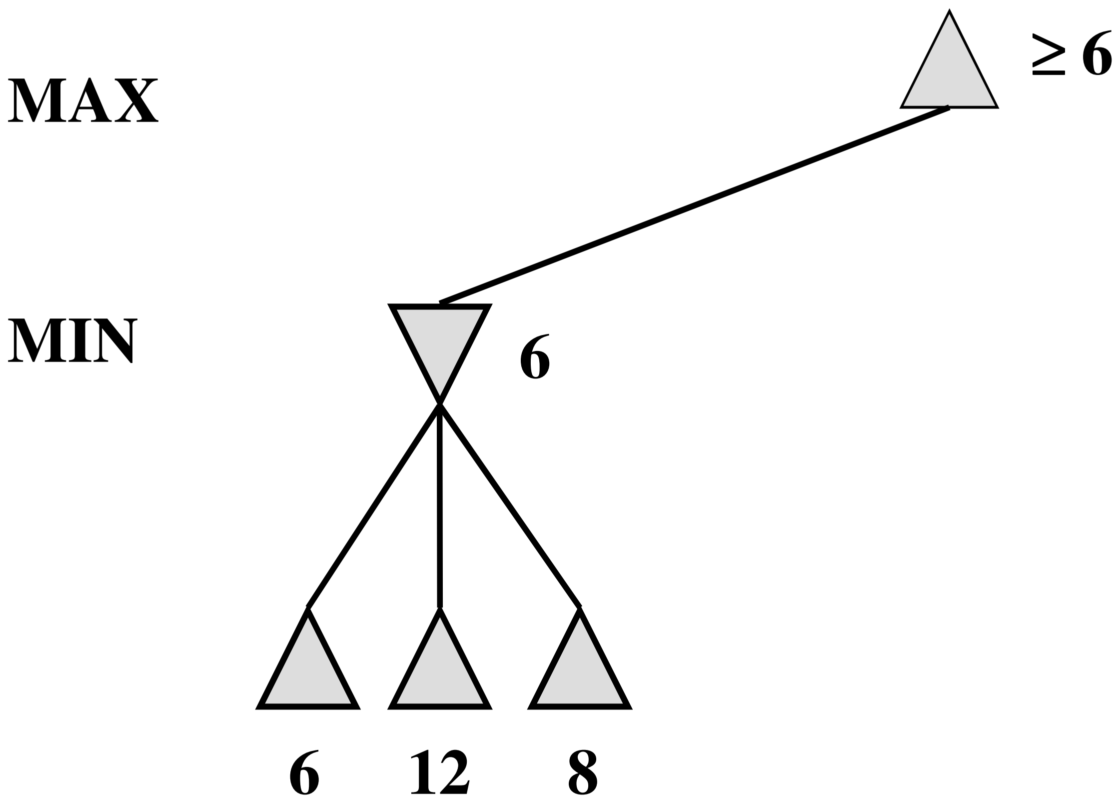
2. α-β pruning: search cutoff

- **Pruning:** eliminating a branch of the search tree from consideration without exhaustive examination of each node

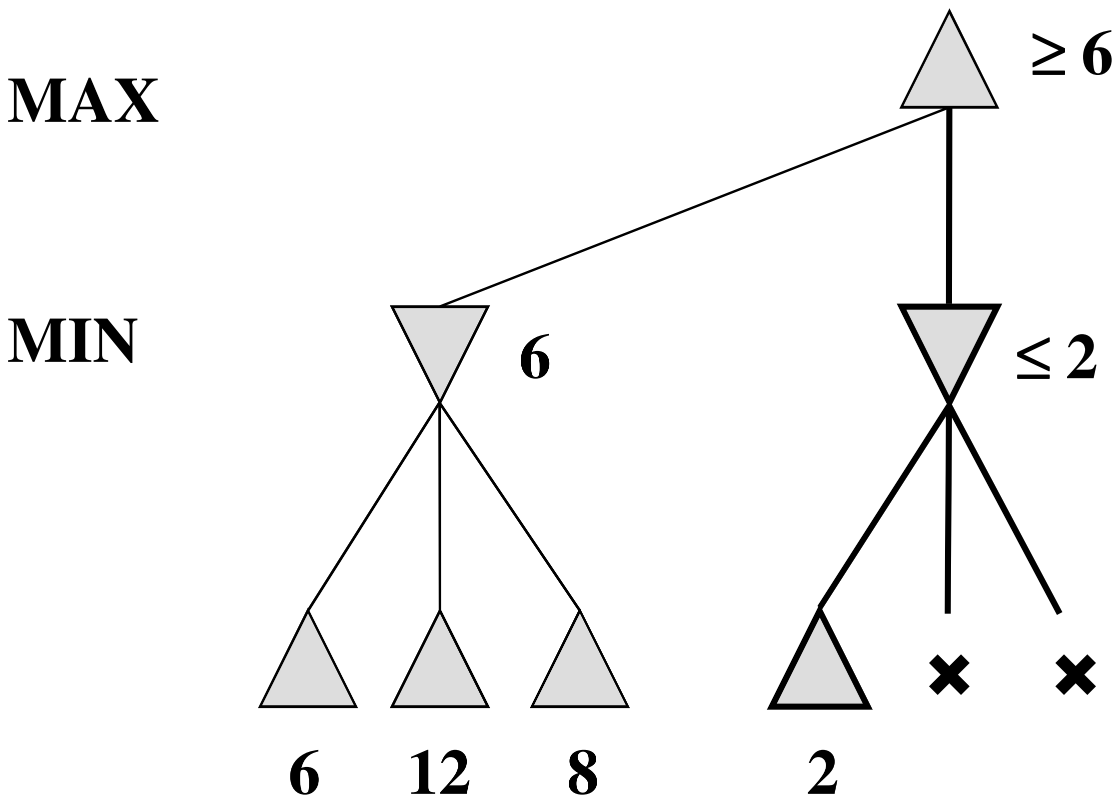
- **α-β pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.

- Does it work? Yes, it roughly cuts the branching factor from b to $\sqrt{b}$ resulting in double as far look-ahead than pure minimax

