

---

# Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

**Universität zu Lübeck**

**Institut für Informationssysteme**

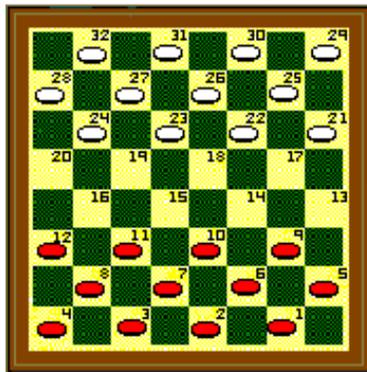
Tanya Braun (Übungen)

sowie viele Tutoren

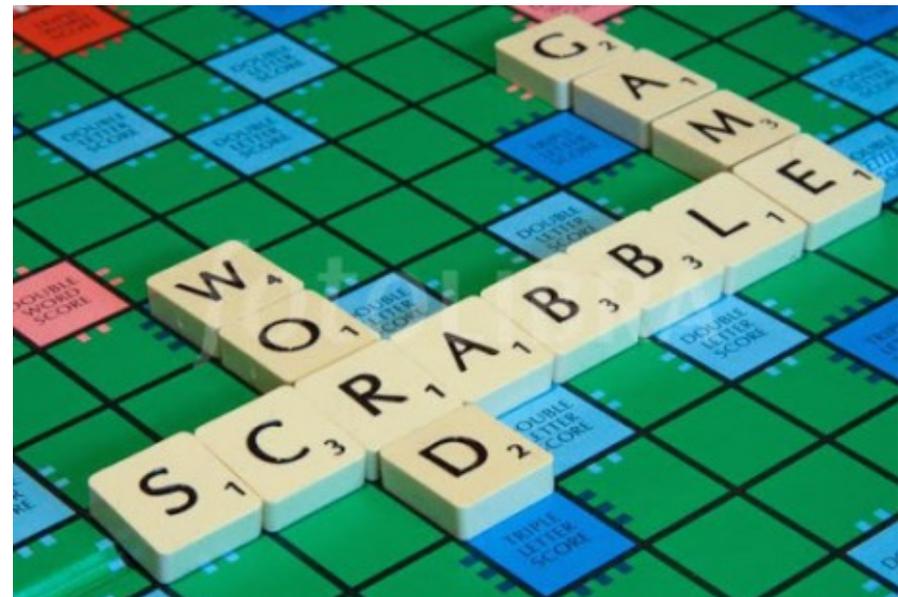
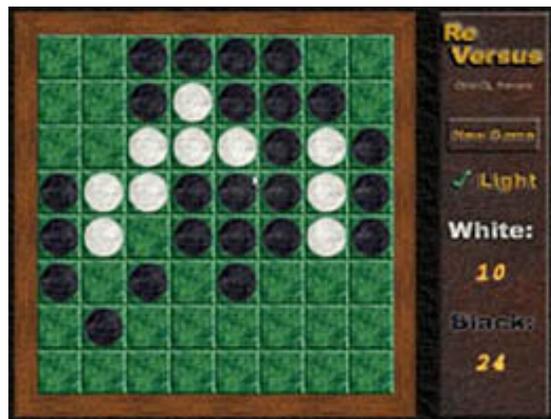
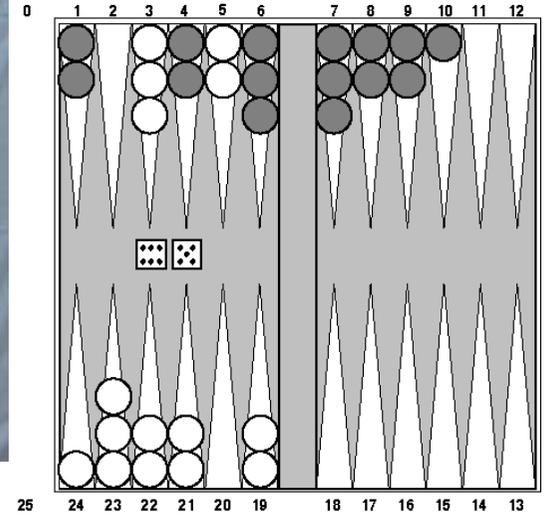
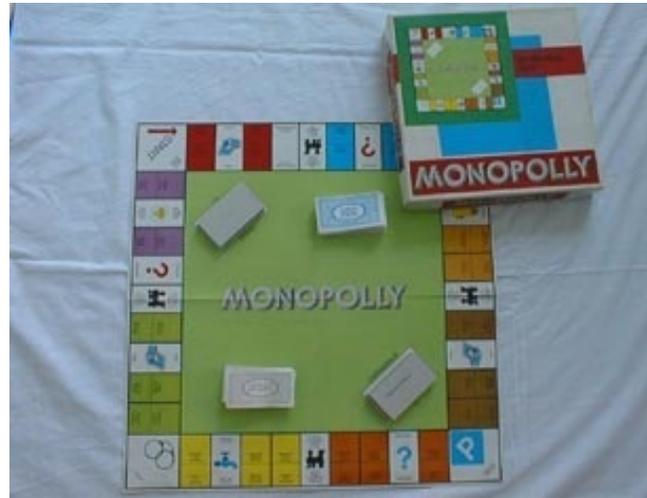


