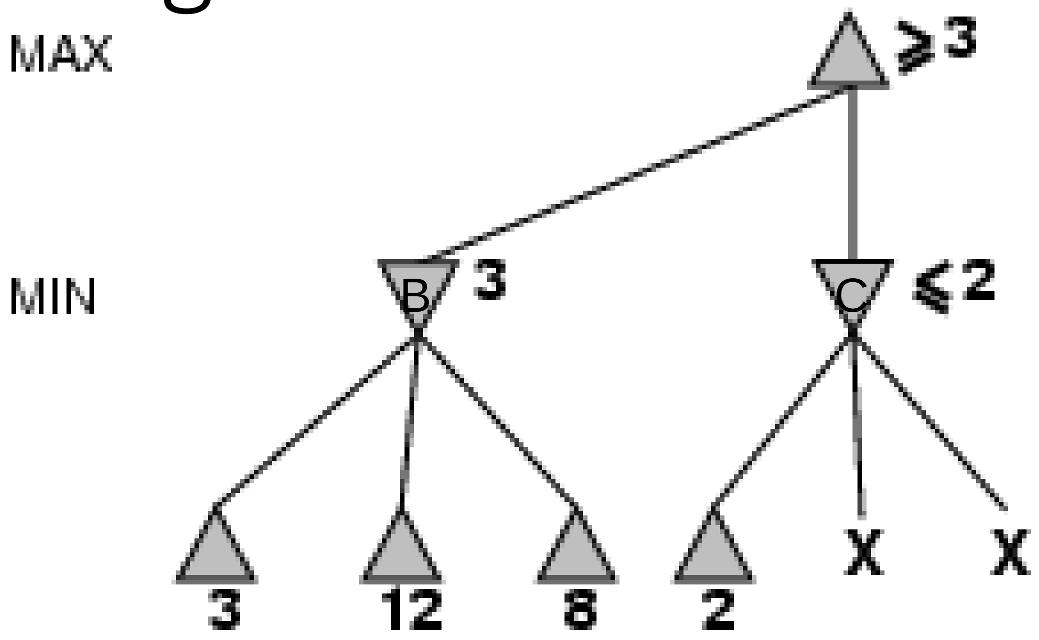
Alpha-beta pruning



- The leaves below *B* have the values 3, 12 and 8.
- The value of *B* is exactly 3.
- It can be inferred that the value at the root is *at least* 3, because MAX has a choice worth 3.

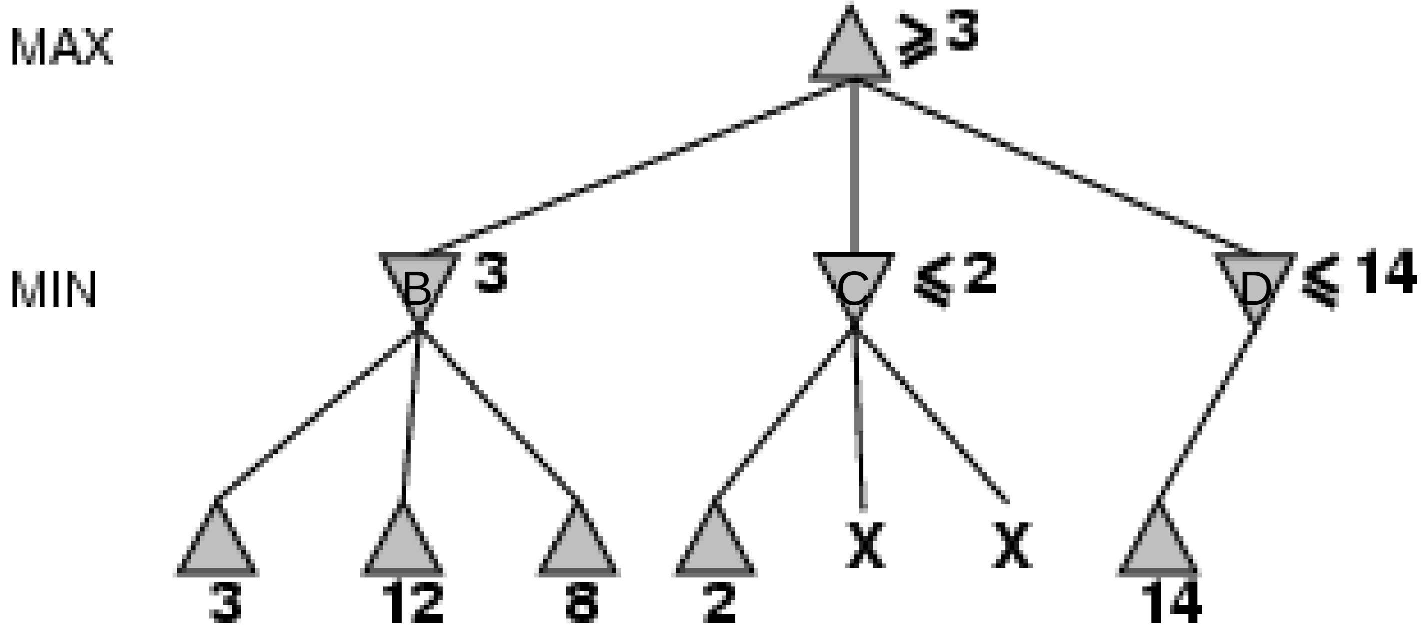
39

Alpha-beta pruning



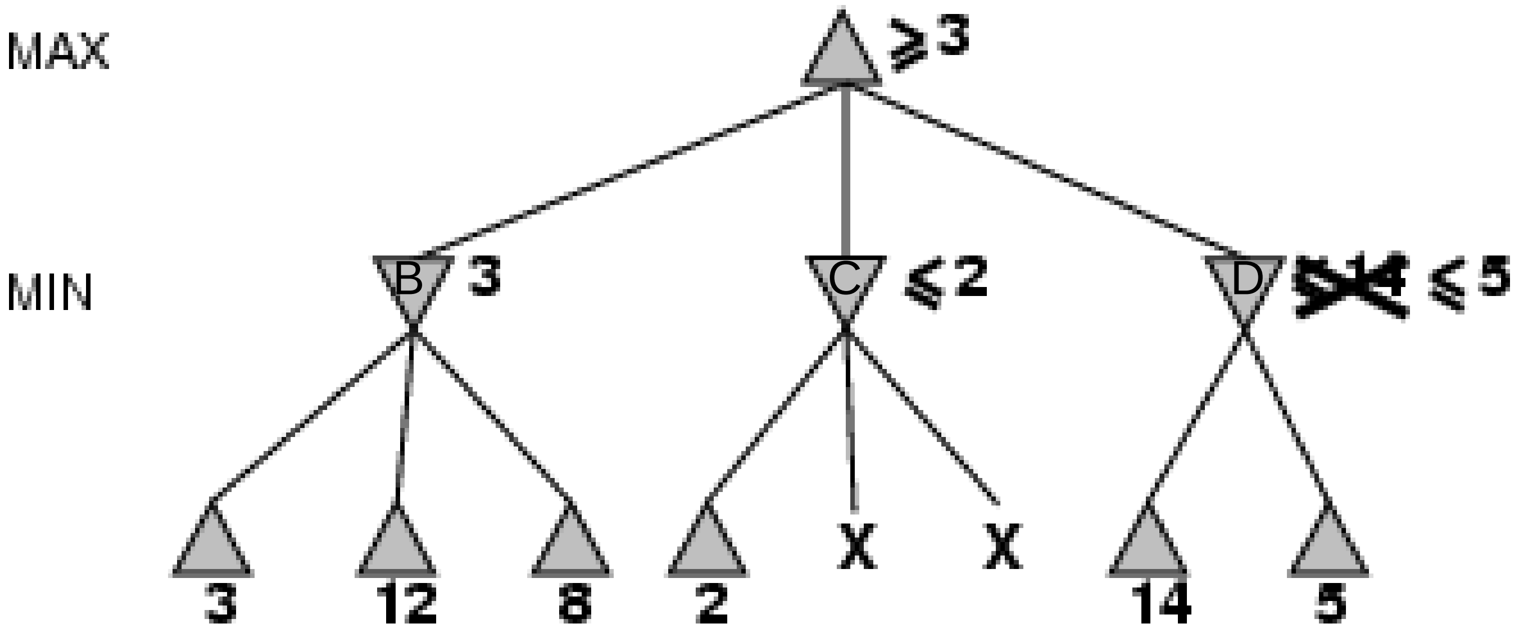
- *C*, which is a MIN node, has a value of *at most* 2.
- But *B* is worth 3, so MAX would never choose *C*.
- Therefore, there is no point in looking at the other successors of *C*.