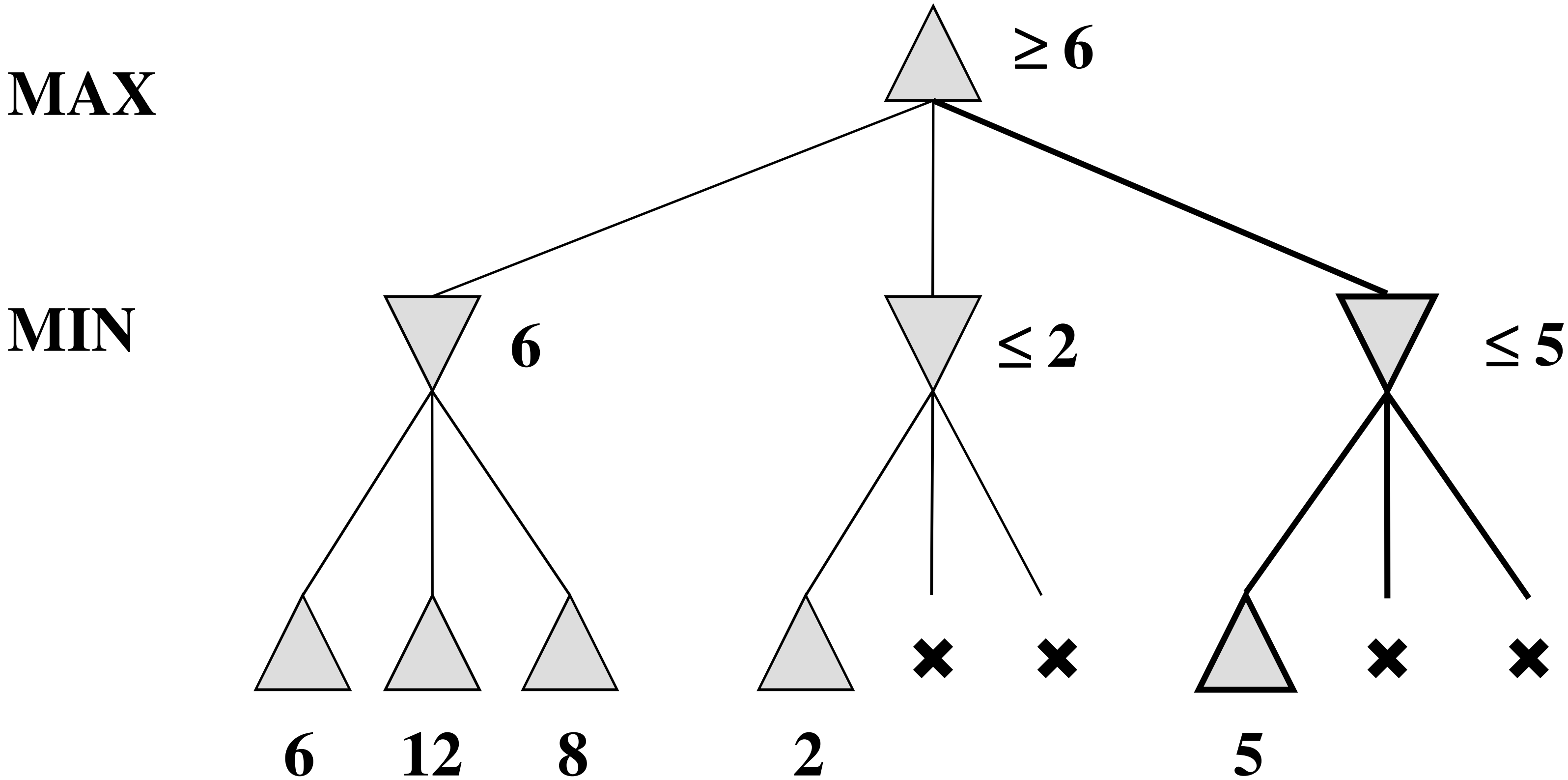
α-β pruning: example



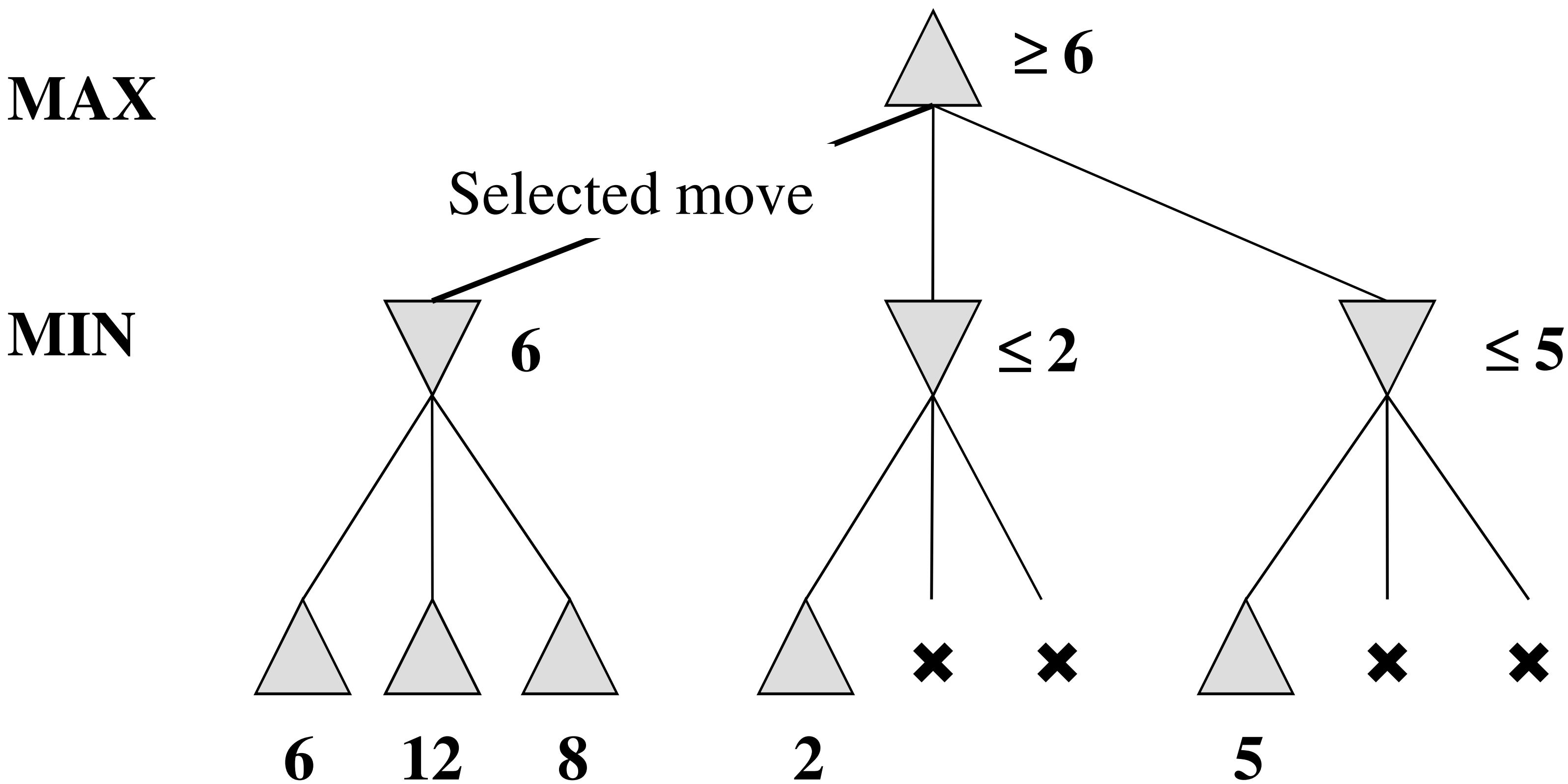
α-β pruning: example



α-β pruning: example



α-β pruning: example



Properties of α-β

Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ *doubles* depth of search
$\Rightarrow$ can easily reach depth 8 and play good chess