# Suchgraphen für 2-Personen-Nullsummenspiele

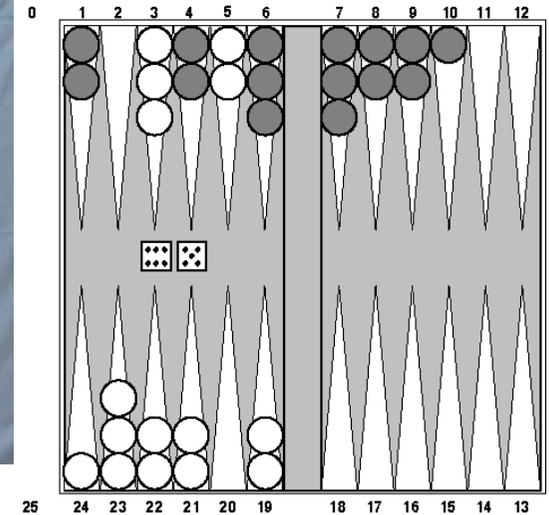
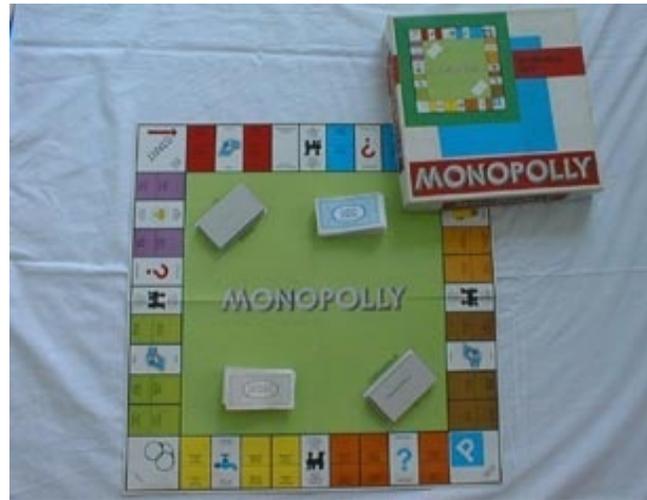
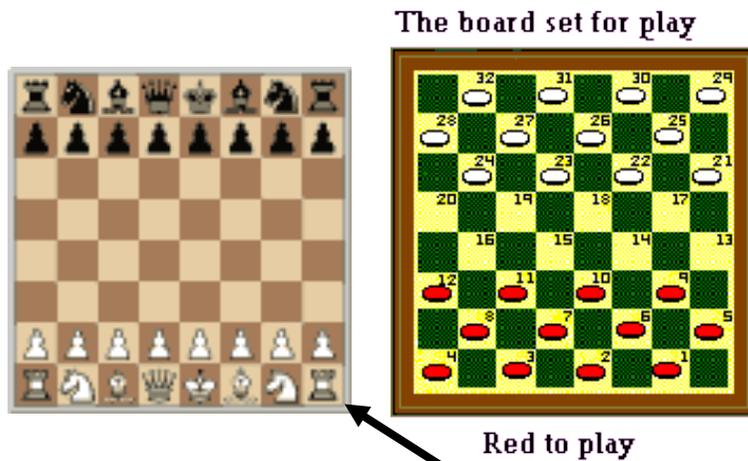
The board set for play



Red to play



# Typen von Spielen



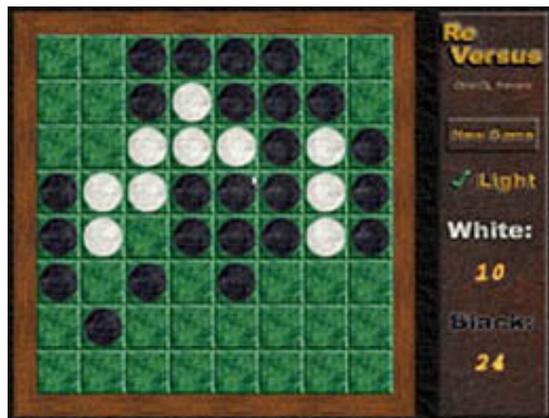
Perfekte Information

Unvollständige Information

deterministisch

zufällig

Schach, Dame, Go, Othello	Backgammon, Monopoly
	Bridge, Poker, Scrabble



# Zweipersonen-Spiele

---

- Ein Spiel als Suchproblem:
  - Anfangszustand: ?
  - Operatoren (Züge): ?
  - Endzustand: ?
  - Nützlichkeitsfunktion: ?