Alpha-beta pruning



- *D*, which is a MIN node, is worth *at most* 14.
- This is still higher than MAX's best alternative (i.e., 3), so *D*'s other successors are explored.

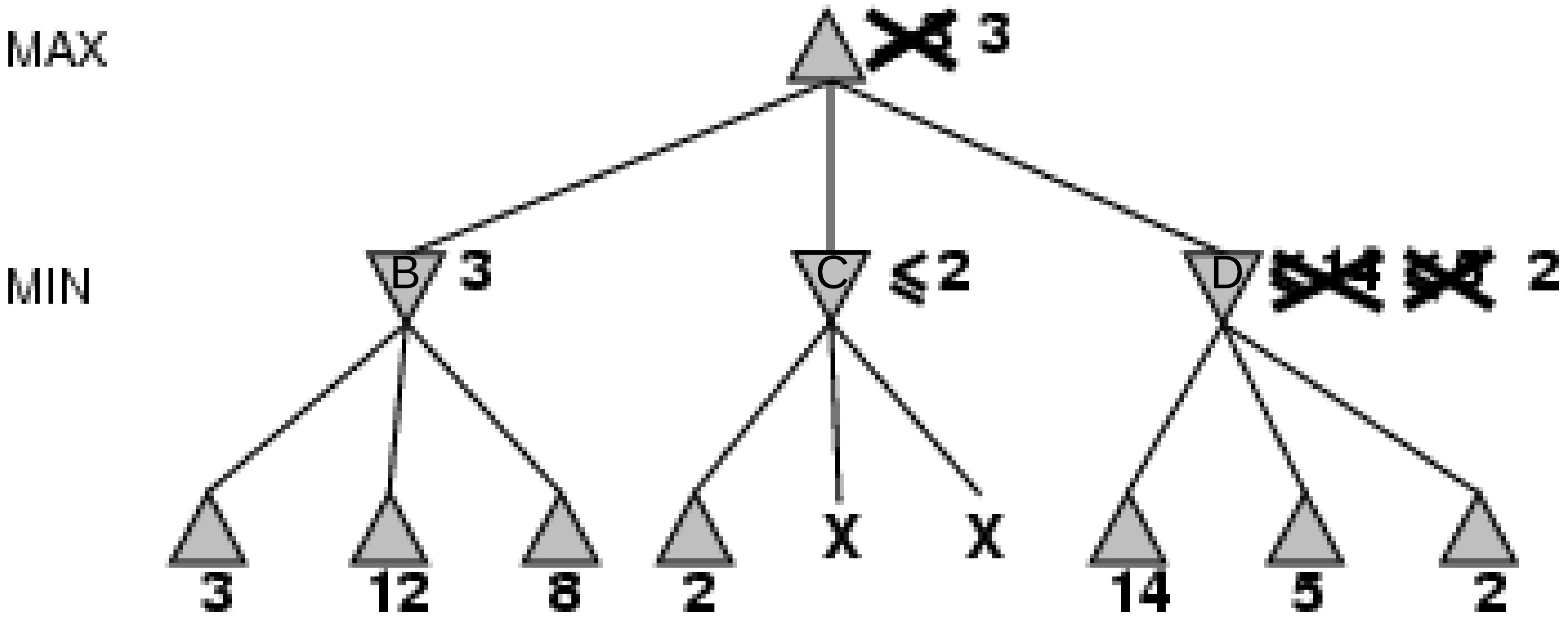
Alpha-beta pruning



- The second successor of *D* is worth 5, so the exploration continues.

42

Alpha-beta pruning



- The third successor is worth 2, so now *D* is worth exactly 2.
- MAX’s decision at the root is to move to *B*, giving a value of 3

Alpha-beta pruning

- Alpha-beta pruning gets its name from two parameters.
  - They describe bounds on the values that appear anywhere along the path under consideration:
    - α = the value of the best (i.e., highest value) choice found so far along the path for MAX
    - β = the value of the best (i.e., lowest value) choice found so far along the path for MIN

44

Alpha-beta pruning

- Alpha-beta search updates the values of α and β as it goes along.
- It prunes the remaining branches at a node (i.e., terminates the recursive call)
  - as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

45

The $\alpha - \beta$ algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in ACTIONS(*state*) with value v

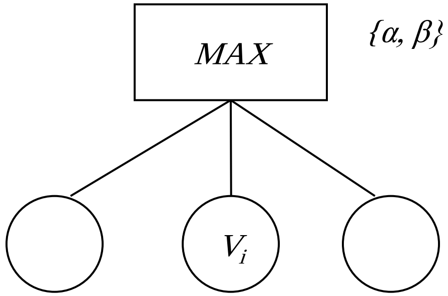
---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

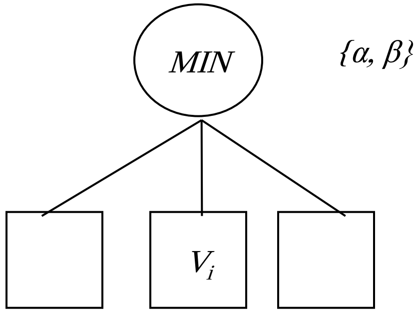
function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Alpha-beta pruning