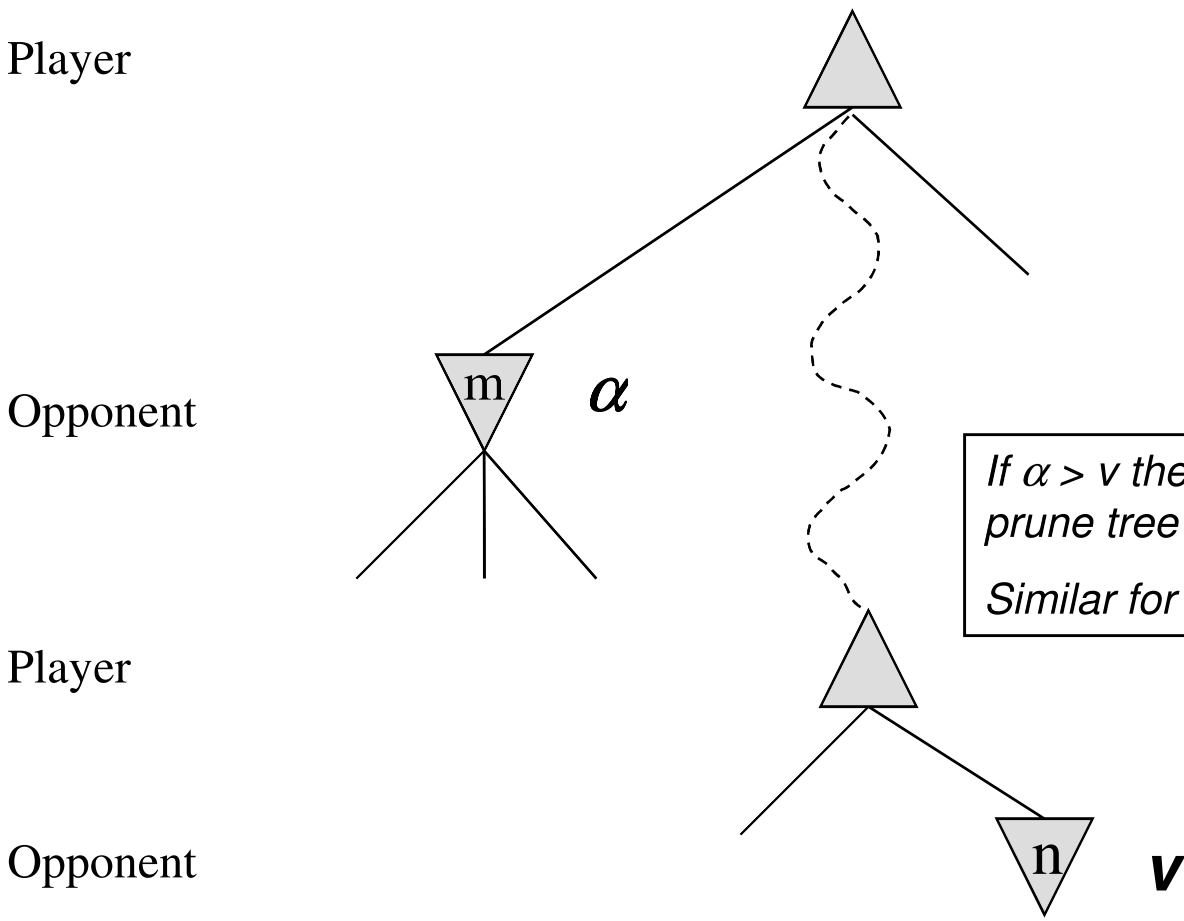
A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

More on the α-β algorithm

- Same basic idea as minimax, but prune (cut away) branches of the tree that we know will not contain the solution.

- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
  - α : Best choice so far for MAX
  - β : Best choice so far for MIN

α-β pruning: general principle



If $\alpha > v$ then MAX will chose m so prune tree under n

Similar for β for MIN

The α-β algorithm:

Basically MINIMAX + keep track of α, β + prune

function MAX-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    α, the best score for MAX along the path to *state*
    β, the best score for MIN along the path to *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each s **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE(s, *game*, α, β))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function MIN-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each s **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE(s, *game*, α, β))
    if $\beta \leq \alpha$ **then return** α
  end
  return β

More on the α-β algorithm: start from Minimax

Basically MINIMAX ~~+ keep track of α, β + prune~~

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           ~~α, the best score for MAX along the path to state~~
           ~~β, the best score for MIN along the path to state~~