# Zweipersonen-Spiele

---

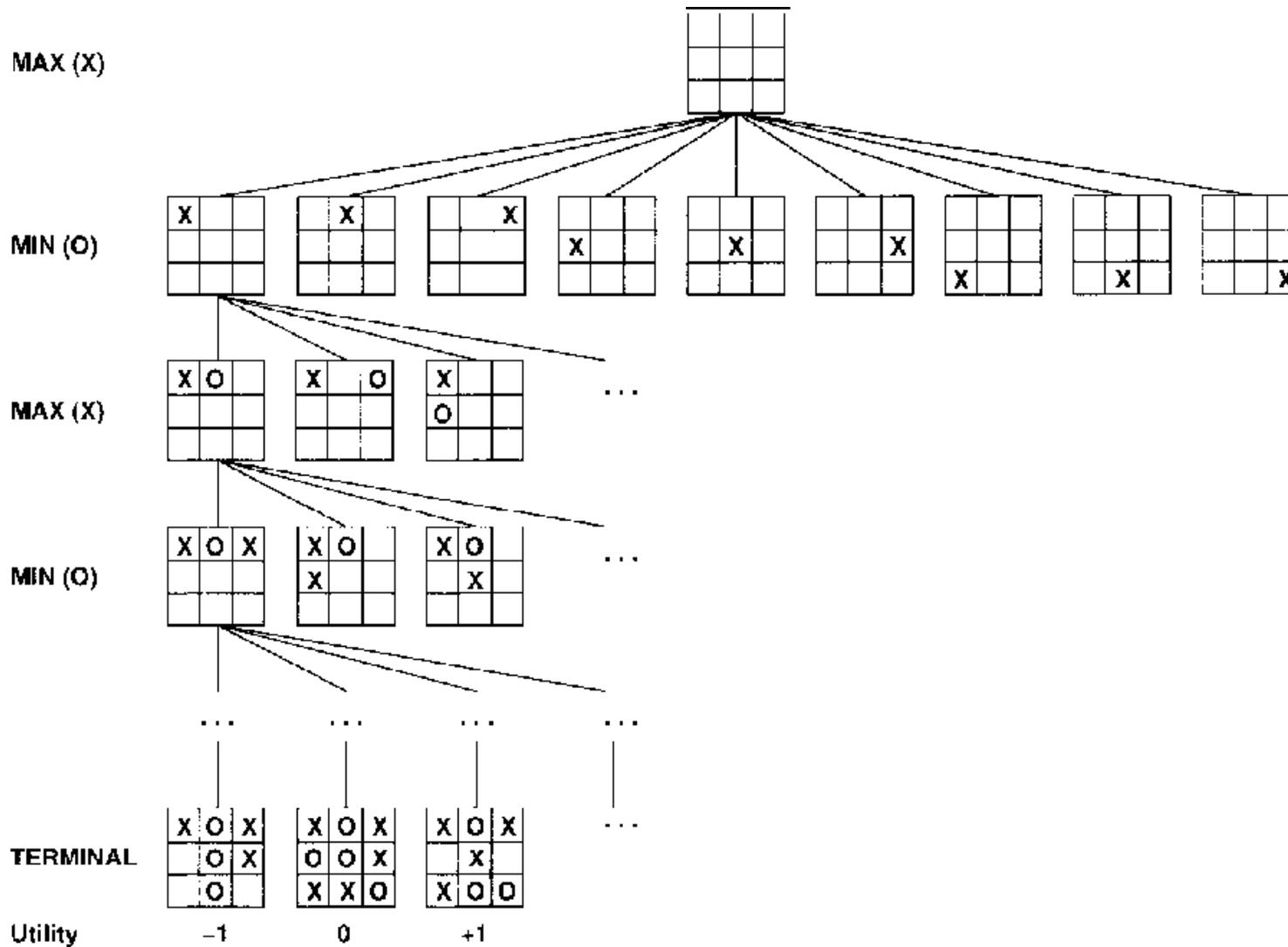
- Ein Spiel als Suchproblem :
  - Anfangszustand: Brettposition und erster Spieler
  - Operatoren (Züge): In Brettposition legale Züge
  - Endzustand: Bedingungen für Spielende
  - Nützlichkeitsfunktion: Num. Wert für Ausgang des Spiels  
-1, 0, 1  
für verloren, unentschieden, gewonnen  
(Payoff- oder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