If $V_i > \alpha$, modify α
If $V_i \geq \beta$, β pruning

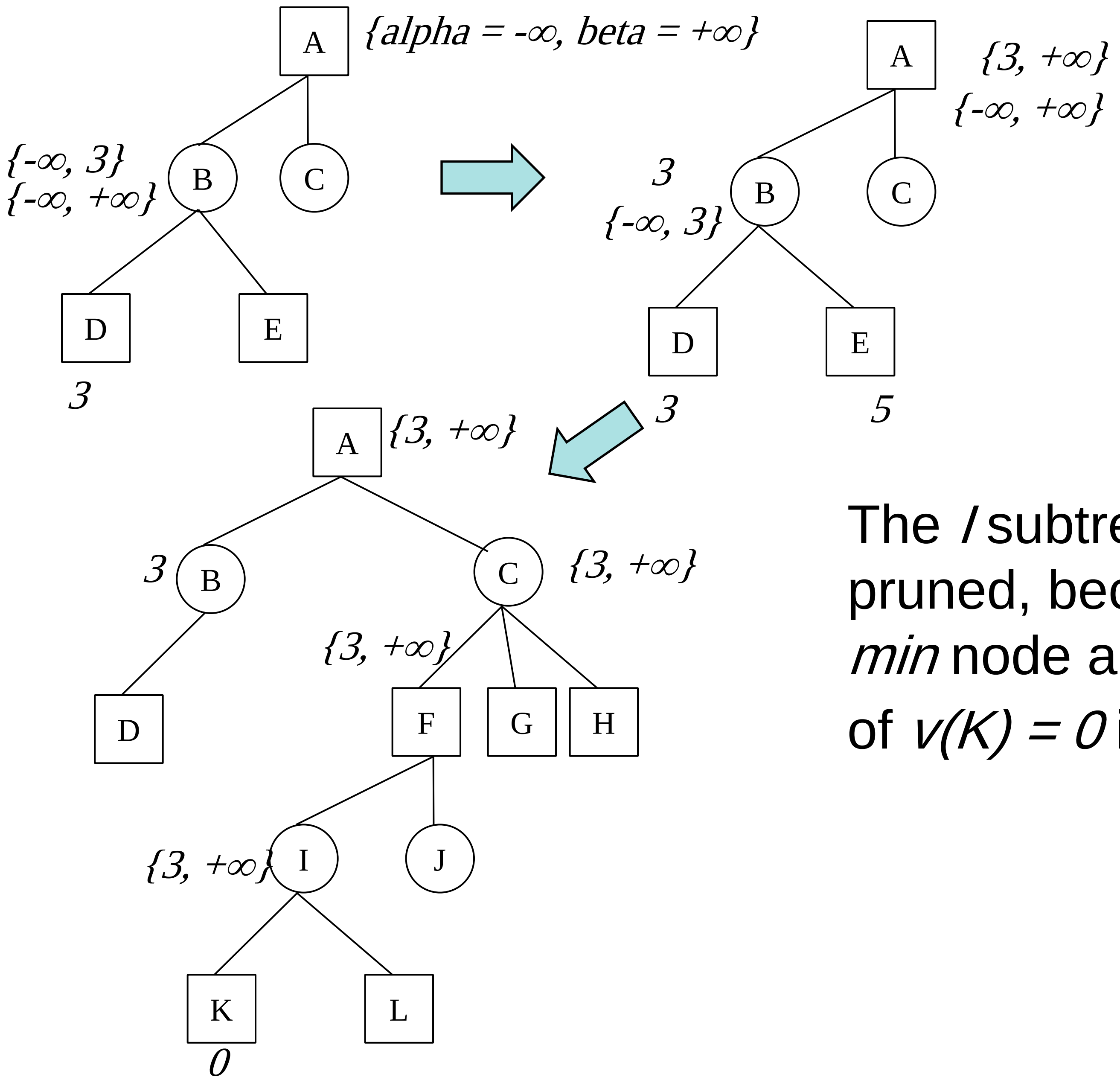
Return α



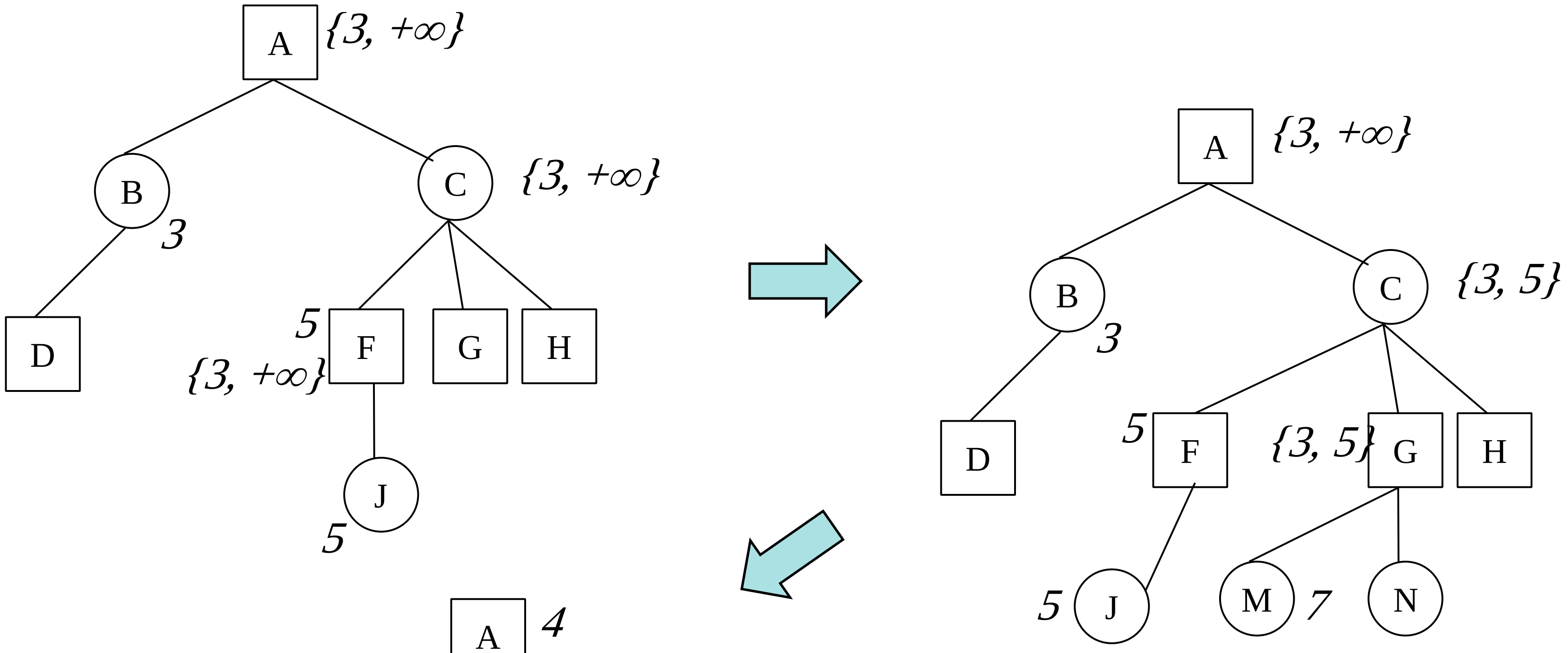
If $V_i < \beta$, modify β
If $V_i \leq \alpha$, α pruning

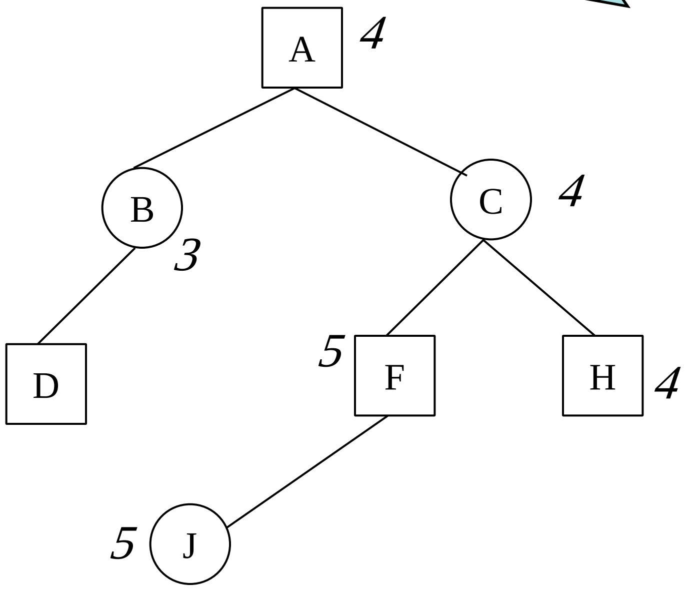
Return β

α and β bounds are transmitted from parent to child in the order of node visit. The effectiveness of pruning highly depends on the order in which successors are examined.



The *I* subtree can be pruned, because *I* is a *min* node and the value of *v(K) = 0* is < $\alpha = 3$





The *G* subtree can be pruned, because *G* is a *max* node and the value of *v(M) = 7* is > β *= 5*

Properties of $\alpha - \beta$

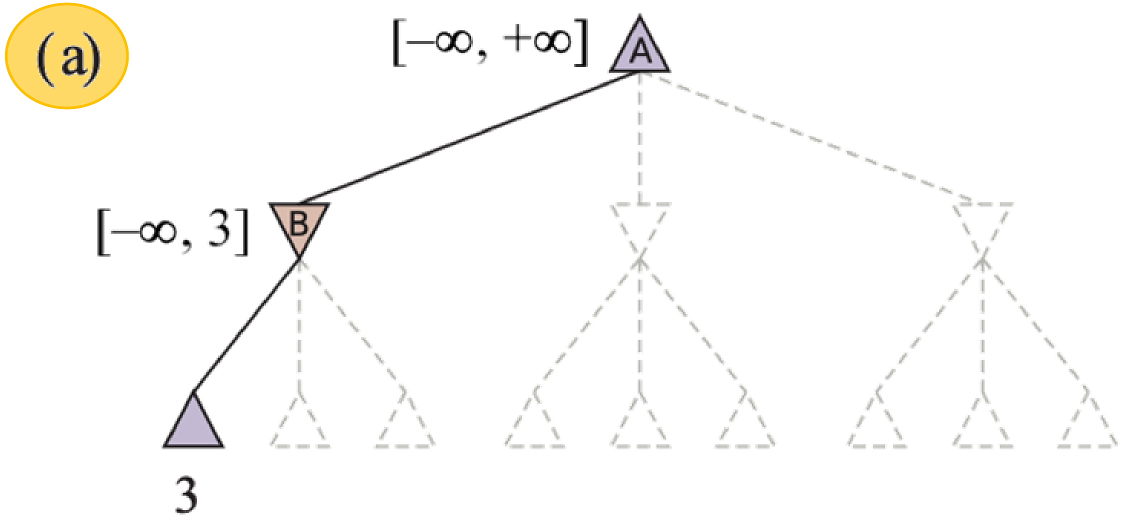
- Pruning does not affect the final result
- Good move ordering (i.e. which nodes to examine first) improves effectiveness of pruning
- With “perfect ordering”, time complexity = $O\left(b^{m/2}\right)$ instead of $O(b^m)$
  - doubles depth of search