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       α ← MAX(α, MIN-VALUE(s, game, α, β))
       ~~if α ≥ β then return β~~
   end
   return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       β ← MIN( β, MAX-VALUE(s, game, α, β))
       ~~if β ≤ α then return α~~
   end
   return β
```

More on the α-β algorithm: start from Minimax

Basically $\textsc{Minimax}$ + keep track of α, β + prune

function $\textsc{Max-Value}(state, game, \alpha, \beta)$ **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    $\boxed{\alpha\text{, the best score for MAX along the path to } state}$
    $\boxed{\beta\text{, the best score for MIN along the path to } state}$

  if $\textsc{Cutoff-Test}(state)$ **then return** $\textsc{Eval}(state)$
  for each s **in** $\textsc{Successors}(state)$ **do**
    $\alpha \leftarrow \textsc{Max}(\alpha, \textsc{Min-Value}(s, game, \alpha, \beta))$
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function $\textsc{Min-Value}(state, game, \alpha, \beta)$ **returns** the minimax value of *state*

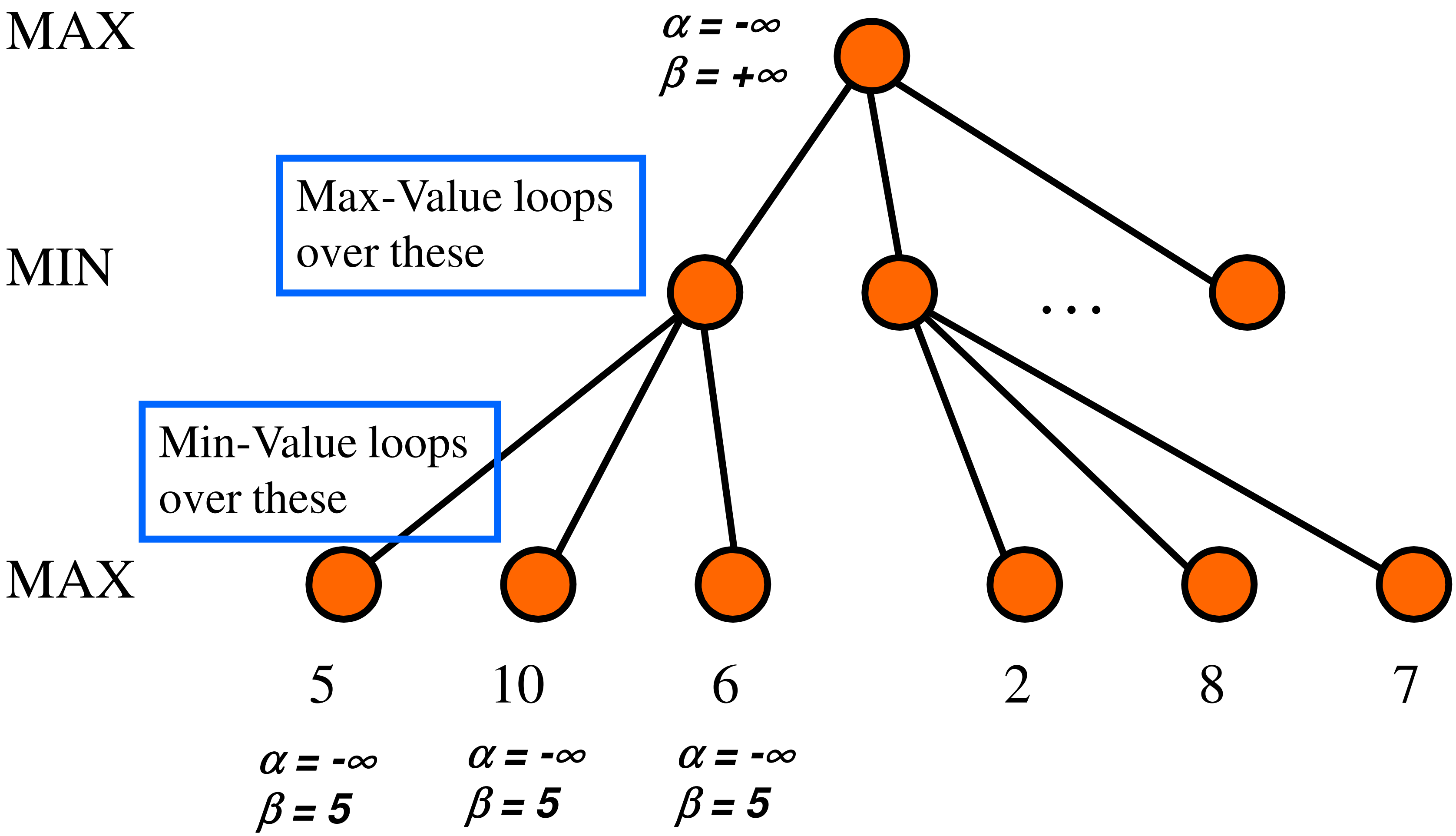
  if $\textsc{Cutoff-Test}(state)$ **then return** $\textsc{Eval}(state)$