# Entwurfsmuster Suchraumbeschneidung

---

- Unser Problem: Bestimme besten Zug
- Entscheidung des Problems über Suche in einem Graphen mit der Idee, den...
- Suchraum systematisch zu bescheiden...
- ohne die richtige Lösung zu verlieren
  
- Später:  
Suchraumbeschneidung als Approximation  
aber mit praktikablem Laufzeitverhalten

# Beispiel: Tic-Tac-Toe



# Minimax-Algorithmus

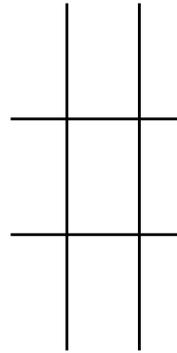
---

- Optimales Spiel für deterministische Umgebungen und perfekte Info
- **Basisidee:** Wähle Zug mit höchstem Nützlichkeitswert in Relation zum besten Spiel des Gegners
- **Algorithmus:**
  1. Generiere Spielbaum vollständig
  2. Bestimme Nützlichkeit der Endzustände
  3. Propagiere Nützlichkeitswerte im Baum nach oben durch Anwendung der Operatoren MIN und MAX auf die Knoten in der jeweiligen Ebene
  4. An der Wurzel wähle den Zug mit maximalem Nützlichkeitswert
- Schritte 2 und 3 nehmen an, dass der Gegner perfekt spielt.