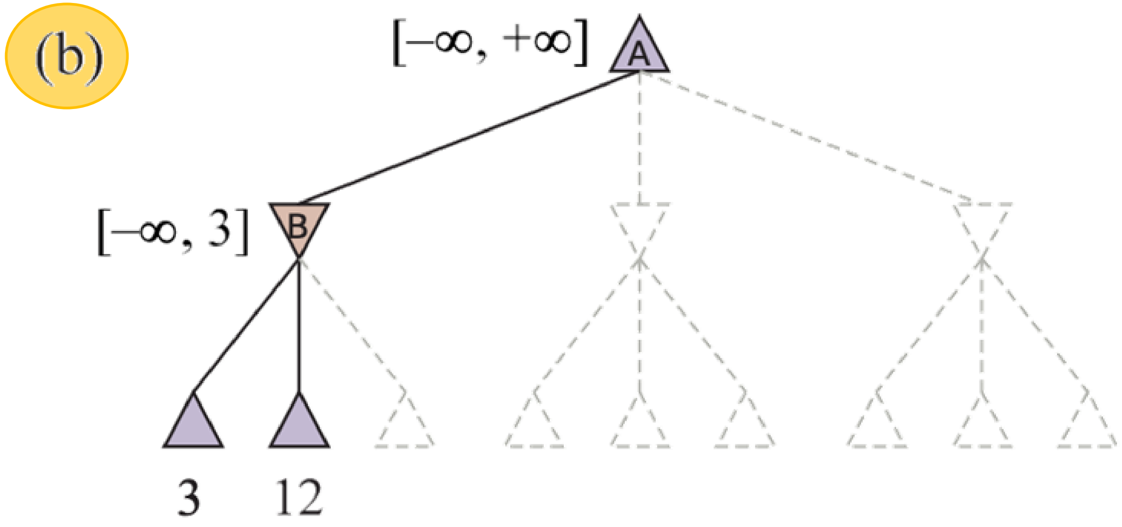
$\alpha - \beta$ pruning

Remember, **Max** is searching

α = the highest-value found so far at any choice point along the path for MAX

β = the lowest-value found so far at any choice point along the path for MIN





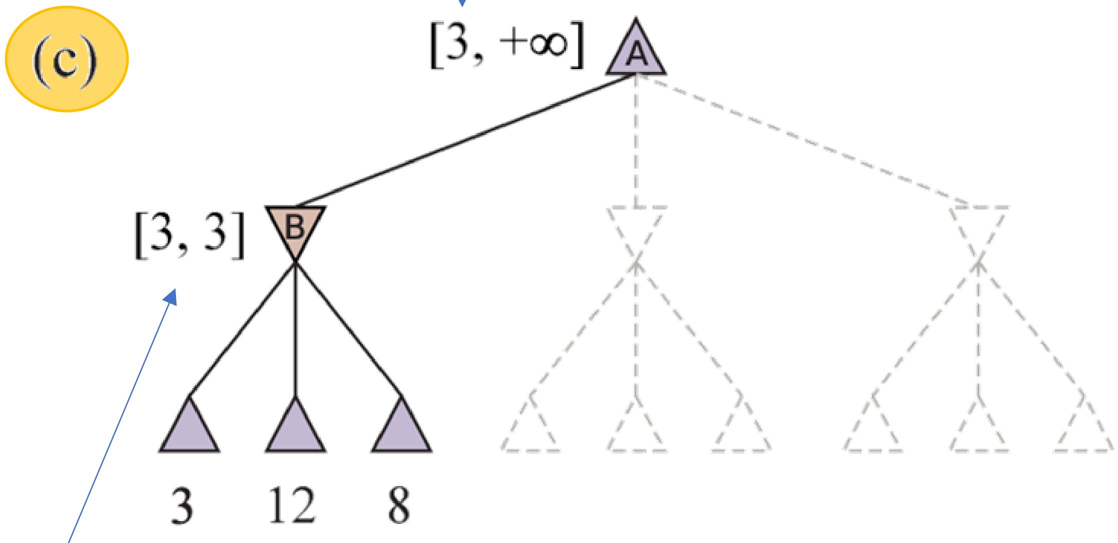
$\alpha - \beta$ pruning

Remember, **Max** is searching

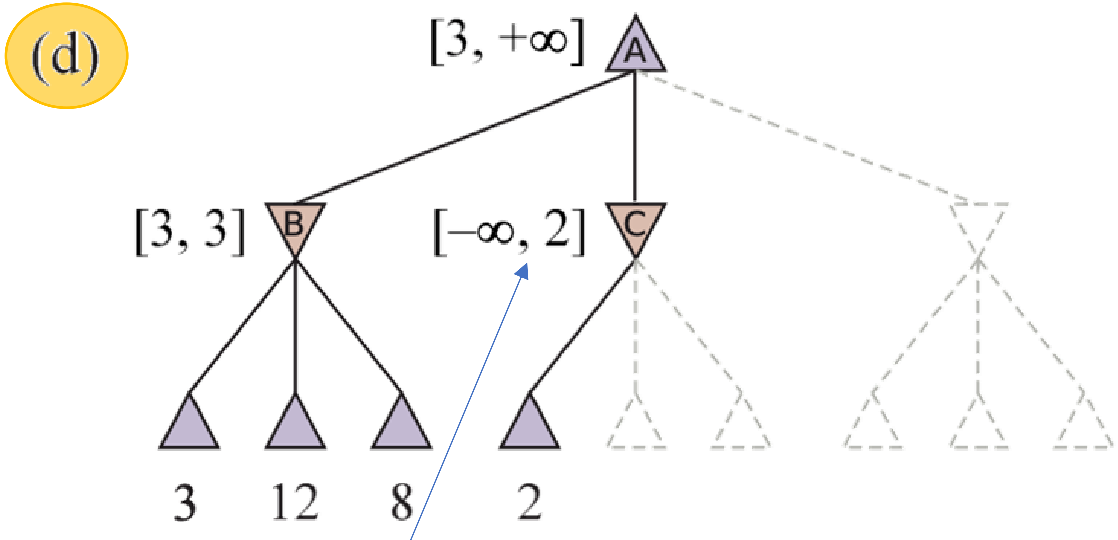
α = the highest-value found so far at any choice point along the path for MAX

β = the lowest-value found so far at any choice point along the path for MIN

MAX can get at least 3



The best MAX can get is 3



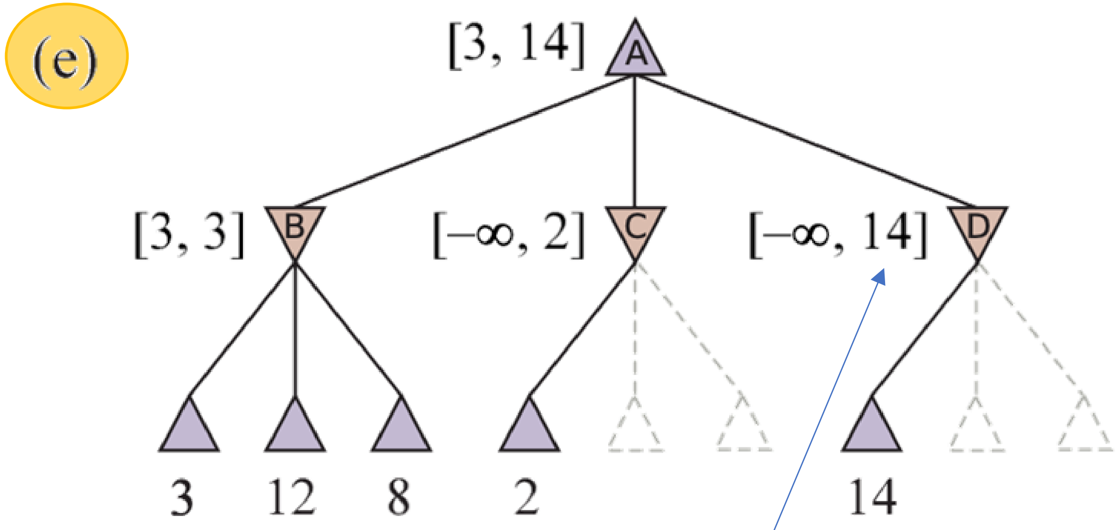
The best MAX can get is 2, which is worse than 3 from node B, so MAX doesn't bother looking in the rest of the sub-tree. Stop.