  for each s **in** $\textsc{Successors}(state)$ **do**
    $\beta \leftarrow \textsc{Min}(\beta, \textsc{Max-Value}(s, game, \alpha, \beta))$
    if $\beta \leq \alpha$ **then return** α
  end
  return β

Note: These are both Local variables. At the Start of the algorithm, We initialize them to ***α = -∞ and β = +∞***

More on the α-β algorithm

In Min-Value:

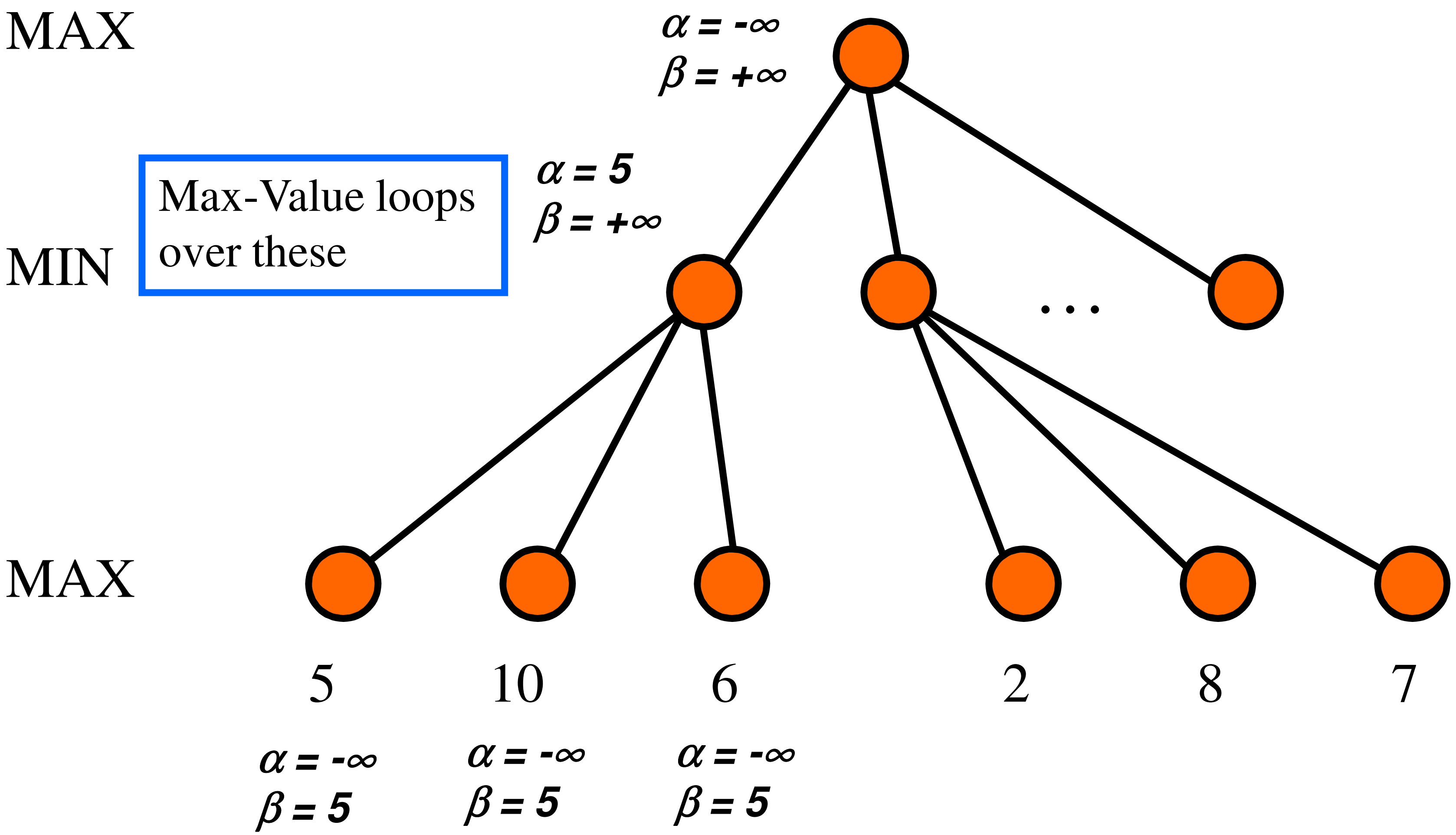
```
for each s in SUCCESSORS(state) do
    β ← MIN(β, MAX-VALUE(s, game, α, β))
    if β ≤ α then return α
end
return β
```



More on the α-β algorithm

In Max-Value:

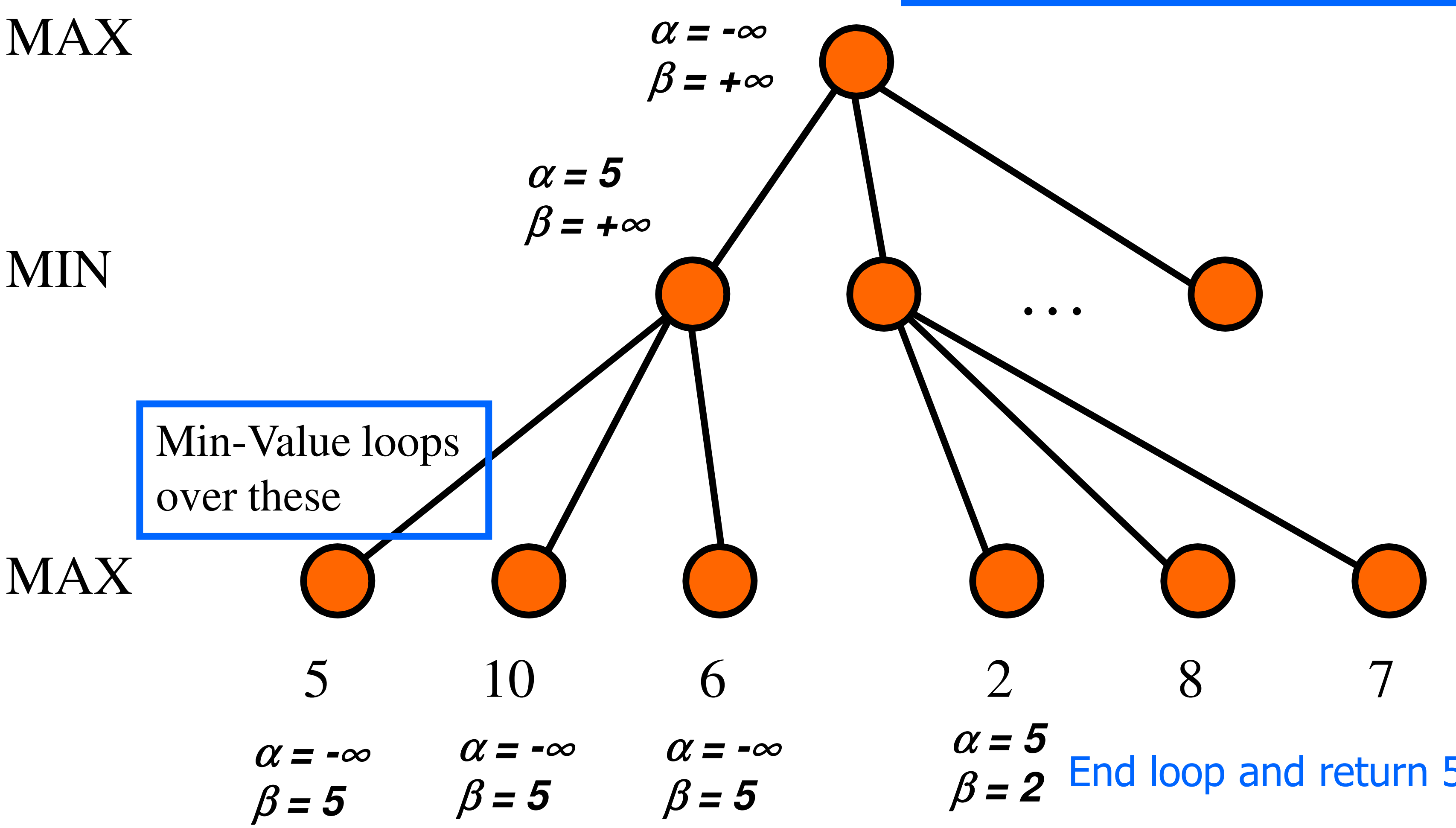
for each s **in** SUCCESSORS($state$) **do**
  $\alpha \leftarrow$ MAX(α, MIN-VALUE(s, $game$, α, β))
  if $\alpha \geq \beta$ **then return** β
end
return α



More on the α-β algorithm

In Min-Value:

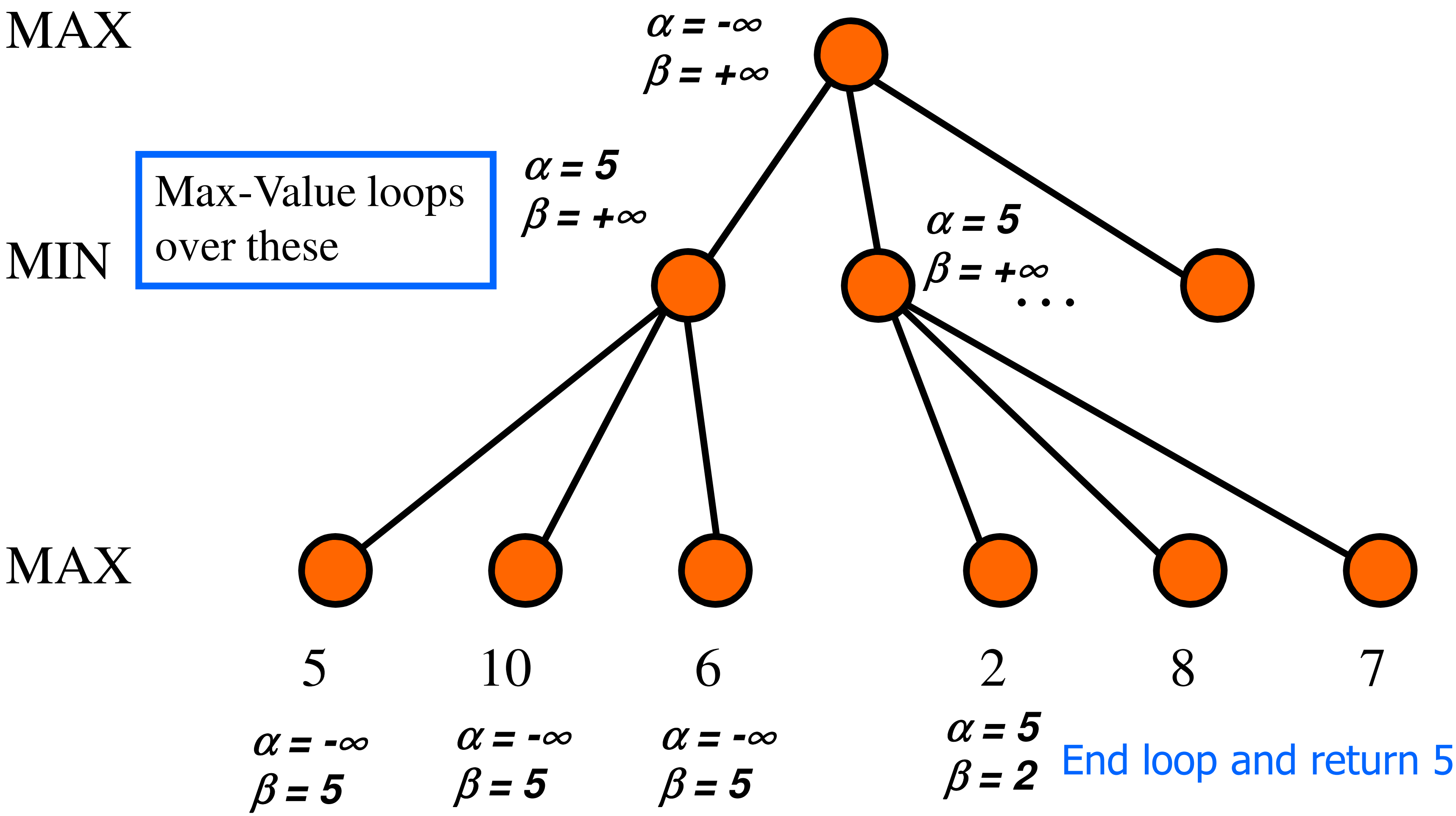
$$
\begin{array}{l}
\textbf{for each } s \textbf{ in } \textsc{Successors}(state) \textbf{ do} \\
\quad \beta \leftarrow \textsc{Min}(\beta, \textsc{Max-Value}(s, game, \alpha, \beta)) \\
\quad \text{if } \beta \leq \alpha \textbf{ then return } \alpha \\
\textbf{end} \\
\textbf{return } \beta
\end{array}
$$



More on the α-β algorithm

In Max-Value:

```
for each s in SUCCESSORS(state) do
    α ← MAX(α, MIN-VALUE(s, game, α, β))
    if α ≥ β then return β
end
return α
```



Another way to understand the algorithm

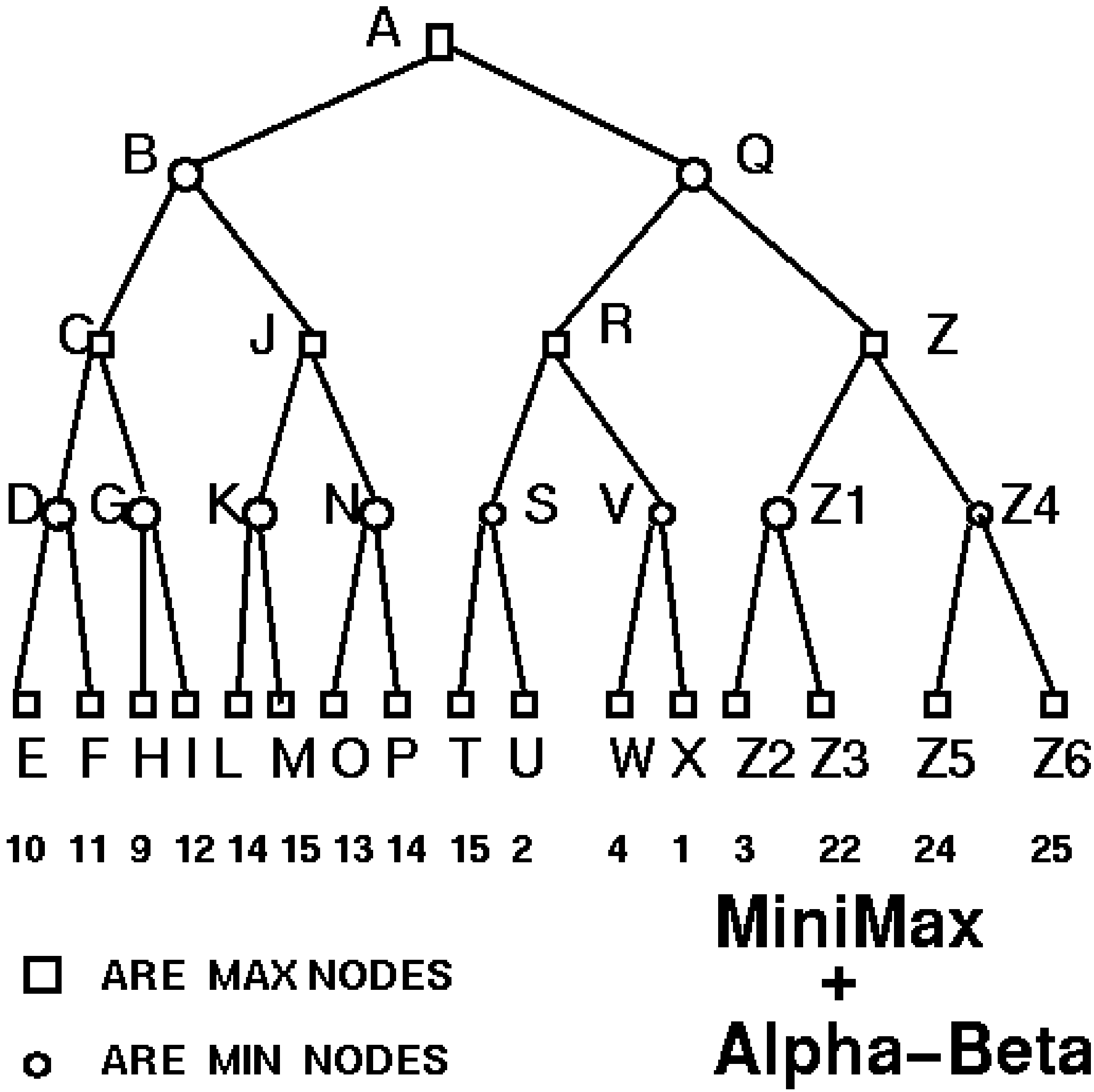
- From:

http://www.cs.swarthmore.edu/~meeden/Minimax/TypicalCase.html
http://www.cs.berkeley.edu/~milch/cs188/alpha-beta.html

- For a given node N,

α is the value of N to MAX
β is the value of N to MIN

Example



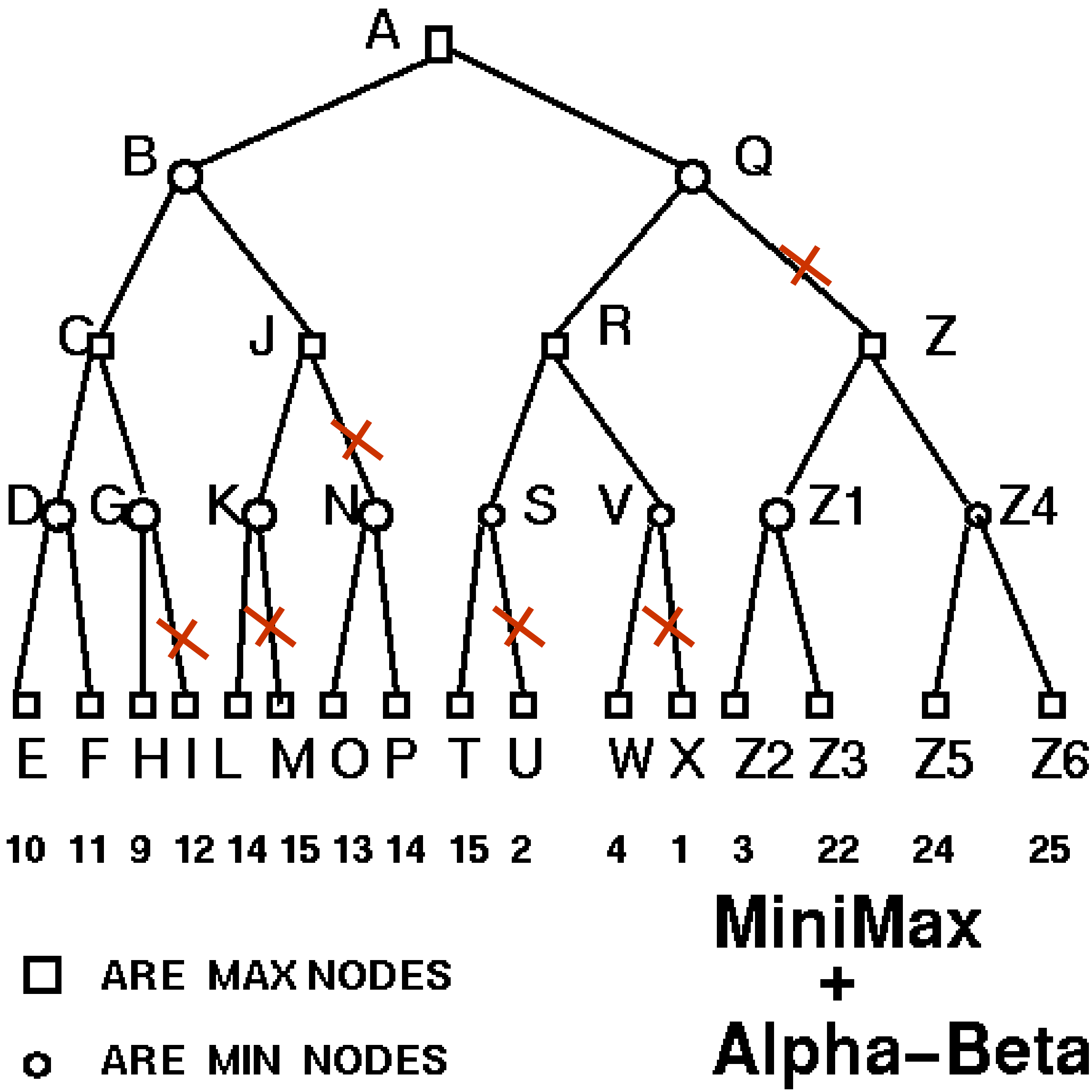
MiniMax + Alpha–Beta

Solution

NODE	TYPE	ALPHA	BETA	SCORE
A	Max	-I	+I	
B	Min	-I	+I	
C	Max	-I	+I	
D	Min	-I	+I	
E	Max	10	10	10
D	Min	-I	10	
F	Max	11	11	11
D	Min	-I	10	10
C	Max	10	+I	
G	Min	10	+I	
H	Max	9	9	9
G	Min	10	9	9
C	Max	10	+I	10
B	Min	-I	10	
J	Max	-I	10	
K	Min	-I	10	
L	Max	14	14	14
K	Min	-I	10	10
…				

NODE	TYPE	ALPHA	BETA	SCORE
…				
J	Max	10	10	10
B	Min	-I	10	10
A	Max	10	+I	
Q	Min	10	+I	
R	Max	10	+I	
S	Min	10	+I	
T	Max	15	15	15
S	Min	10	15	15