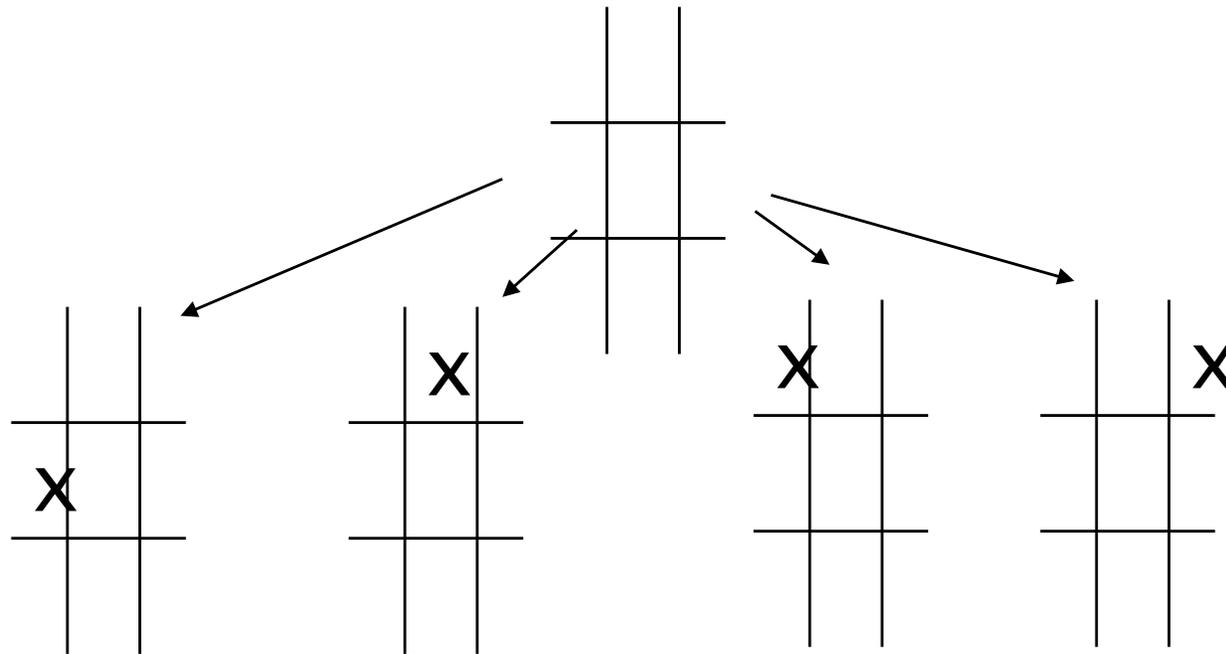
# Erzeuge Spielbaum

---

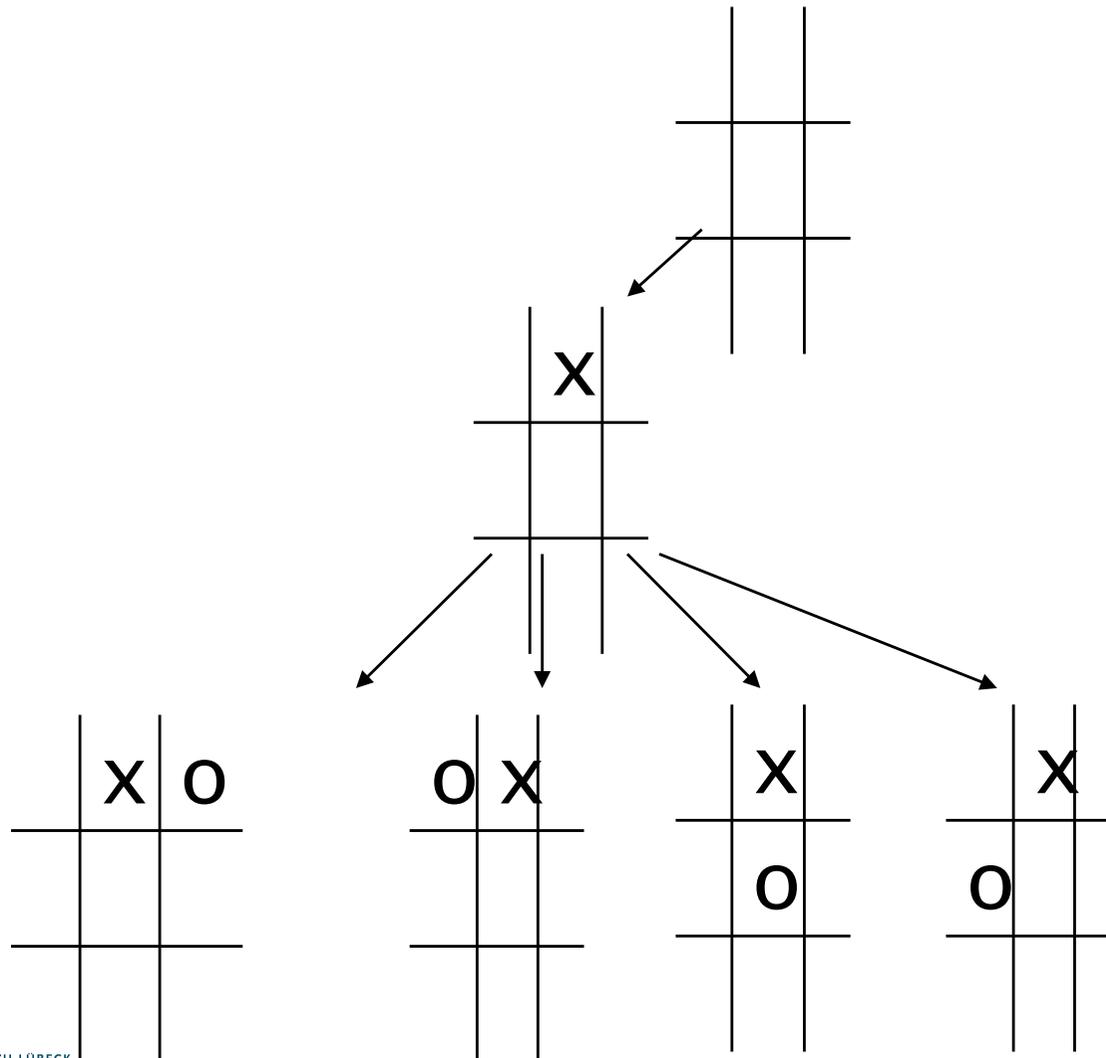


# Erzeuge Spielbaum

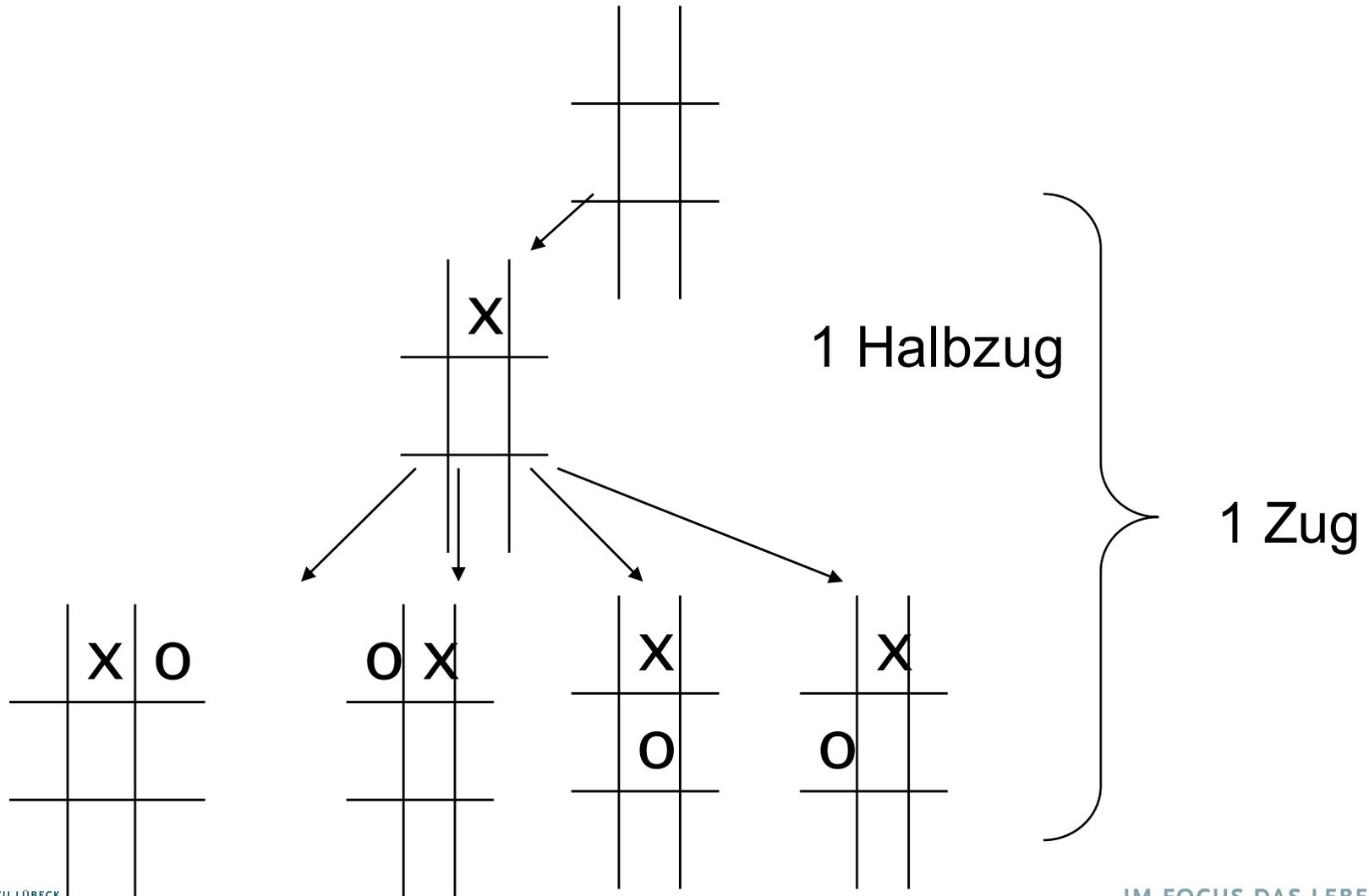
