Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

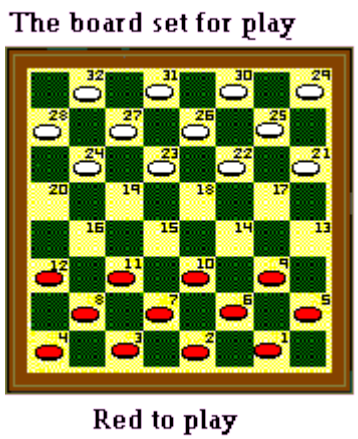
Universität zu Lübeck

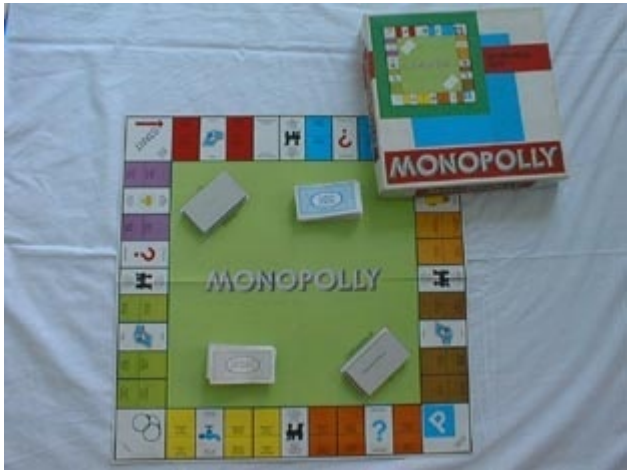
Institut für Informationssysteme

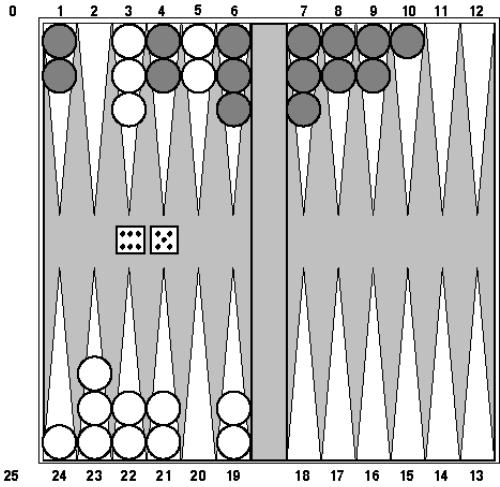
Tanya Braun (Übungen)