$\alpha - \beta$ pruning

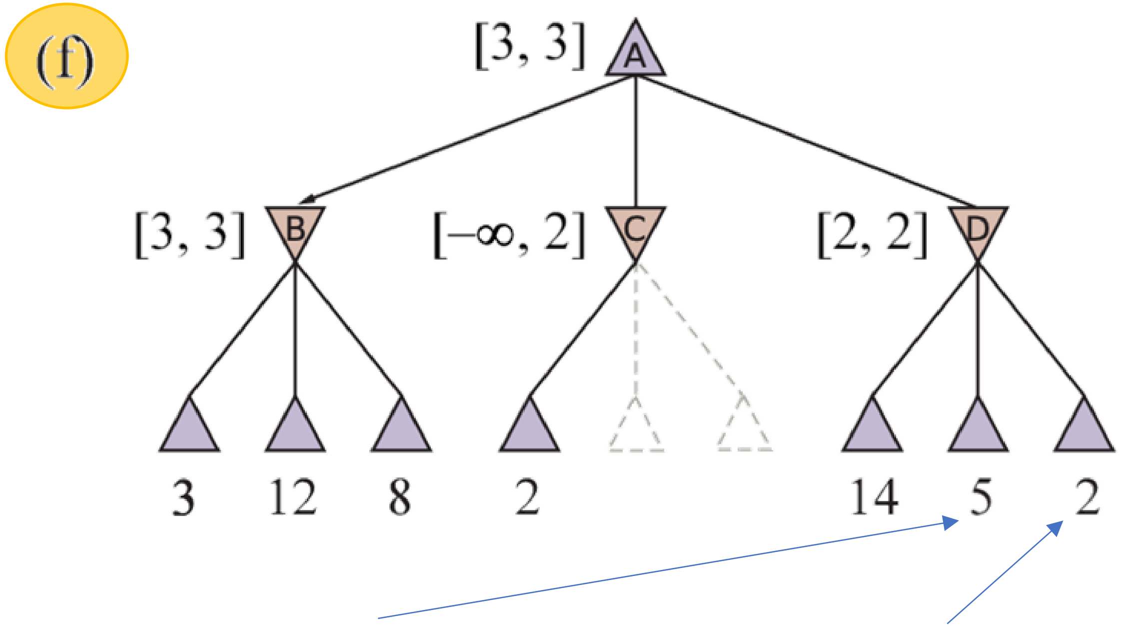
Remember, **Max** is searching

α = the highest-value found so far at any choice point along the path for MAX

β = the lowest-value found so far at any choice point along the path for MIN



The best MAX can get is 14, keep looking (better than 3)



The best MAX can get now is 5, keep looking (still better than 3)

The best MAX can get now is 2, stop

MAX-VALUE(A, $-\infty,+\infty$)

Example

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

$v \leftarrow$MAX-VALUE(state,$-\infty,+\infty$)
$v \leftarrow$MAX-VALUE(A,$-\infty,+\infty$)

$[-\infty, +\infty]$ 

function MAX-VALUE(A,$-\infty,+\infty$) **returns** a utility value
  if TERMINAL-TEST(A) **then return** UTILITY(A)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(A) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))

$-\infty$ B, $-\infty,+\infty$

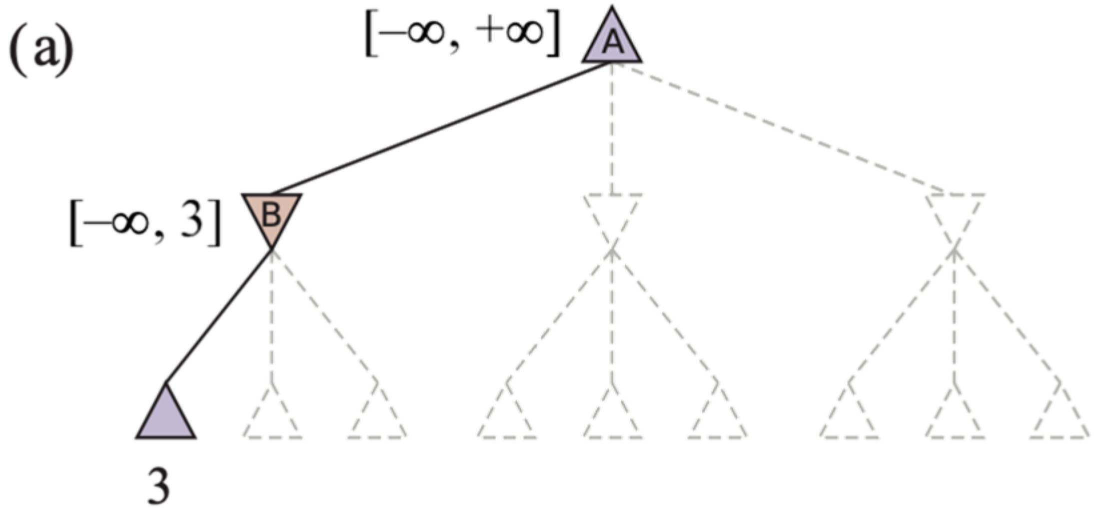
C Second Loop
D Third Loop

MIN-VALUE(B, $-\infty,+\infty$)

Example

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(B, α, β) **returns** a utility value
  if TERMINAL-TEST(B) **then return** UTILITY(B)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(B) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B,a), α, β))
    $+\infty$ B1, $-\infty,+\infty$
    B2
    B3



function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

$v \leftarrow$ MIN($+\infty$, 3) $v = 3, \alpha = -\infty, \beta = +\infty$
If $v \leq \alpha$ **then return** v $3 \leq -\infty$? No
$\beta \leftarrow$ MIN($+\infty$, 3) $\beta = 3$

MIN-VALUE(B, $-\infty$,$+\infty$)

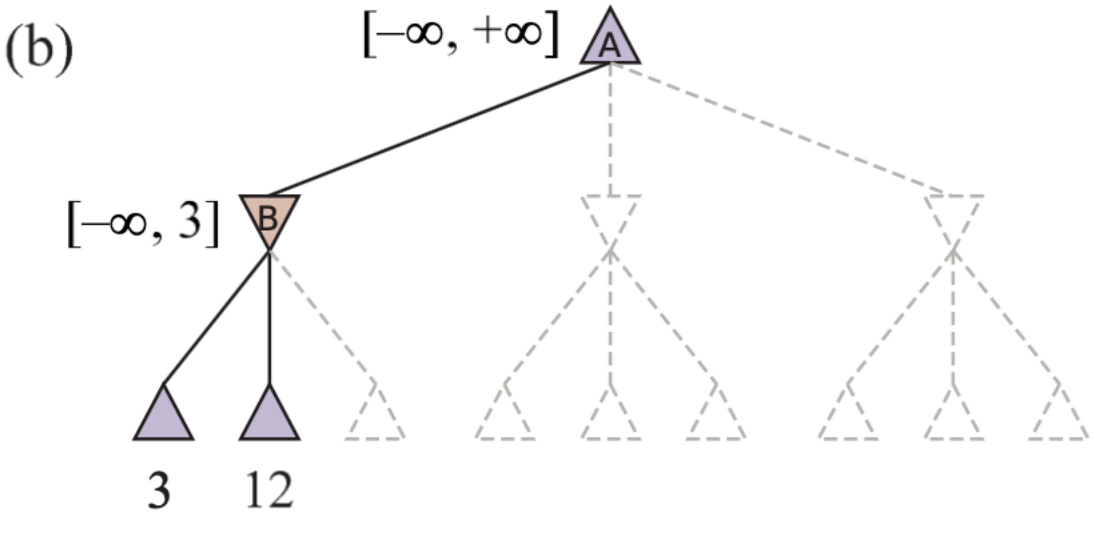
Example

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(B, α, β) **returns** a utility value
  if TERMINAL-TEST(B) **then return** UTILITY(B)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(B) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B,a), α, β))