---



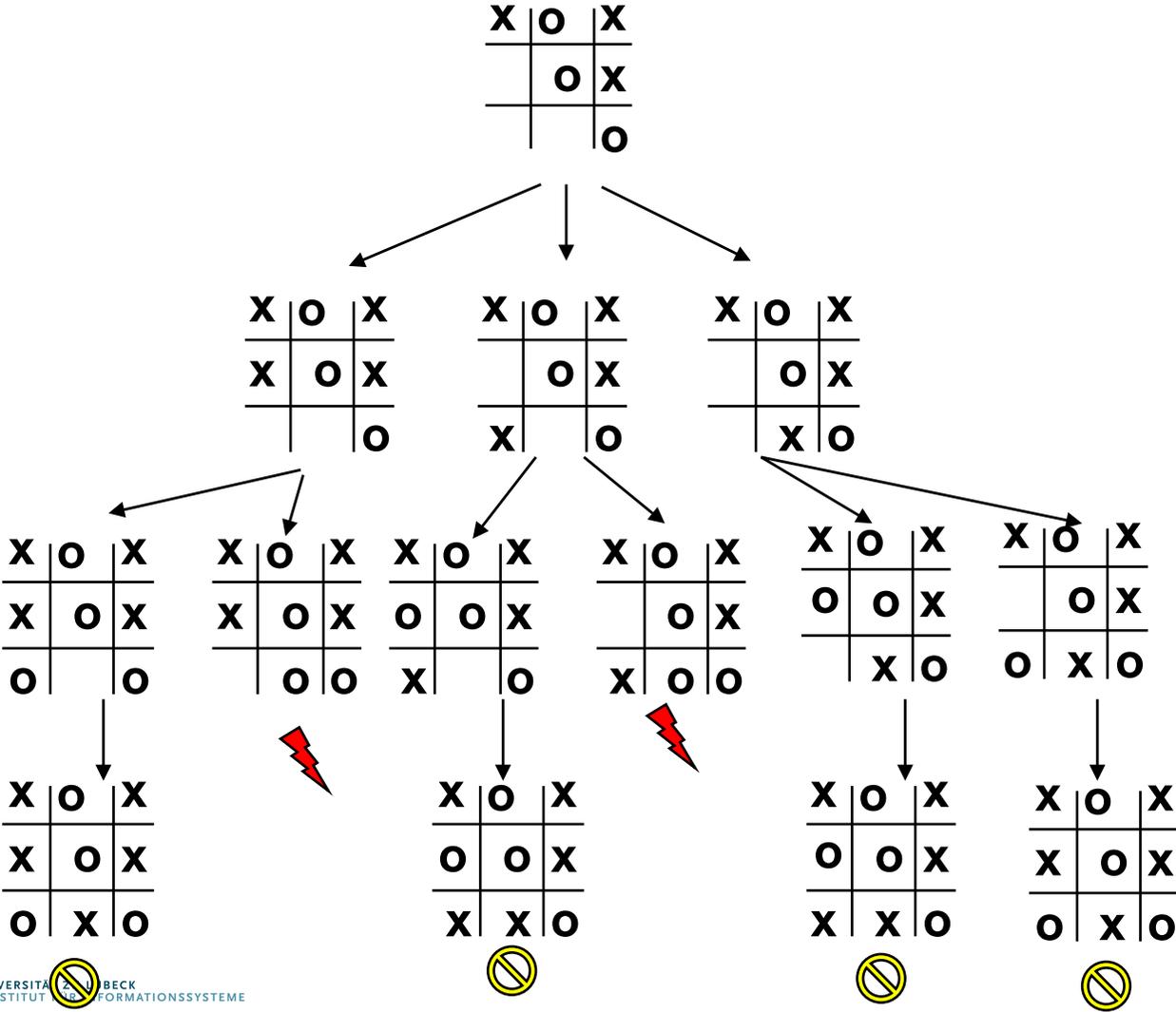
# Erzeuge Spielbaum



# Erzeuge Spielbaum

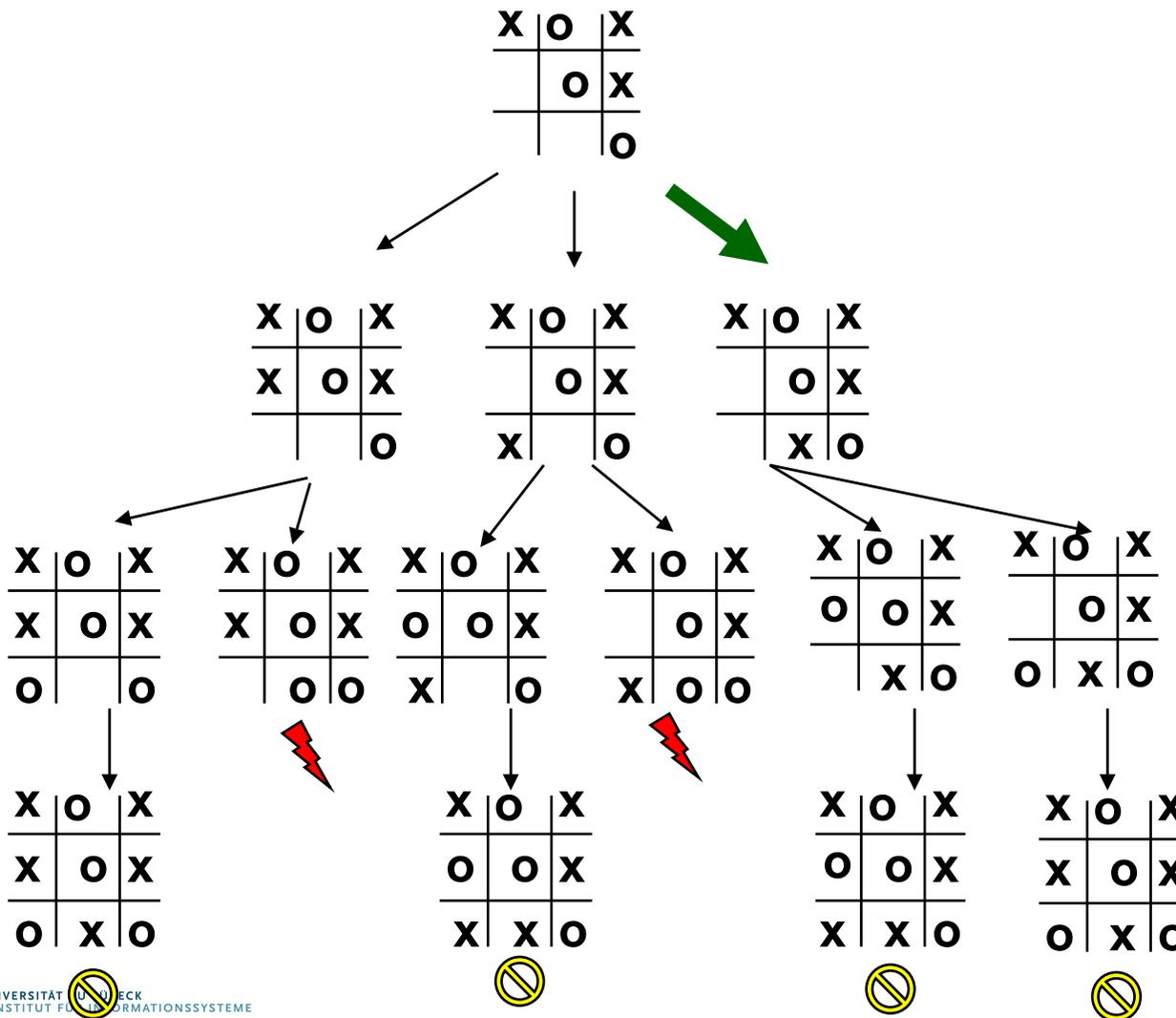