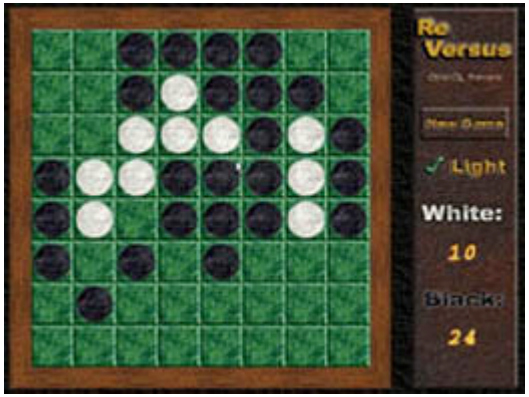
sowie viele Tutoren

Suchgraphen für 2-Personen-Nullsummenspiele





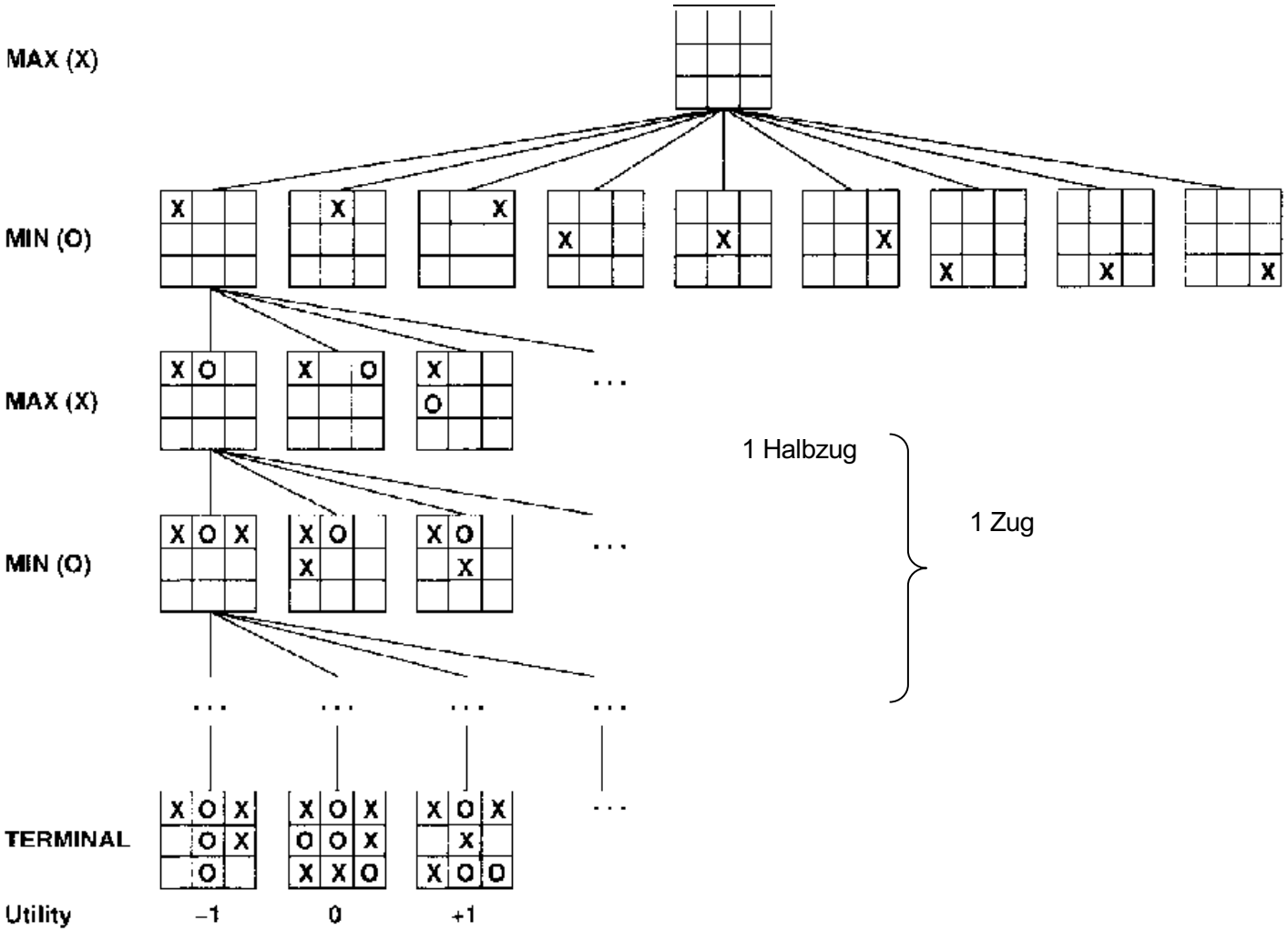




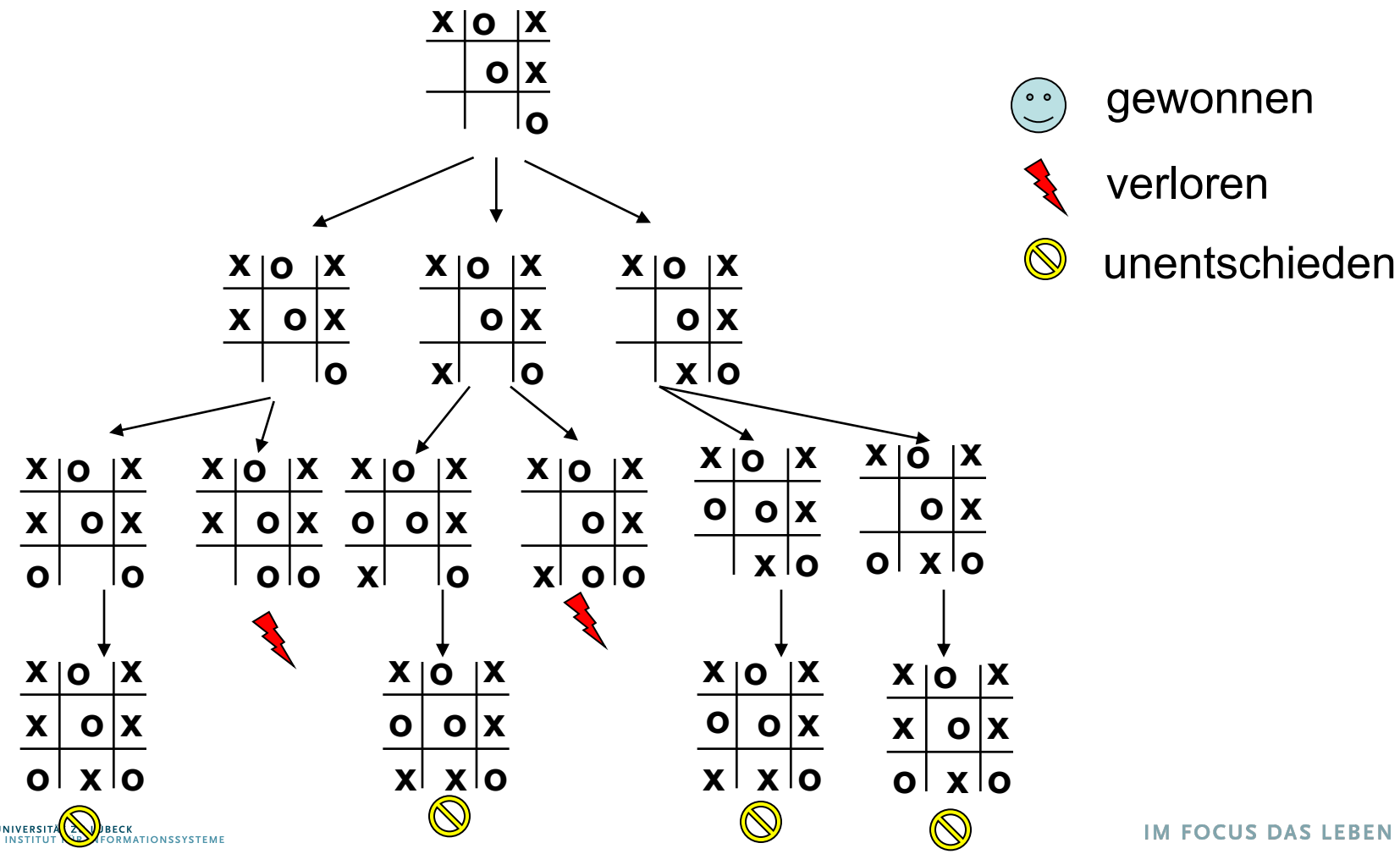
Zweipersonen-Spiele

- Ein Spiel als Suchproblem (z.B. Matt in drei Zügen):
  - Anfangszustand: Brettposition und erster Spieler
  - Operatoren (Züge): In Brettposition legale Züge
  - Endzustand: Bedingungen für Spielende
  - Nützlichkeitsfunktion: Num. Wert für Ausgang des Spiels
    -1, 0, 1
    für verloren, unentschieden, gewonnen
    (Payoff- oder Utility-Funktion)

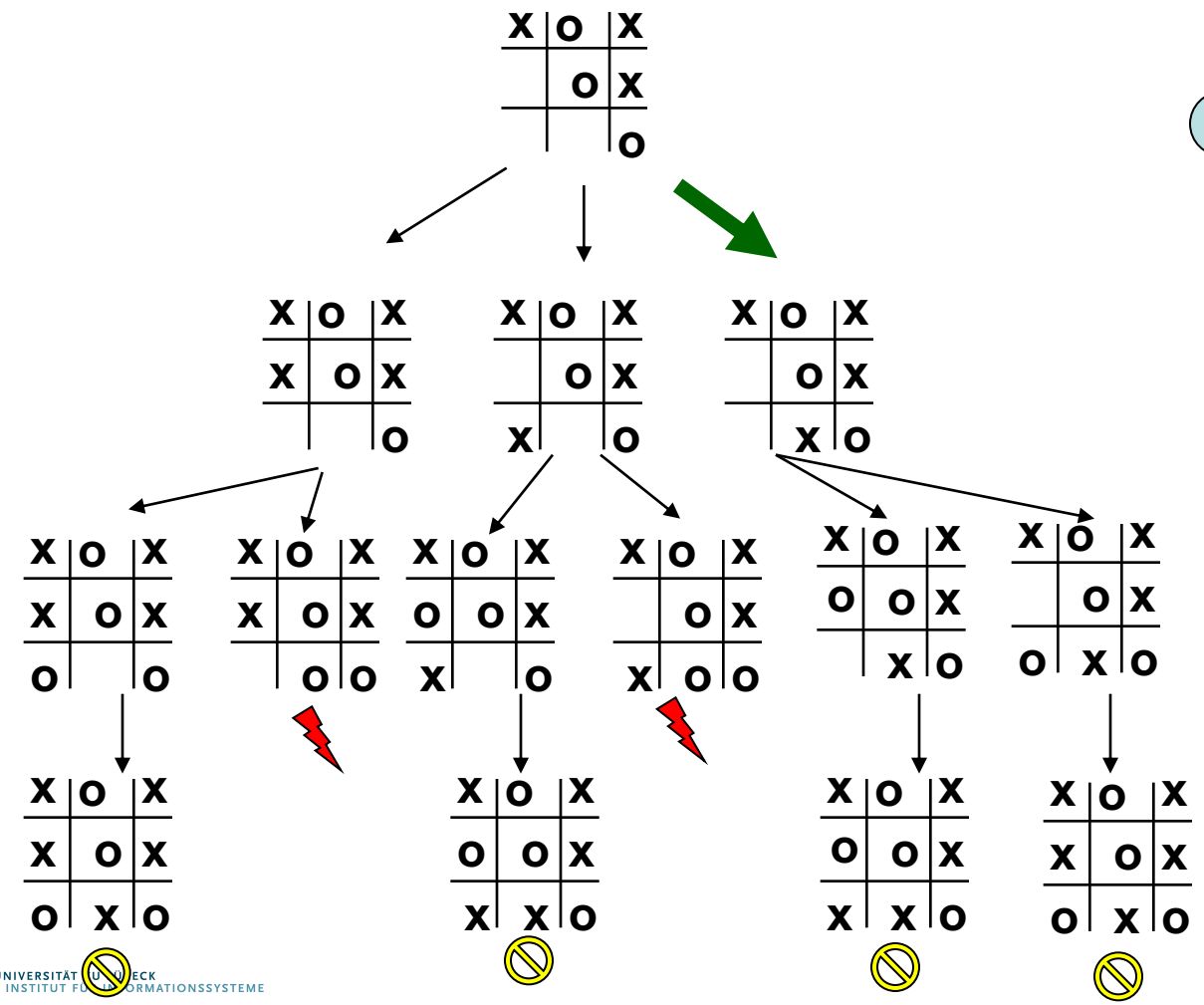
Beispiel: Tic-Tac-Toe

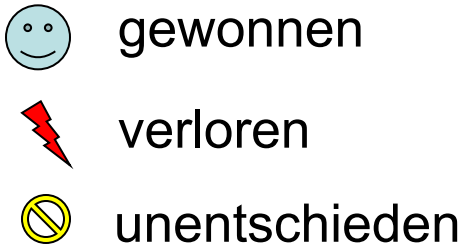


Ein Teilbaum

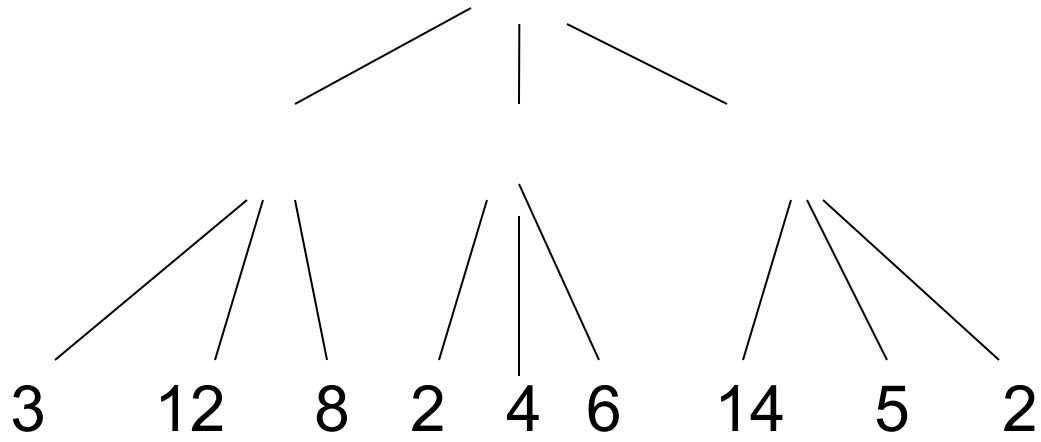


Was ist ein guter Zug?