R	Max	10	+I	
V	Min	10	+I	
W	Max	4	4	4
V	Min	10	4	4
R	Max	10	+I	10
Q	Min	10	10	10
A	Max	10	10	10

Pruned nodes of the tree…



State-of-the-art for deterministic games

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

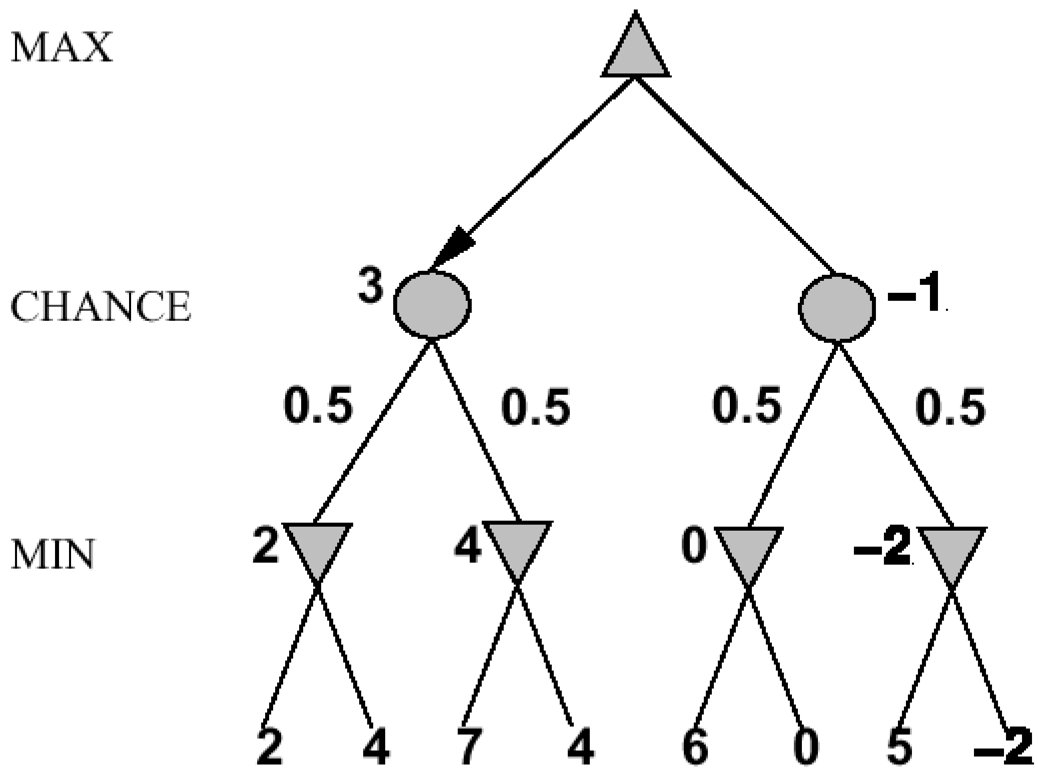
Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Nondeterministic games

E..g, in backgammon, the dice rolls determine the legal moves

Simplified example with coin-flipping instead of dice-rolling:



Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play

Just like MINIMAX, except we must also handle chance nodes:

...
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
...

A version of α–β pruning is possible
but only if the leaf values are bounded. Why??

Because value of chance nodes is based on averages; and you need the bounds for averages without looking at nodes

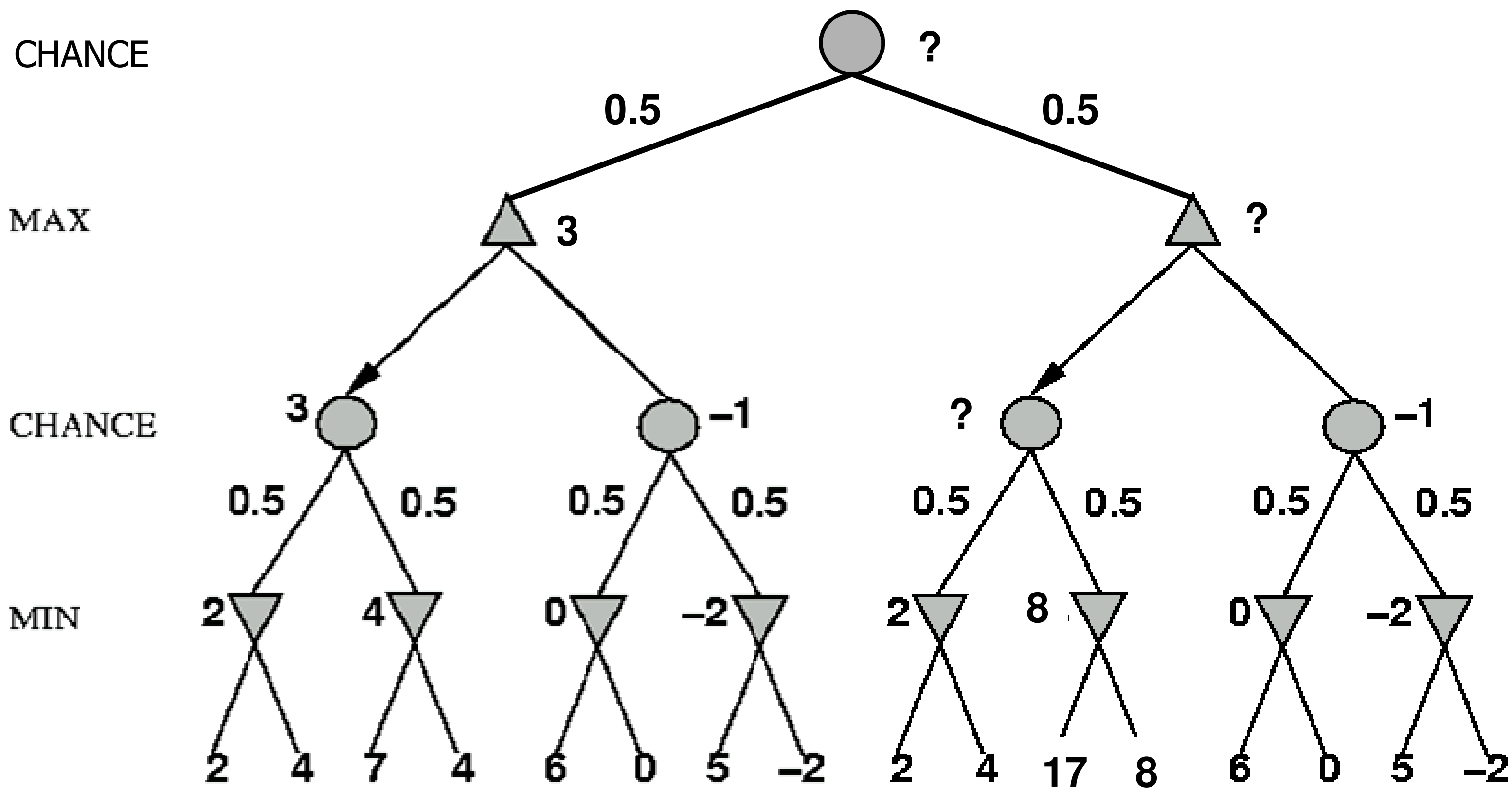
Remember: Minimax algorithm