# Ein Teilbaum



-  gewonnen
-  verloren
-  unentschieden

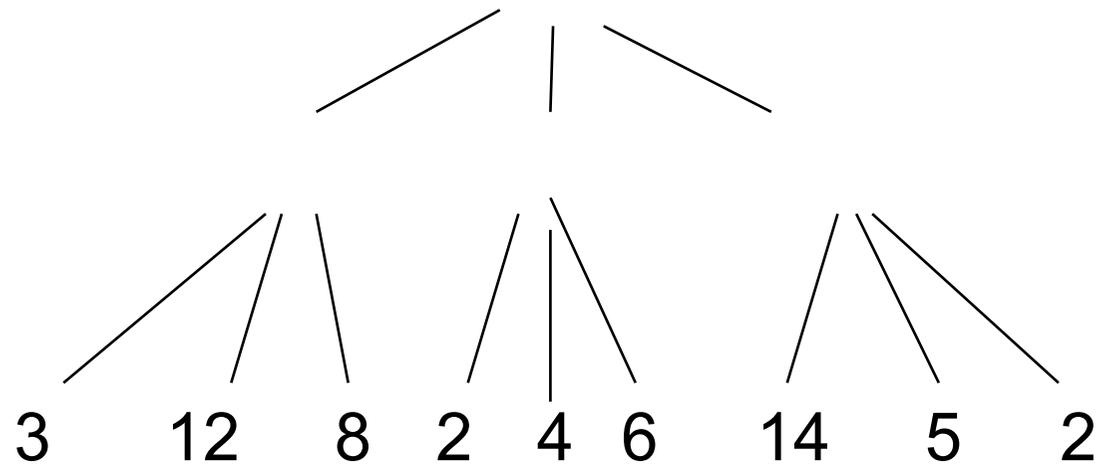
# Was ist ein guter Zug?



-  gewonnen