3

B1, $-\infty$,$+\infty$
B2, $-\infty$, 3
B3,



function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

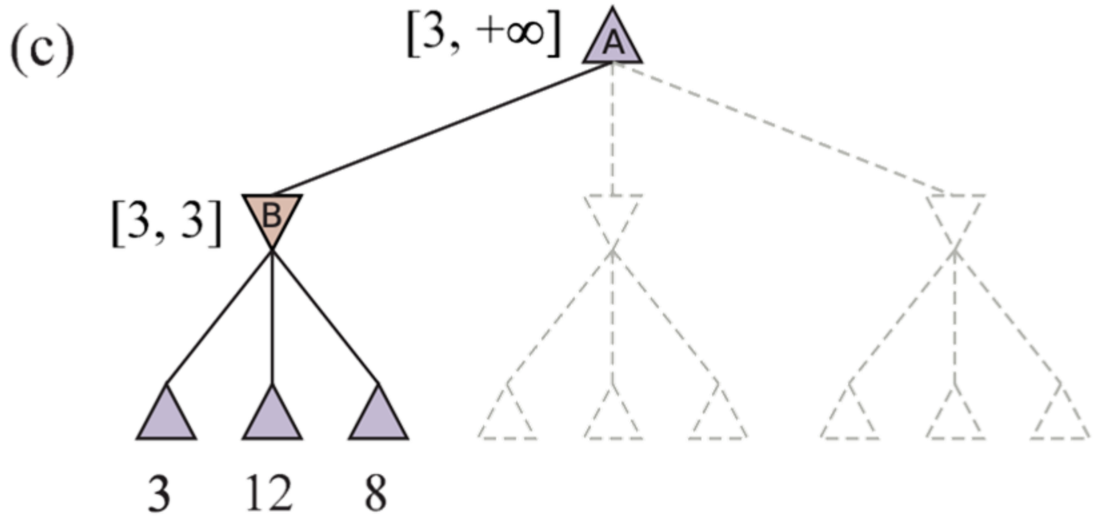
$v \leftarrow$ MIN(3, 12) $v = 3, \alpha = -\infty, \beta = 3$
If $v \leq \alpha$ **then return** v $3 \leq -\infty$? No
$\beta \leftarrow$ MIN(3, 3) $\beta = 3$

MIN-VALUE(B, $-\infty,+\infty$)

Example

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(B, α, β) **returns** a utility value
  if TERMINAL-TEST(B) **then return** UTILITY(B)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(B) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(B,a), α, β))
    3   B1, $-\infty,+\infty$
    B2, $-\infty$, 3
    B3, $-\infty$, 3



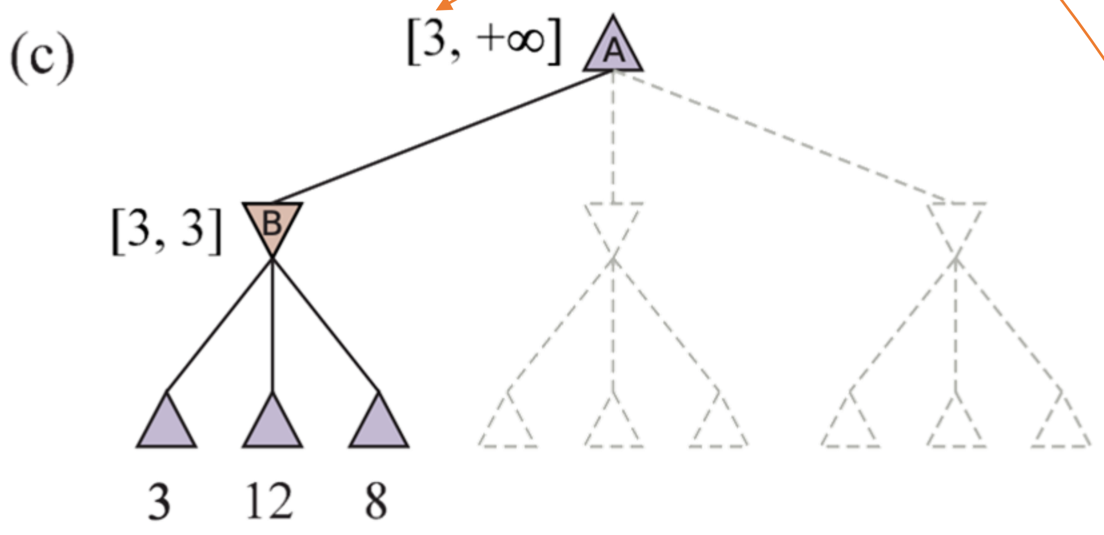
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

$v \leftarrow$ MIN(3, 8) $v = 3, \alpha = -\infty, \beta = 3$
If $v \leq \alpha$ **then return** v $3 \leq -\infty$? No
$\beta \leftarrow$ MIN(3, 3) $\beta = 3$
return v $v = 3$ Loop of state B finished, return to MAX-VALUE(A, $-\infty,+\infty$)

MAX-VALUE(A, $-\infty, +\infty$)

Example

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v



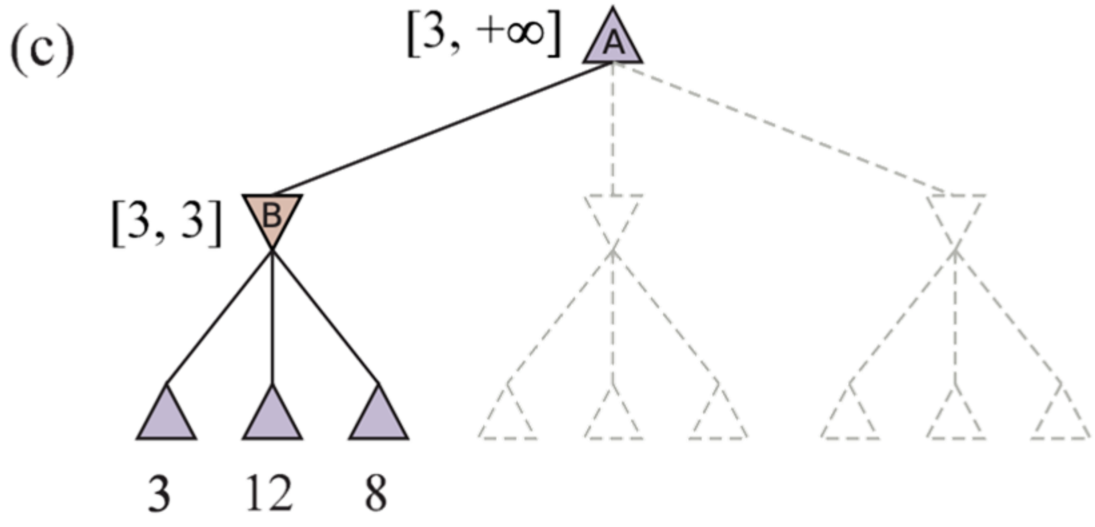
function MAX-VALUE(A, α, β) **returns** a utility value
  if TERMINAL-TEST(A) **then return** UTILITY(A)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(A) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))
        $-\infty$ B, $-\infty, +\infty$
                C
                D
    $v \leftarrow$ MAX($-\infty$, 3), $v = 3$
  If $v \geq \beta$ **then return** v $3 \geq +\infty$? No
  $\alpha \leftarrow$ MAX (α, v) MAX($-\infty$, 3), $\alpha = 3$

MAX-VALUE(A, $-\infty$,$+\infty$)

Example

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

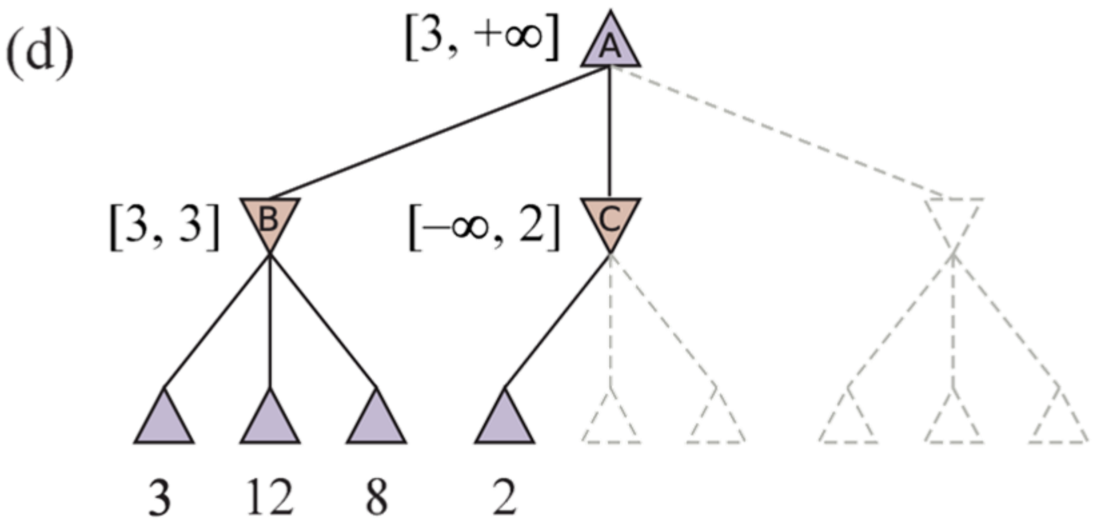
function MAX-VALUE(A, α, β) **returns** a utility value
  if TERMINAL-TEST(A) **then return** UTILITY(A)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(A) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))
      3   B, $-\infty$,$+\infty$
      C, 3, $+\infty$
      D