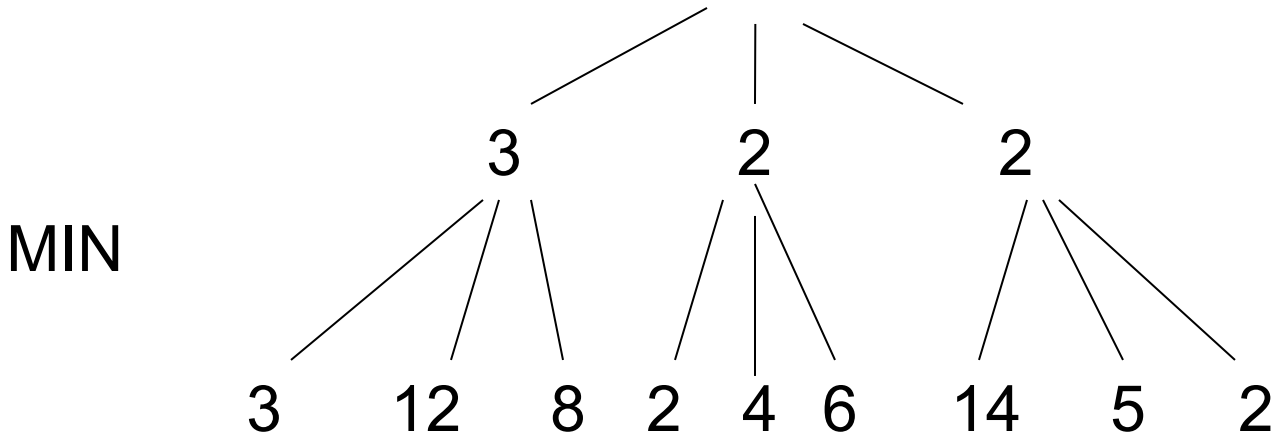


Minimax



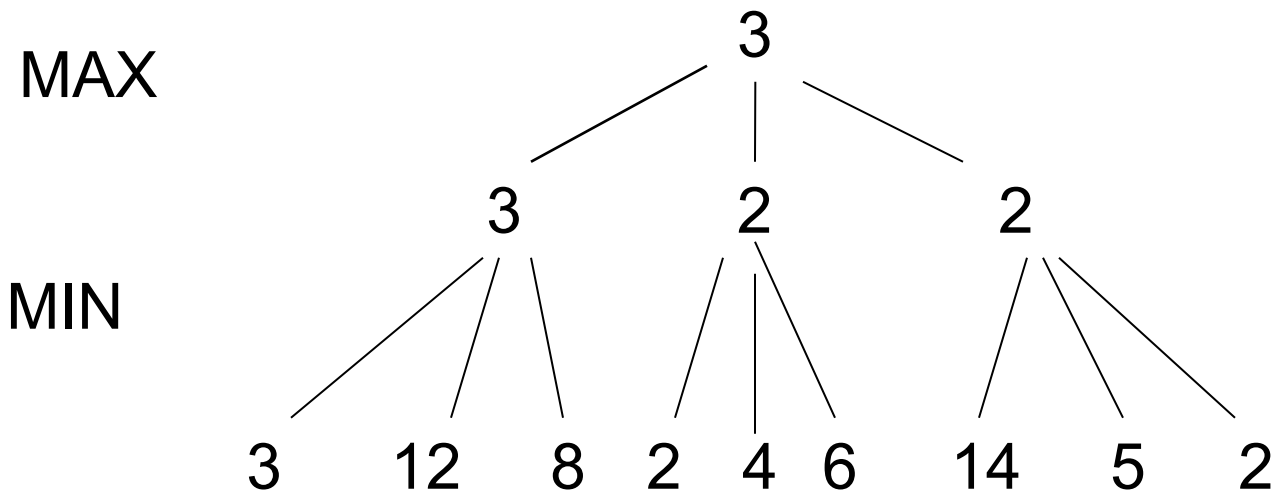
- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

Minimax



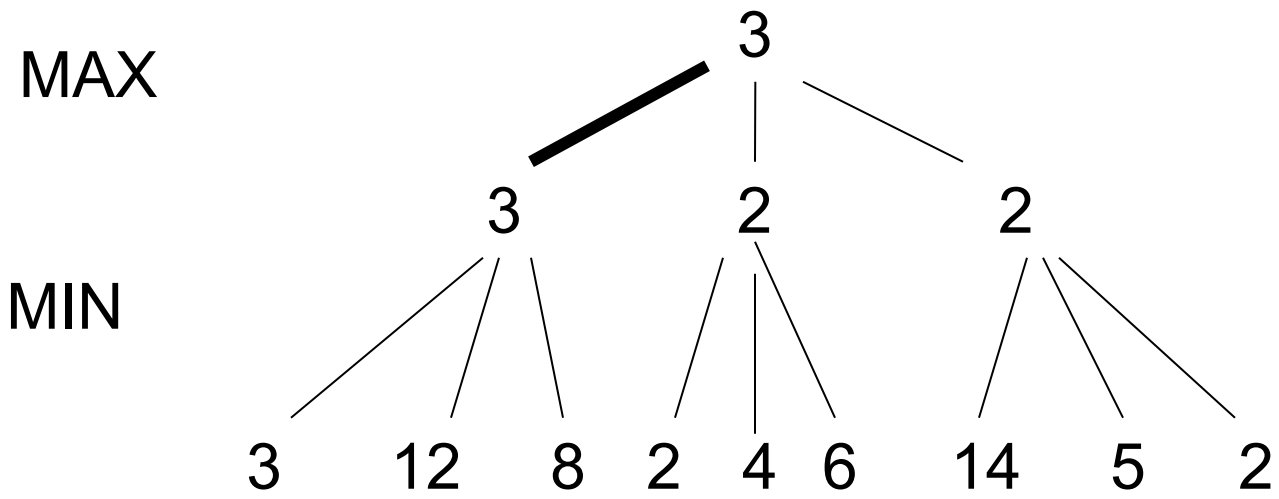
- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

Minimax



- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

Minimax



- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

Minimax: Rekursive Implementation

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(state)} MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(state, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state, a)))
  return v
```

Vollständig: ?
Optimal: ?

Zeitkomplexität: ?
Platzkomplexität: ?

Minimax: Rekursive Implementation

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(state)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(state, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state, a)))
  return v
```

Vollständig: Ja, für endl. Zust.raum **Zeitkomplexität:** $O(b^m)$

Optimal: Ja **Platzkomplexität:** O(bm) (= DFS)

m = Suchtiefe bis Endzustände erreicht

b = Verzweigungsfaktor (braching)

Spielen als naive Suche?

- **Komplexität:** Viele Spiele haben einen großen Suchraum
  - **Schach:** *b = 35, m=100* **#***Knoten= 35* ^{100}
    Falls Betrachtung eines Knotens 1 ns benötigt,
    braucht jeder Zug ***10*** ^{50} **Jahrhunderte**
    zur Berechnung

- Endzustände können in vertretbarer Zeit
  nicht alle generiert werden

Algorithmen für perfektes Spiel: John von Neumann, **1944**
Endlicher Horizont, approximative Zustandsbewertung: Zuse **1945** , Shannon **1950** , Samuel **1952-57**
Suchbaumabschneidungen zur Einsparung von Zeit: McCarthy **1956**

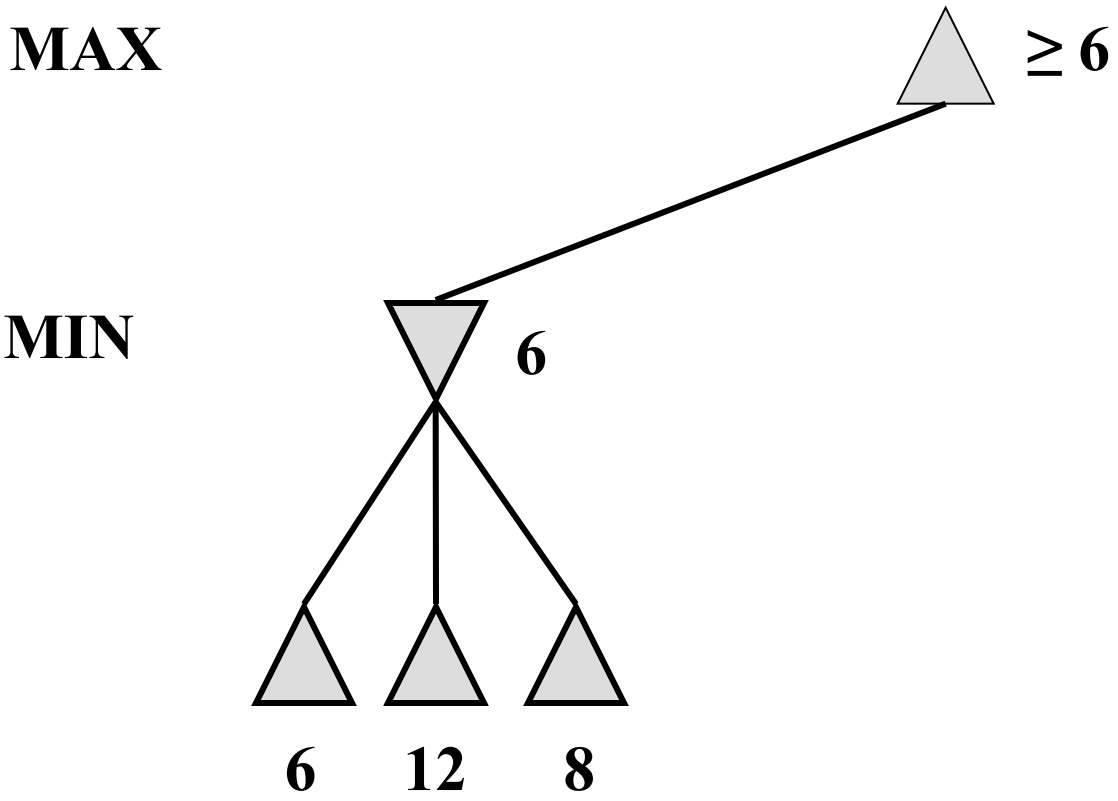
Lösungsansätze

Reduktion des Ressourcenproblems (Zeit, Speicher):