```
function MINIMAX-DECISION(game) returns an operator

    for each op in OPERATORS[game] do
        VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest VALUE[op]
```

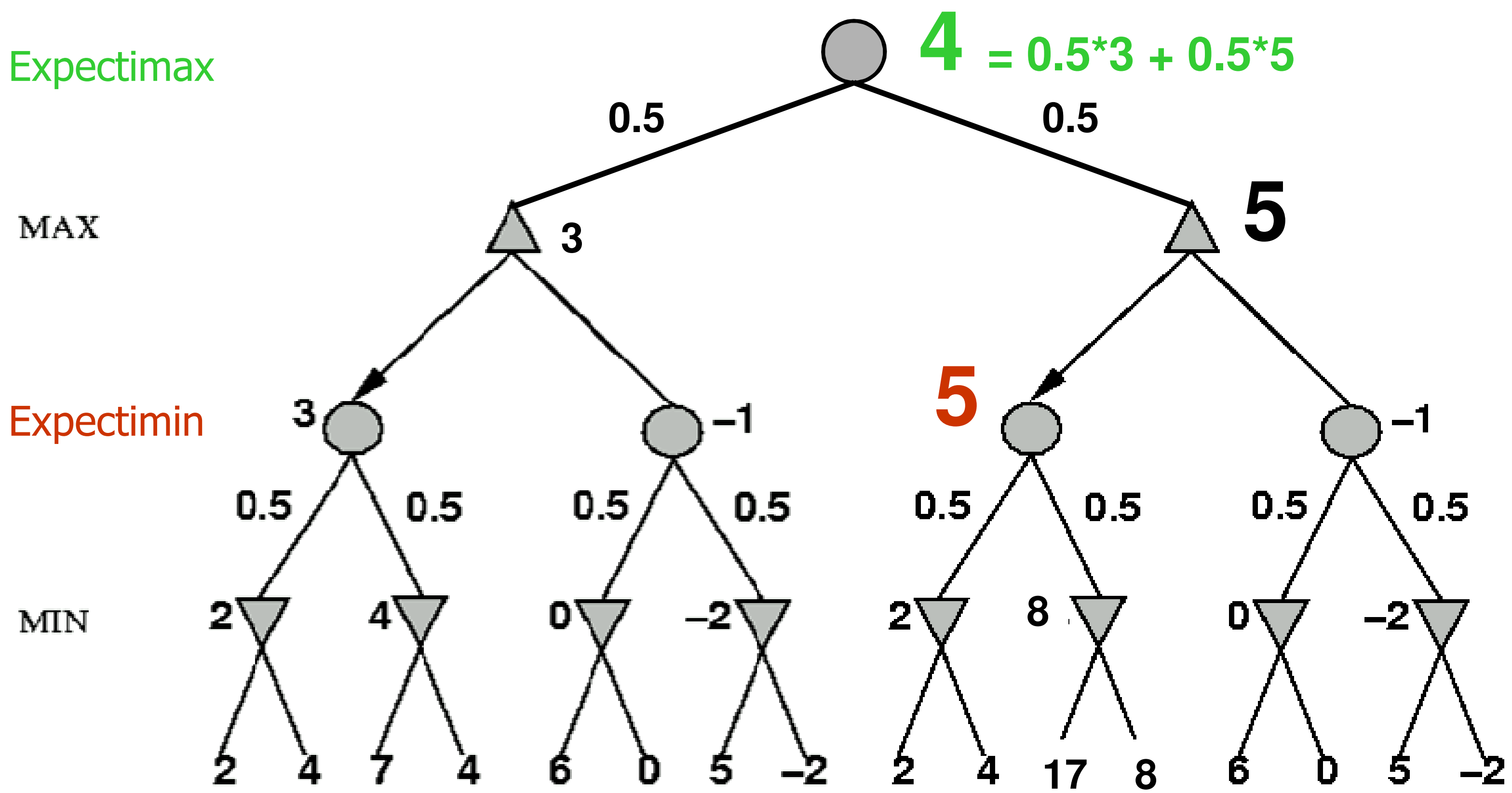
```
function MINIMAX-VALUE(state, game) returns a utility value

    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Nondeterministic games: the element of chance

expectimax and **expectimin**, expected values over all possible outcomes

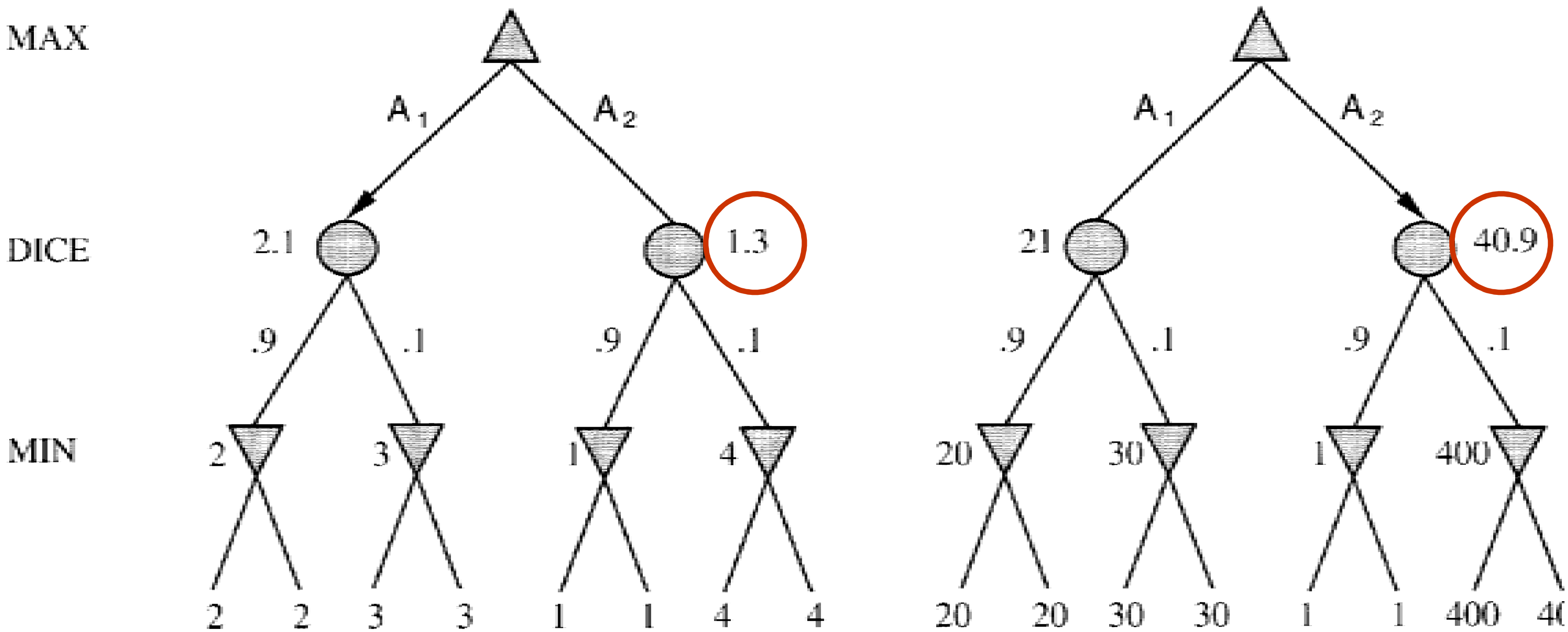


Nondeterministic games: the element of chance



Evaluation functions: Exact values DO matter once you have chance nodes!

Order-preserving transformation do not necessarily behave the same!



State-of-the-art for nondeterministic games

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

◊ perfection is unattainable ⇒ must approximate

◊ good idea to think about what to think about

◊ uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design

Exercise: Game Playing

Consider the following game tree in which the evaluation function values are shown below each leaf node. Assume that the root node corresponds to the maximizing player. Assume the search always visits children left-to-right.

(a) Compute the backed-up values computed by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.

(b) Compute the backed-up values computed by the alpha-beta algorithm. What nodes will not be examined by the alpha-beta pruning algorithm?

(c) What move should Max choose once the values have been backed-up all the way?