Example

MIN-VALUE(C, 3,+∞)

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v



function MIN-VALUE(C, α, β) **returns** a utility value
  if TERMINAL-TEST(C) **then return** UTILITY(C)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(C) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(C,a), α, β))
      $+\infty$ C1, 3,+∞
            C2
            C3

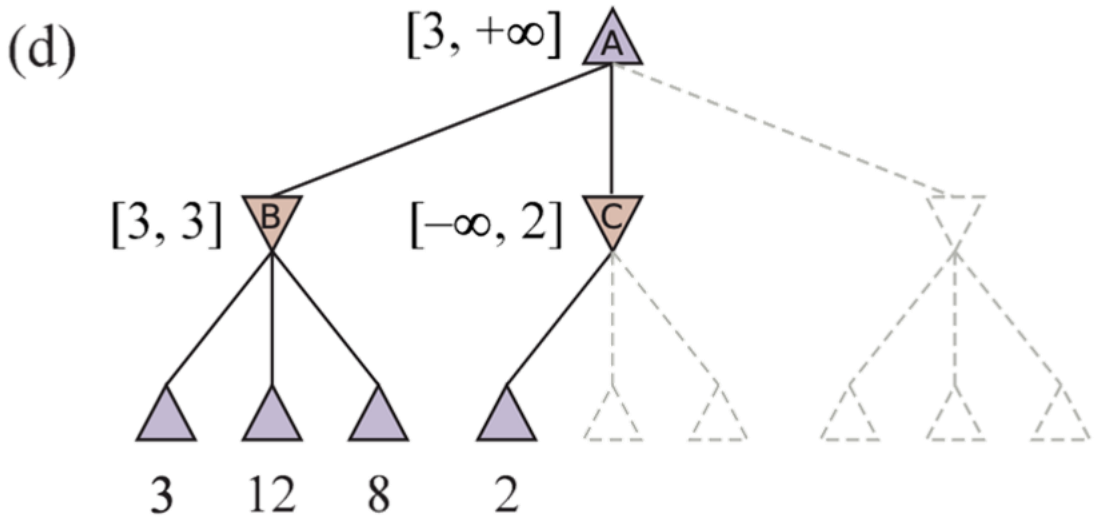
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

$v \leftarrow$ MIN(+∞, 2) $v = 2, \alpha = 3, \beta = +\infty$
If $v \leq \alpha$ **then return** v $2 \leq 3$? YES
So: Return **2** to MAX-VALUE(A, −∞,+∞)
We don't check C2 and C3

MAX-VALUE(A, $-\infty$,$+\infty$)

Example

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
if TERMINAL-TEST($state$) **then return** UTILITY($state$)
$v \leftarrow -\infty$
for each a **in** ACTIONS($state$) **do**
$v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
if $v \geq \beta$ **then return** v
$\alpha \leftarrow$ MAX(α, v)
return v

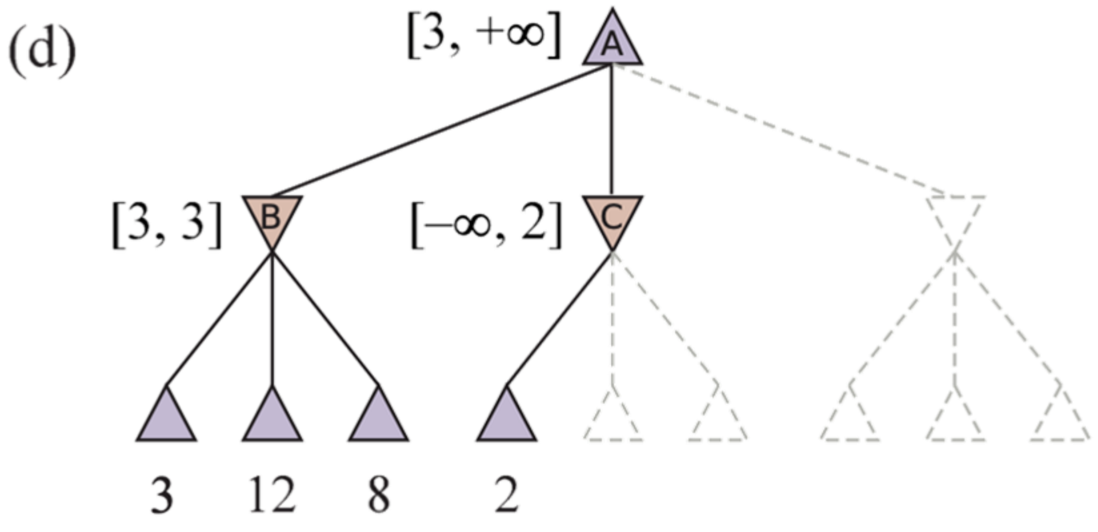


function MAX-VALUE(A, α, β) **returns** a utility value
if TERMINAL-TEST(A) **then return** UTILITY(A)
$v \leftarrow -\infty$
for each a **in** ACTIONS(A) **do**
$v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))
3 B, $-\infty$,$+\infty$
C, 3, $+\infty$
D
$v \leftarrow$ MAX(3, 2), $v = 3$
If $v \geq \beta$ **then return** v $3 \geq +\infty$? No
$\alpha \leftarrow$ MAX (α, v) MAX(3, 3), $\alpha = 3$

Example

MAX-VALUE(A, $-\infty$,$+\infty$)

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v



function MAX-VALUE(A, α, β) **returns** a utility value
  if TERMINAL-TEST(A) **then return** UTILITY(A)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(A) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))
      3

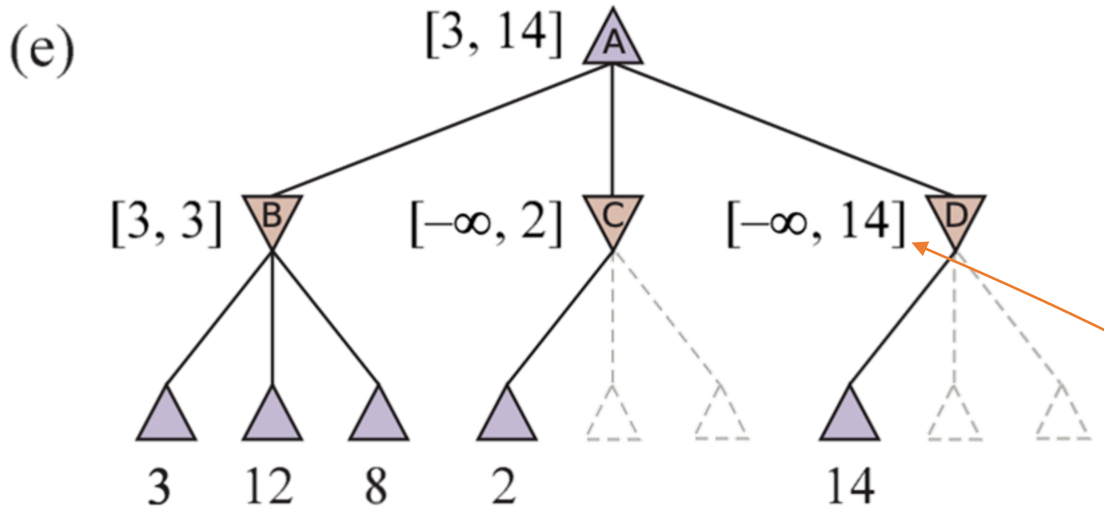
B, $-\infty$,$+\infty$
C, 3, $+\infty$
D, 3, $+\infty$

MIN-VALUE(D, 3,+∞)