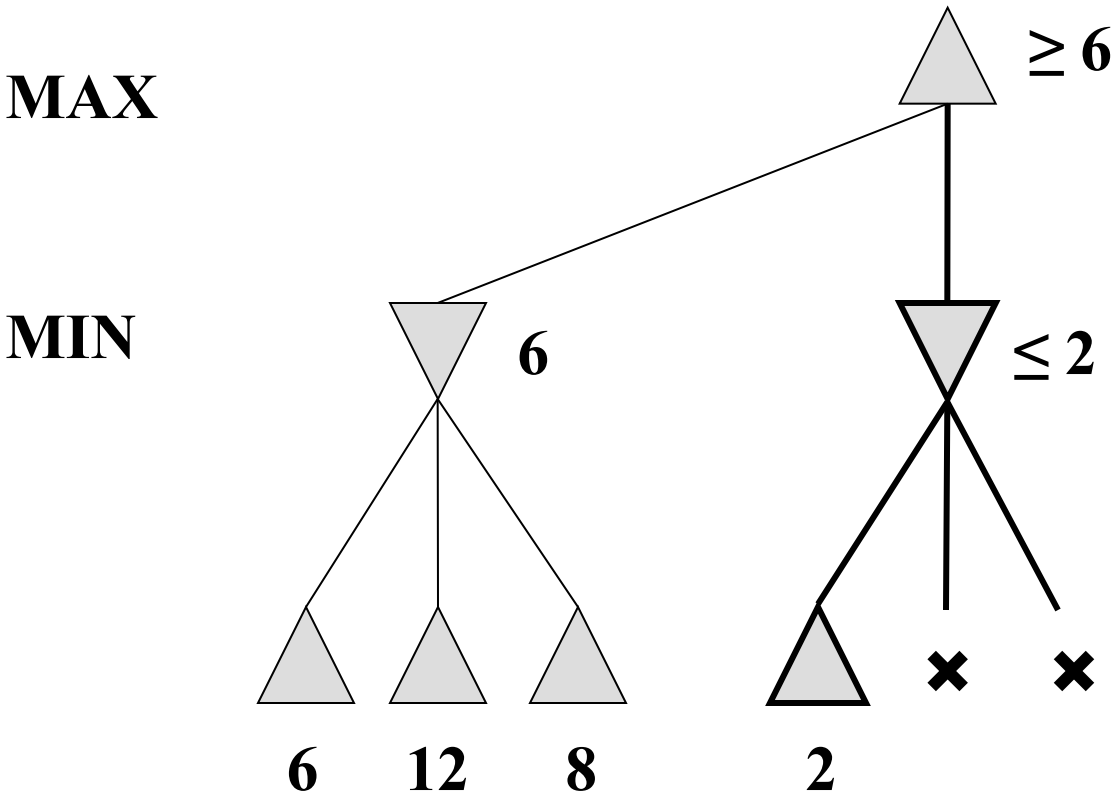
1. **Suchraumbeschneidung (Pruning):** macht Suche schneller durch Entfernen von Teilen des Suchbaums, in denen beweisbar keine bessere als die aktuell beste Lösung gefunden werden kann

2. **Bewertungsfunktion:** Heuristik zur Bewertung der Nützlichkeit eines Spielzustands (Knoten) ohne vollständige Suche

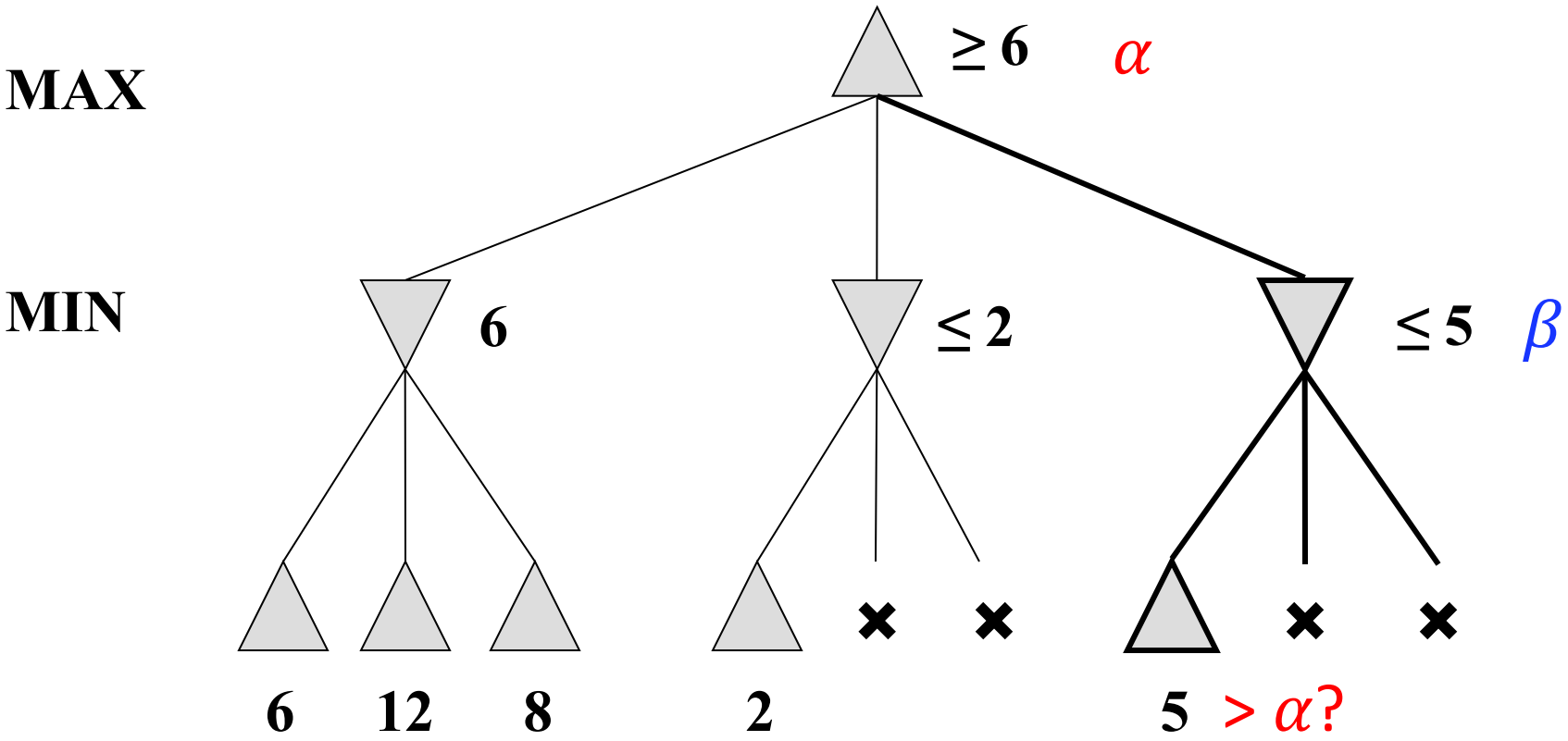
Pruning: Beispiel



Pruning: Beispiel



α-β-Pruning



MINIMAX(root) = max(min(6,12,8),min(2,a,b),min(5,b,d))
= max(6,z,y) wobei z=min(2,a,b)≤ 2 und y=min(5,b,d) ≤ 5
= 6

Suche mit Pfadbeschneidung

function ALPHA-BETA-SEARCH (state)

α := -∞ ; β := ∞

if TERMINAL-TEST(state) then

return ⊥

else

return

$\text{argmax}_{a \in \text{ACTIONS(state)}}$ { v := MIN-VALUE(RESULT(state, a), α, β)

α := max(α, v)

v }

Von mehreren Forschern **um 1960** erfunden.

MAX-VALUE und MIN-VALUE mit α-β-Pruning

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($state$,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

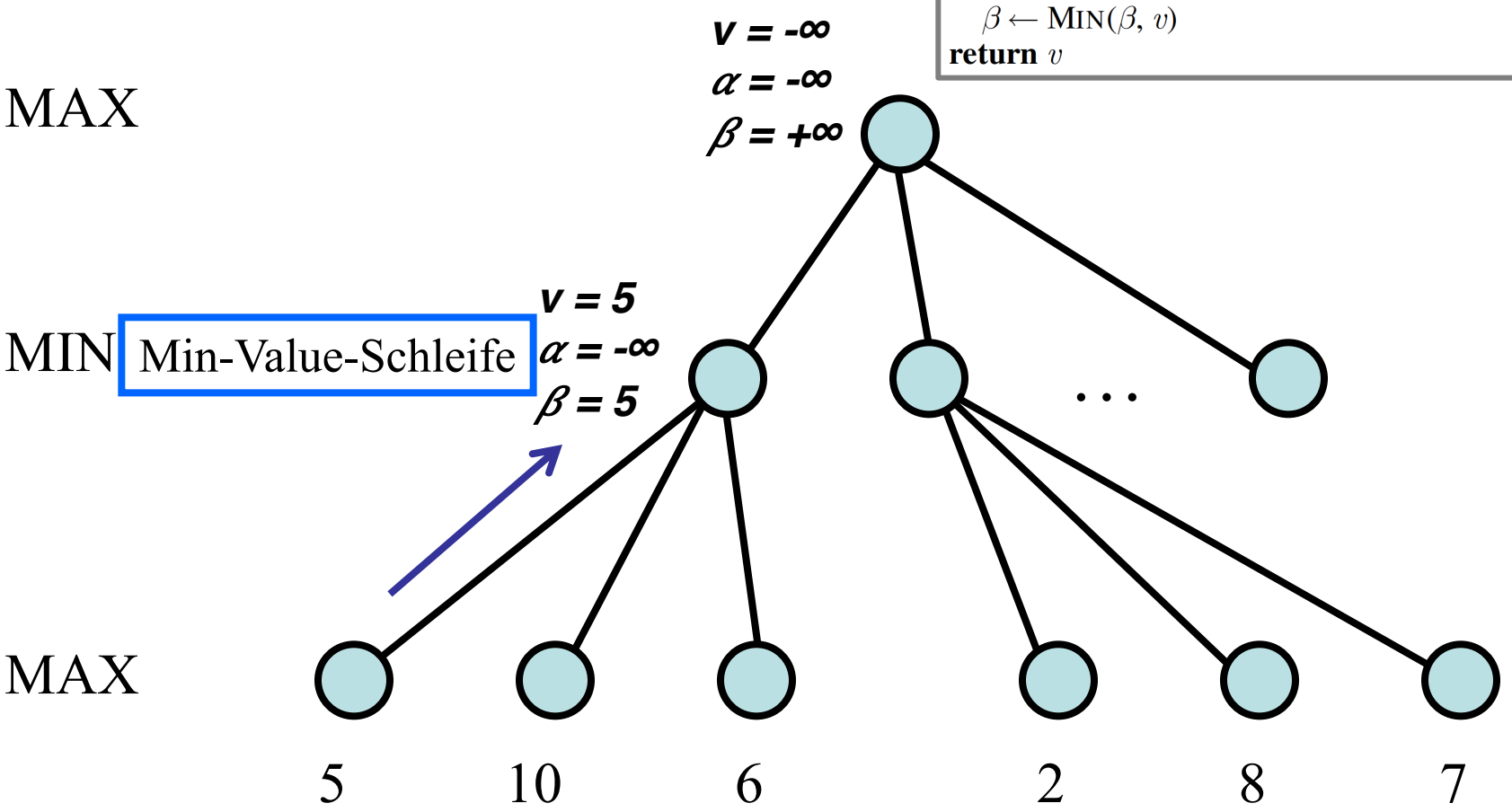
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT($state$,a), α, β))
    if $v \leq \alpha$ **then return**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

Darstellung entnommen aus:
Russell, St., Norvig, P., Artificial Intelligence:
A Modern Approach, 3rd Ed., **2010**

α-β-Suchalgorithmus

In Min-Value:

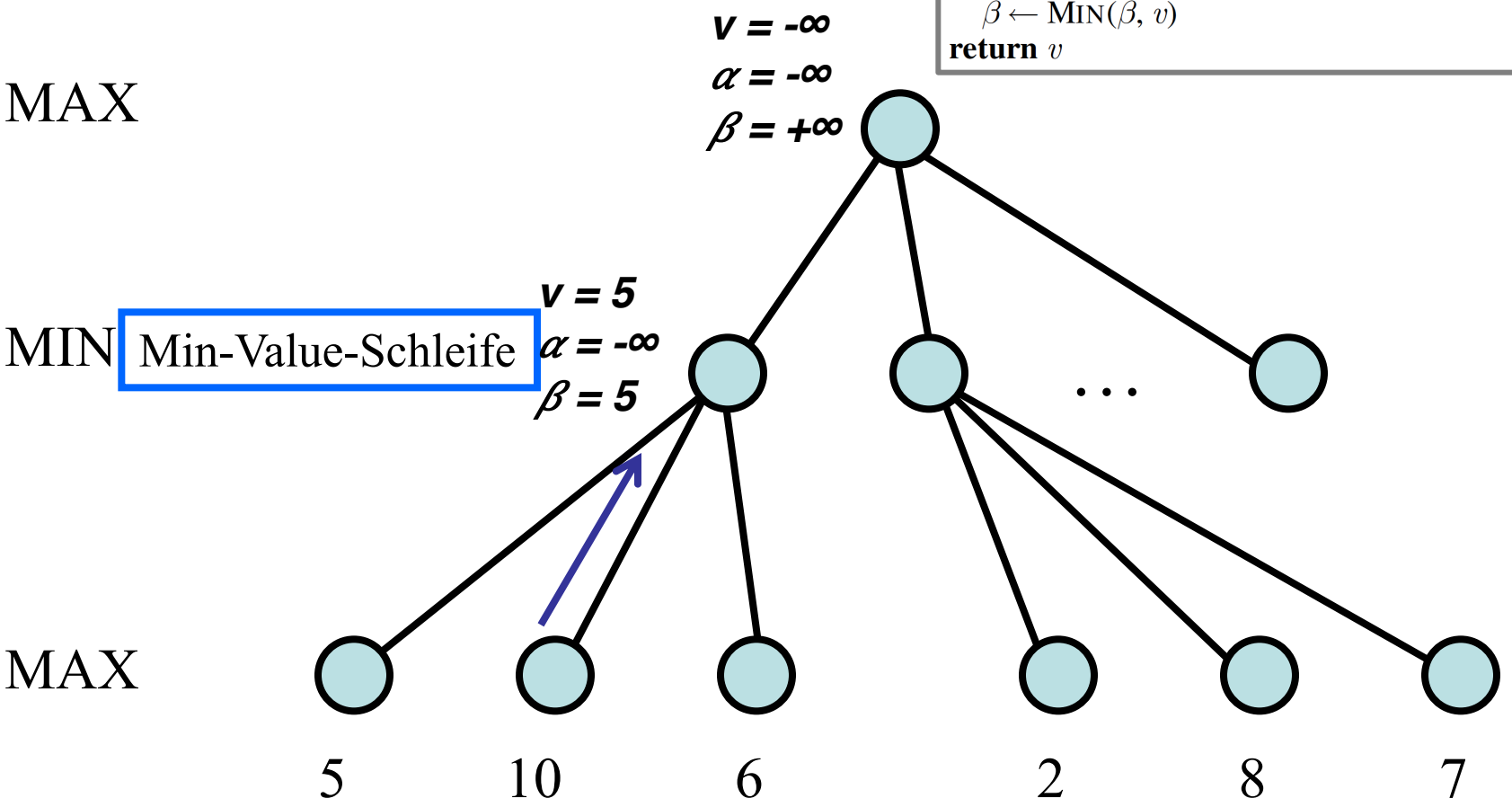
```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Min-Value:

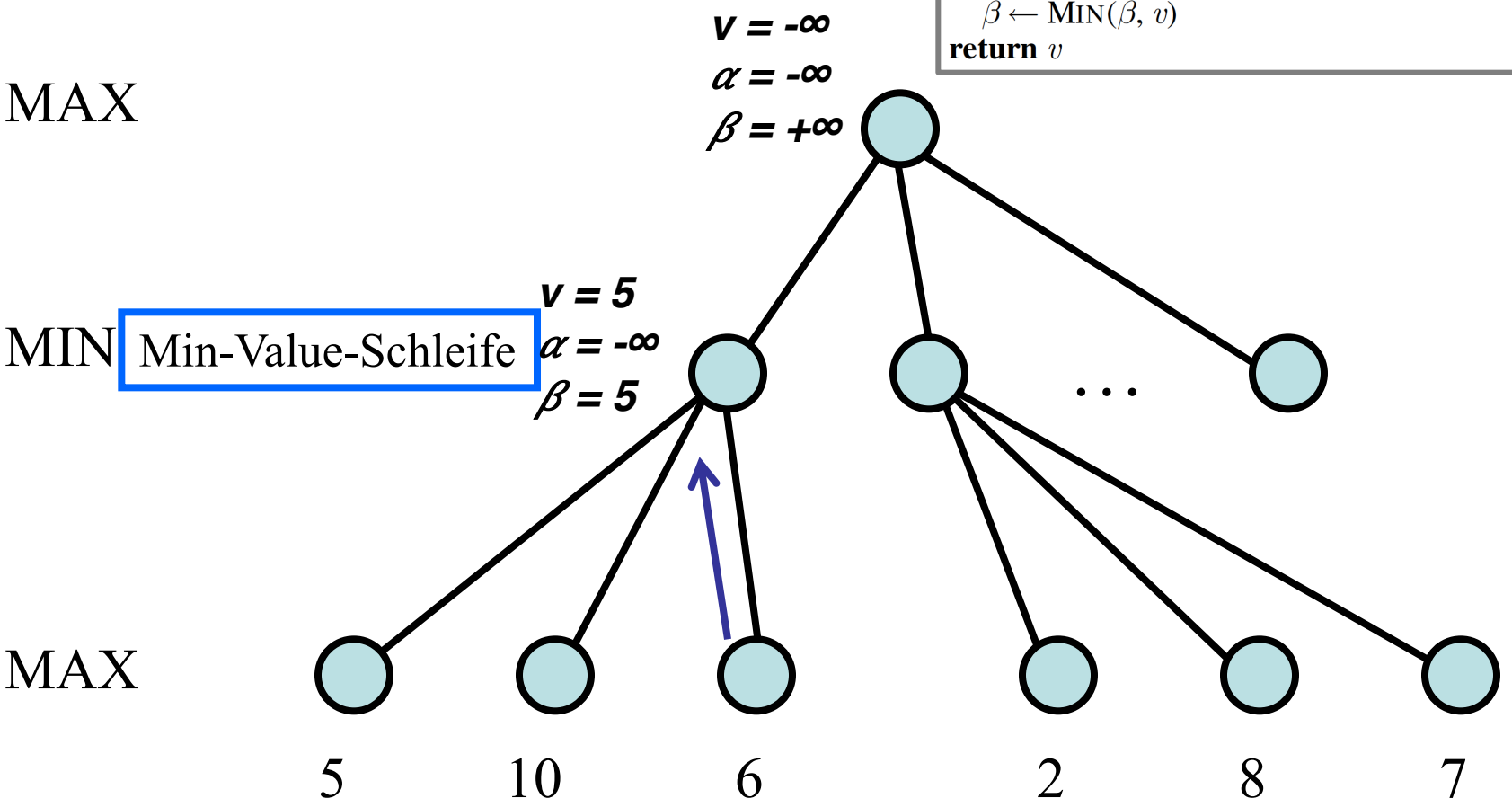
```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Min-Value:

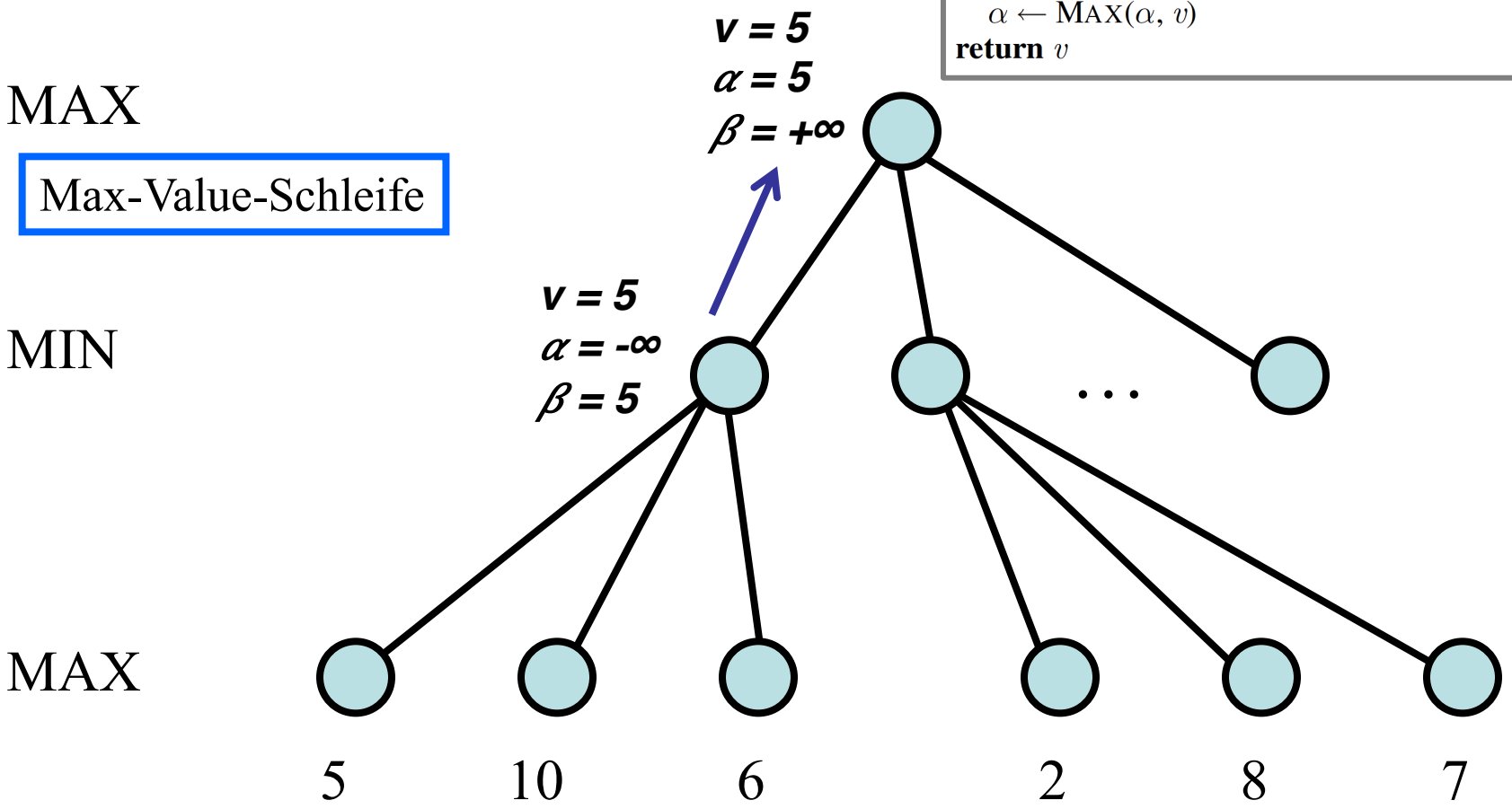
```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Max-Value:

$v \leftarrow -\infty$
for each a **in** ACTIONS($state$) **do**
  $v \leftarrow$ MAX(v, MIN-VALUE(RESULT($state$,a), α, β))
  if $v \geq \beta$ **then return** v
  $\alpha \leftarrow$ MAX(α, v)
return v



α-β-Suchalgorithmus

In Max-Value:

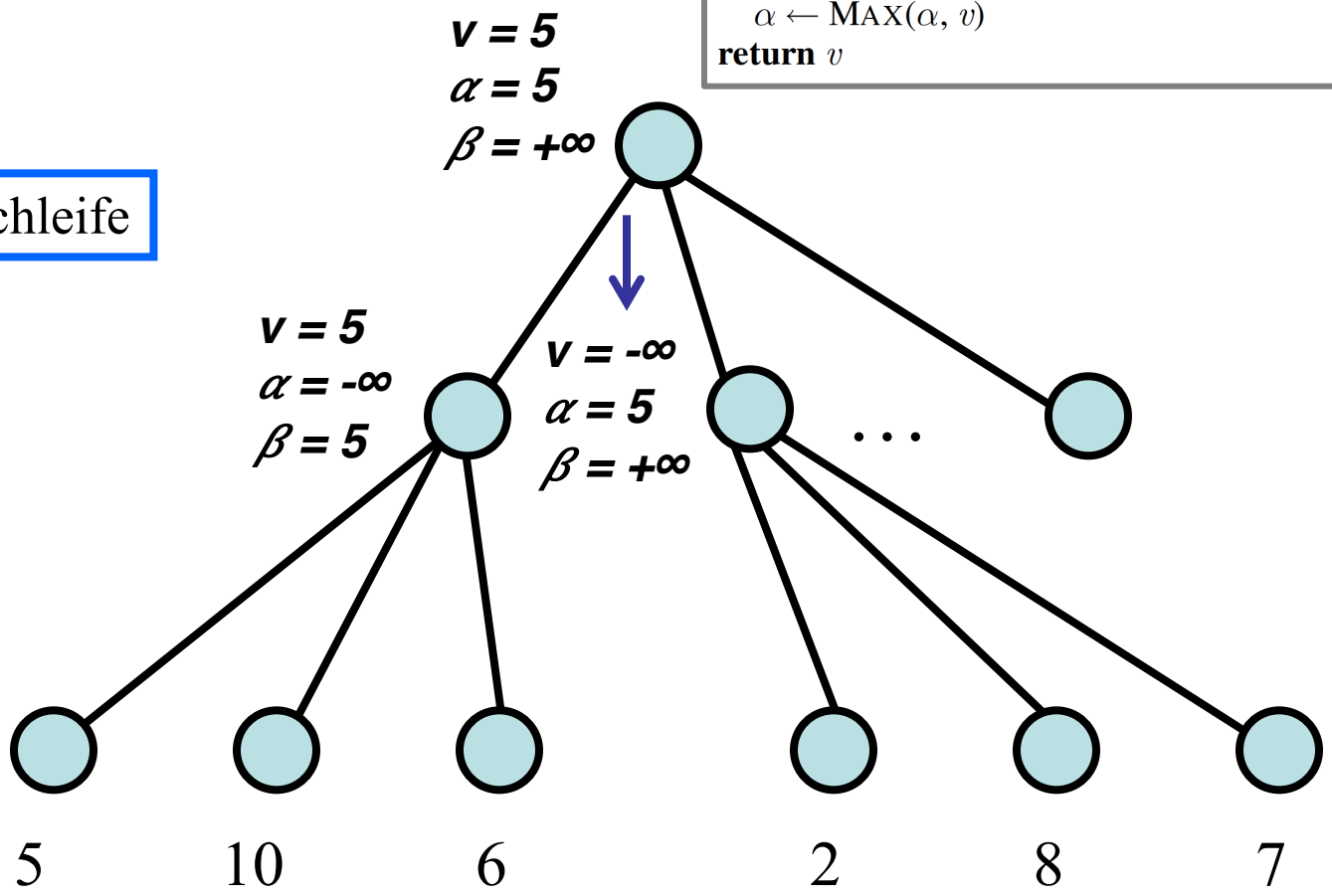
$v \leftarrow -\infty$
for each a **in** ACTIONS($state$) **do**
  $v \leftarrow$ MAX(v, M IN-VALUE(RESULT($state$,a), α, β))
  if $v \geq \beta$ **then return** v
  $\alpha \leftarrow$ MAX(α, v)
return v

MAX

Max-Value-Schleife

MIN

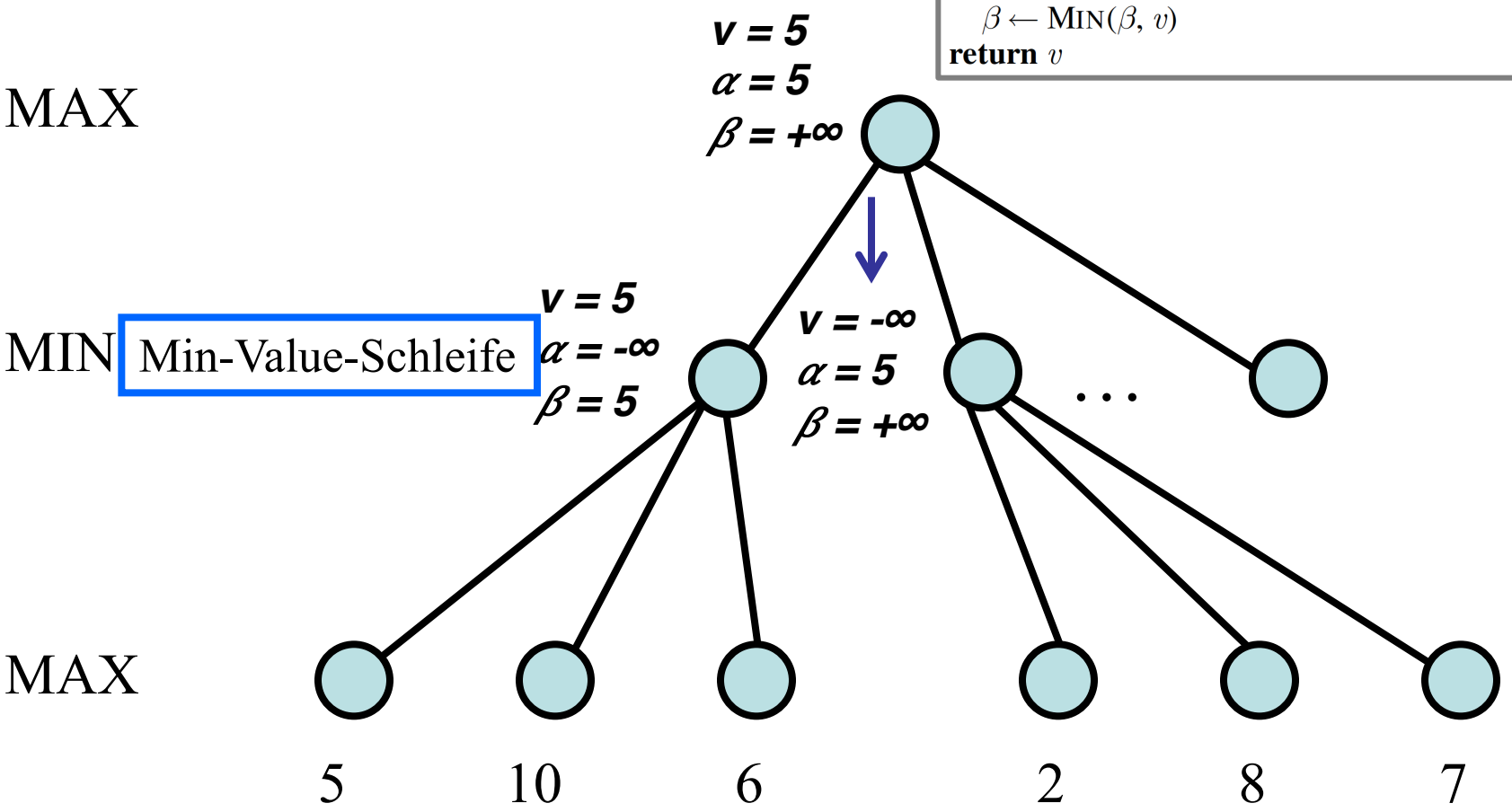
MAX



α-β-Suchalgorithmus

In Min-Value:

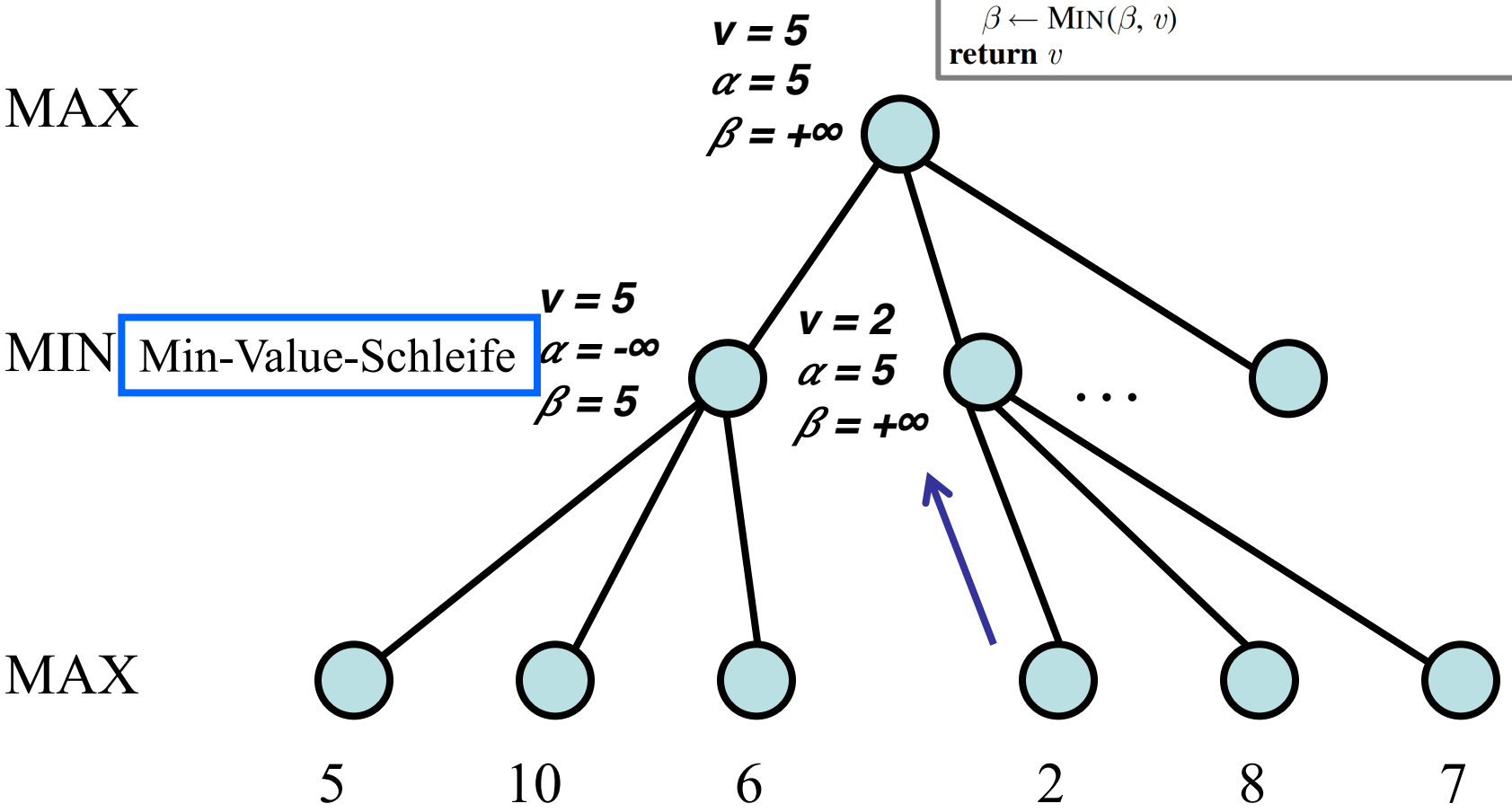
```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Min-Value:

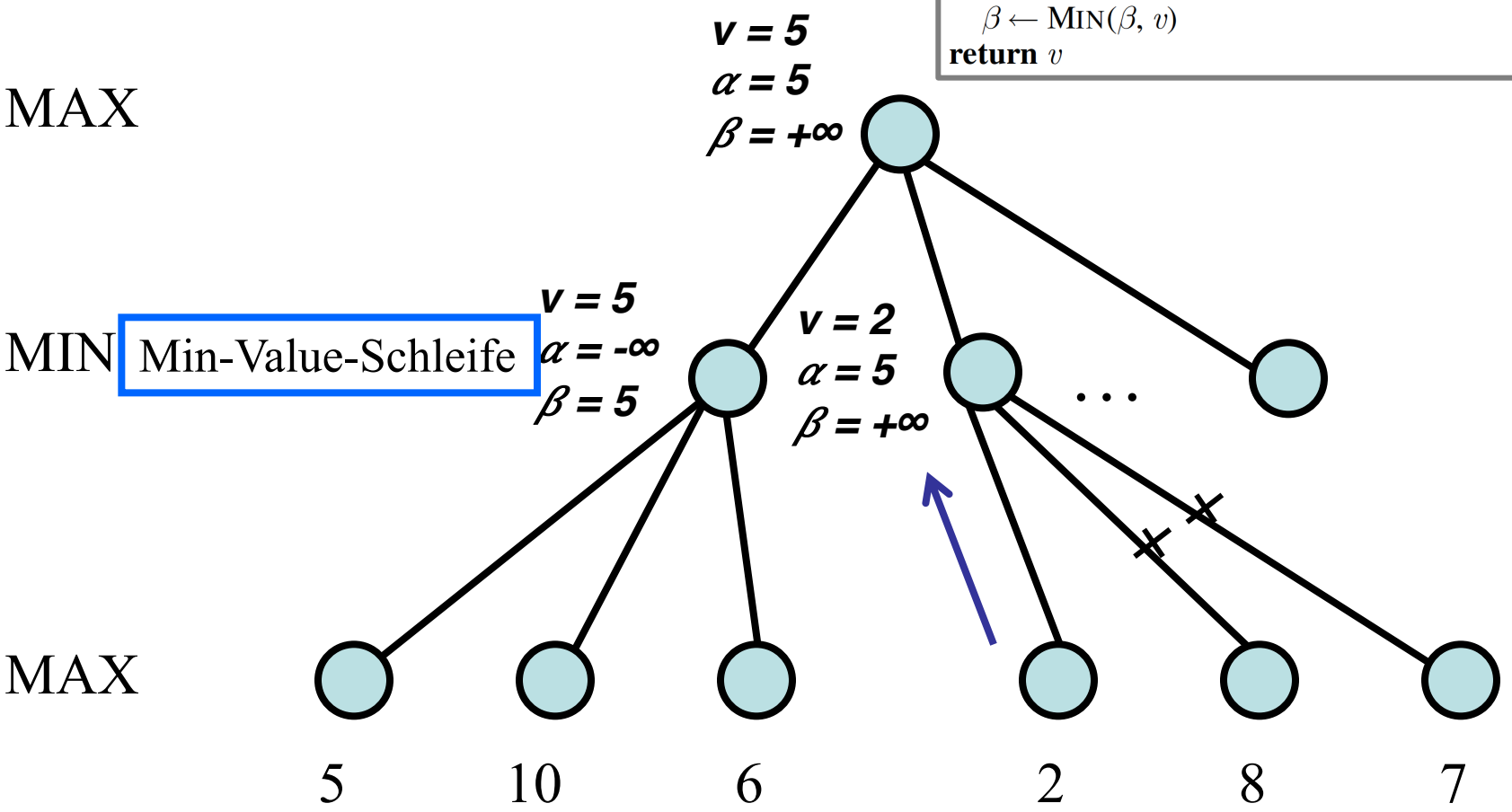
```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Min-Value:

```
v ← +∞
for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
return v
```



α-β-Suchalgorithmus

In Max-Value:

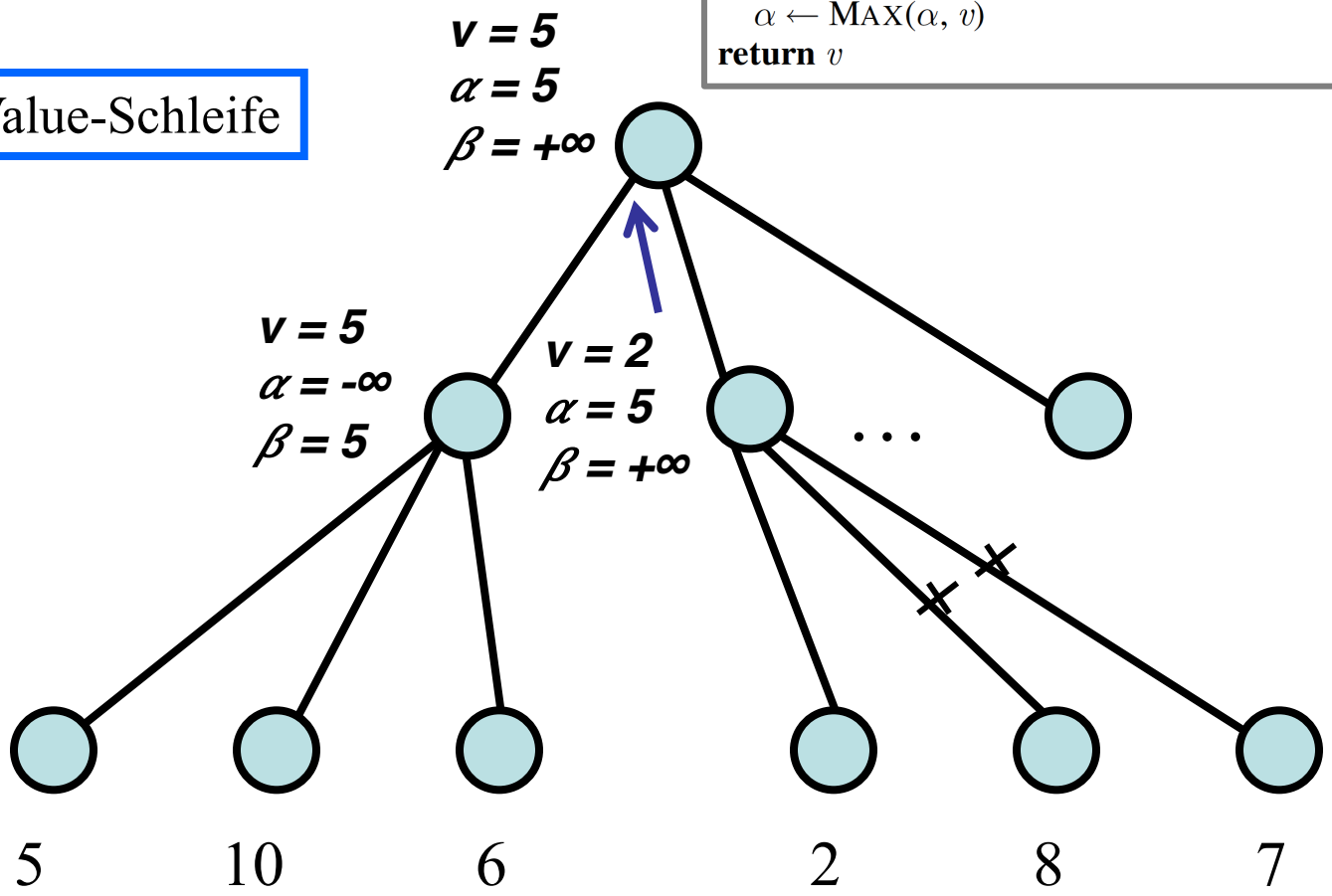
$v \leftarrow -\infty$
for each a **in** ACTIONS($state$) **do**
  $v \leftarrow$ MAX(v, MIN-VALUE(RESULT($state$,a), α, β))
  if $v \geq \beta$ **then return** v
  $\alpha \leftarrow$ MAX(α, v)
return v

MAX

Max-Value-Schleife

MIN

MAX



2. Zugbewertung ohne erschöpfende Suche

- Der Minimax-Algorithmus generiert den vollständigen Spielsuchraum, während der α-β–Algorithmus große Teile davon abschneidet (ohne die beste Lösung zu verlieren)
- Trotzdem: Vollständige Suche meist zu aufwendig
- Idee: Propagierung von unten nach oben durch Anwendung einer Bewertungsfunktion ersetzen
- **Bewertungsfunktion:** Evaluiert der Wert von Zuständen durch Heuristiken und beschneidet dadurch auch den Suchraum

- **New MINIMAX:**
  - CUTOFF-TEST: Ersetzt die Terminierungsbedingung
  - EVAL: Bewertungsfunktion als Ersatz für die Nützlichkeitsfunktion

Bewertungsfunktion: Beispiel

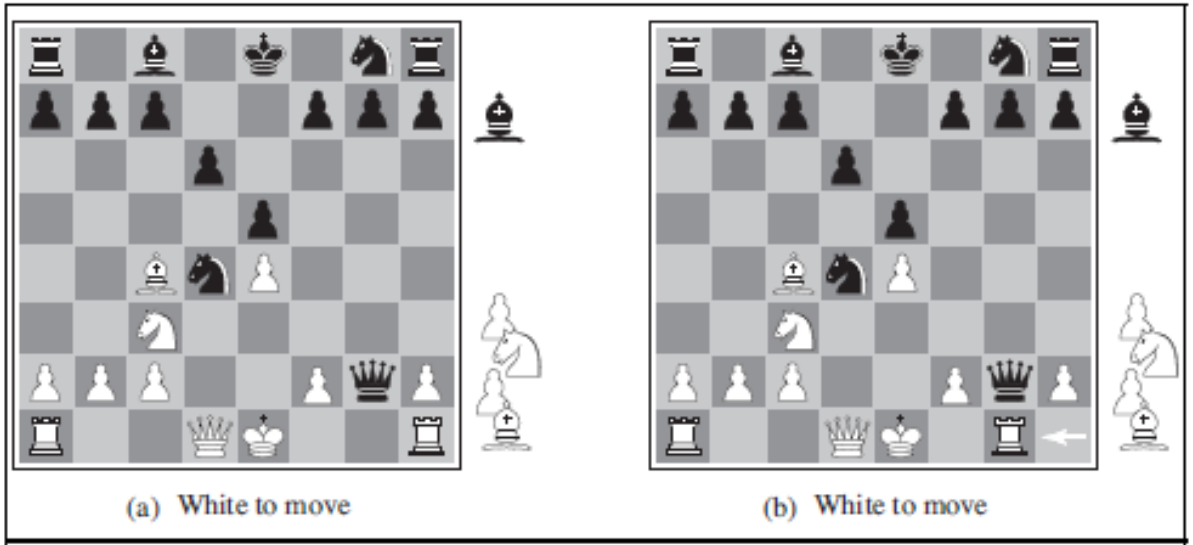


Figure 5.8 **FILES: figures/chess-evaluation3.eps (Tue Nov 3 16:22:33 2009).** Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

- **Gewichtete Linearkombination:** Kombination von mehreren Heuristiken

$$f = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

- Beispiel: f_1= Bewertung #Bauern, f_2 = Bewertung #Türme, …
  w_1 = Gewicht #Bauern, w_2 = Gewicht #Türme
  (Gewichte können von der Suchtiefe abhängen)

Max-Value mit CUTOFF und EVAL

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*

  if CUTOFF-TEST(*state*, *depth*) **then return** EVAL(*state*)

  $v \leftarrow -\infty$

  for each a **in** ACTIONS(*state*) **do**

    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(*state*,a), α, β))

    if $v \geq \beta$ **then return** v

    $\alpha \leftarrow$ MAX(α, v)

  return v

Minimax mit CUTOFF: Brauchbarer Algorithmus?

MinimaxCUTOFF is identisch zu MinimaxVALUE außer

1. TERMINAL? wird ersetzt durch CUTOFF?
2. UTILITY wird ersetzt durch EVAL

Funktioniert das in der Praxis?

$b^m = 10^6$, $b = 35$ ➔ $m = 4$

Annahme: 100 Sekunden Rechenzeit werden pro Zug verwendet und wir können 10^4 Knoten/s bewerten; dann können wir 10^6 Knoten/Zug durchrechnen

4 Halbzüge Vorausschau: Anfängerniveau

8 Halbzüge Vorausschau: meisterhaft

12 Halbzüge Vorausschau: großmeisterhaft (~DeepBlue)

Meister durch Computer geschlagen im Jahr…

- Dame: 1994
- Schach: 1997
- Go: 2016