Example

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(D, α, β) **returns** a utility value
  if TERMINAL-TEST(D) **then return** UTILITY(D)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(D) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(D,a), α, β))
      $+\infty$ D1, 3,+∞
                D2
                D3



function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

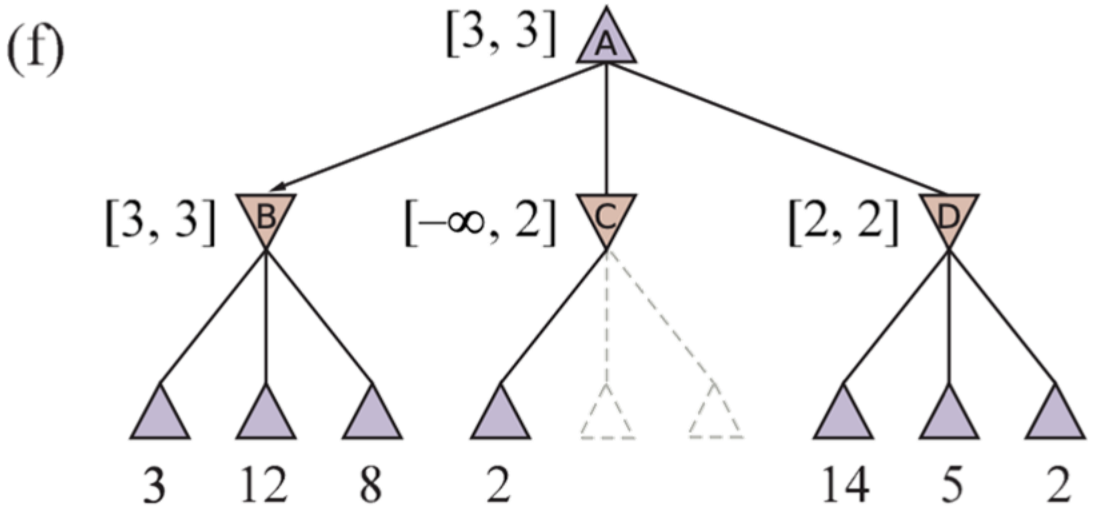
$v \leftarrow$ MIN(+∞, 14) $v = 14, \alpha = 3, \beta = +\infty$
If $v \leq \alpha$ **then return** v $14 \leq 3$? No
$\beta \leftarrow$ MIN(+∞, 14) $\beta = 14$

MIN-VALUE(D, 3,+∞)

Example

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(D, α, β) **returns** a utility value
  if TERMINAL-TEST(D) **then return** UTILITY(D)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(D) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(D,a), α, β))
    14 D1, 3,+∞
    D2, 3, 14
    D3



function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

$v \leftarrow$ MIN(14, 5) $v = 5, \alpha = 3, \beta = 14$
If $v \leq \alpha$ **then return** v $5 \leq 3$? No
$\beta \leftarrow$ MIN(14, 5) $\beta = 5$

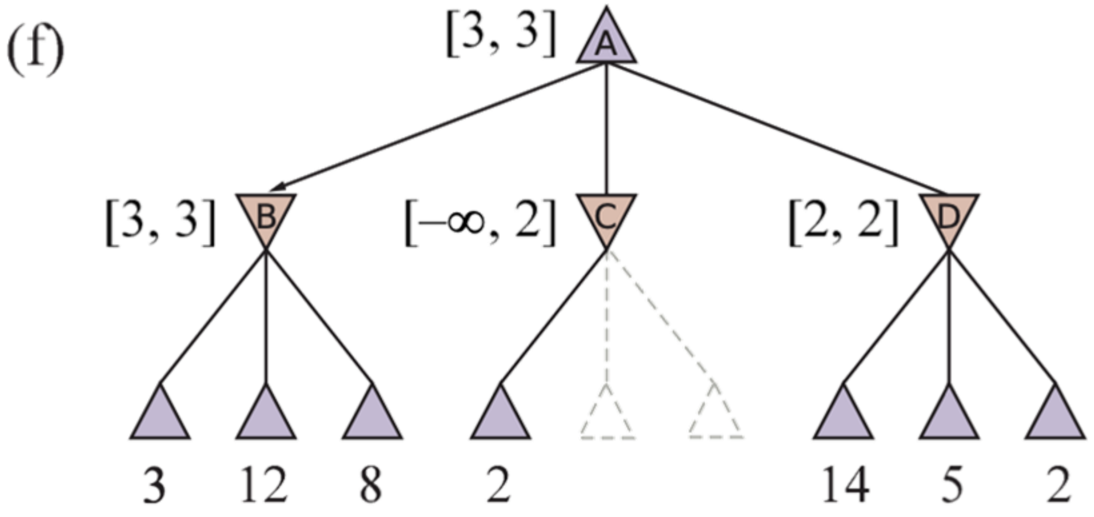
MIN-VALUE(D, 3,+∞)

Example

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) ,α, β))
    if $v \le \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

function MIN-VALUE(D, α, β) **returns** a utility value
  if TERMINAL-TEST(D) **then return** UTILITY(D)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(D) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(D,a), α, β))
    5   D1, 3,+∞
                     D2, 3, 14
                     D3, 3, 5

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)

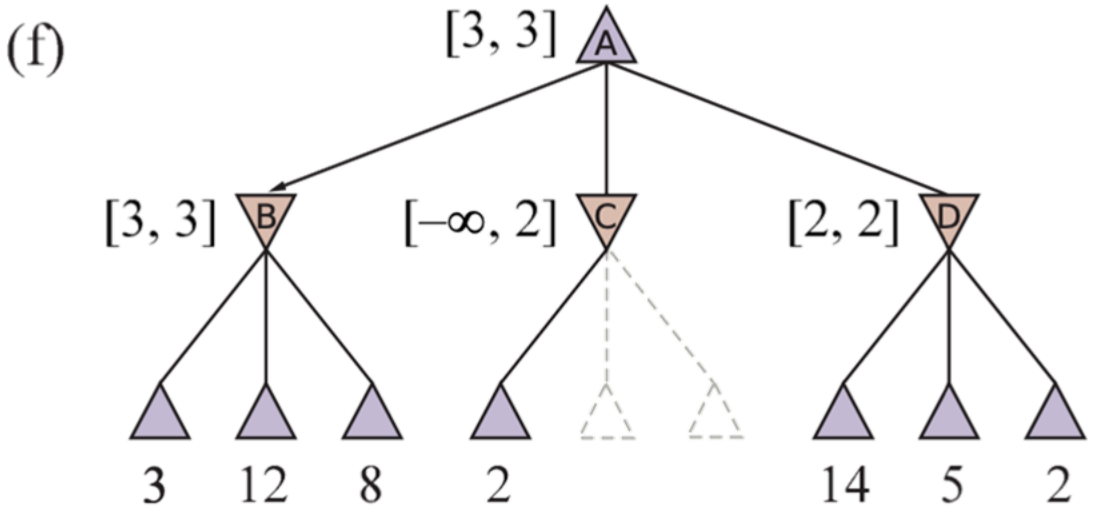


$v \leftarrow$ MIN(5, 2) $v = 2, \alpha = 3, \beta = 5$
If $v \le \alpha$ **then return** v $2 \le 3$? YES
So: Return **2** to MAX-VALUE(A, $-\infty, +\infty$)
We don't check any more, unfortunately, it is the last node

MAX-VALUE(A, $-\infty$,$+\infty$)

Example

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v



function MAX-VALUE(A, α, β) **returns** a utility value
  if TERMINAL-TEST(A) **then return** UTILITY(A)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(A) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(A,a), α, β))
      3   B, $-\infty$,$+\infty$
      C, 3, $+\infty$
      D, 3, $+\infty$
    $v \leftarrow$ MAX(3, 2), $v = 3$
  If $v \geq \beta$ **then return** v $3 \geq +\infty$? No
  $\alpha \leftarrow$ MAX (α, v) MAX(3, 3), $\alpha = 3$
Return $v = 3$
return the action in ACTIONS(A) with value $v = 3$

Resource limits

- The minimax algorithm generates the entire game search space
- Alpha–Beta algorithm allows us to prune large parts of the search space
  - Still must search all the way to terminal states: not practical
- Programs should cut off the search earlier and apply a heuristic **evaluation function** to states in the search:

1. Replace the Utility function by a heuristic evaluation function EVAL
2. Replace the Terminal Test by a Cutoff Test that decides when to apply EVAL

Improved algorithm with cutoff

$$H - MINIMAX(s,d) \ =$$

$$\begin{cases} EVAL(s) & if\ CUTOFF - TEST(s,d) \\ max_{a \in Actions(s)}\ H - MINIMAX(RESULT(s,a), d\ +\ 1) & if\ PLAYER(s)\ =\ MAX \\ min_{a \in Actions(s)}\ H - MINIMAX(RESULT(s,a), d\ +\ 1) & if\ PLAYER(s)\ =\ MIN \end{cases}$$

Summary

- Use Minimax: not practical for large games.
- Apply alpha-beta cuts → exact solution, but the gain depends on move ordering and still not practical for large games.
- Memory and time limitations → search with cutoff and evaluation function.