-  verloren
-  unentschieden

# Minimax

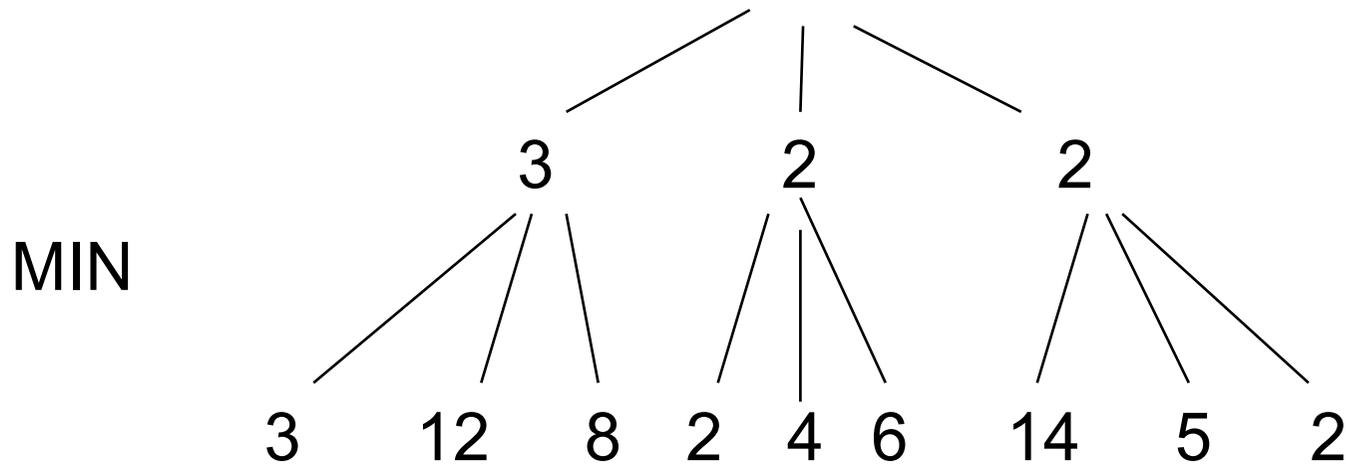
---



- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

# Minimax

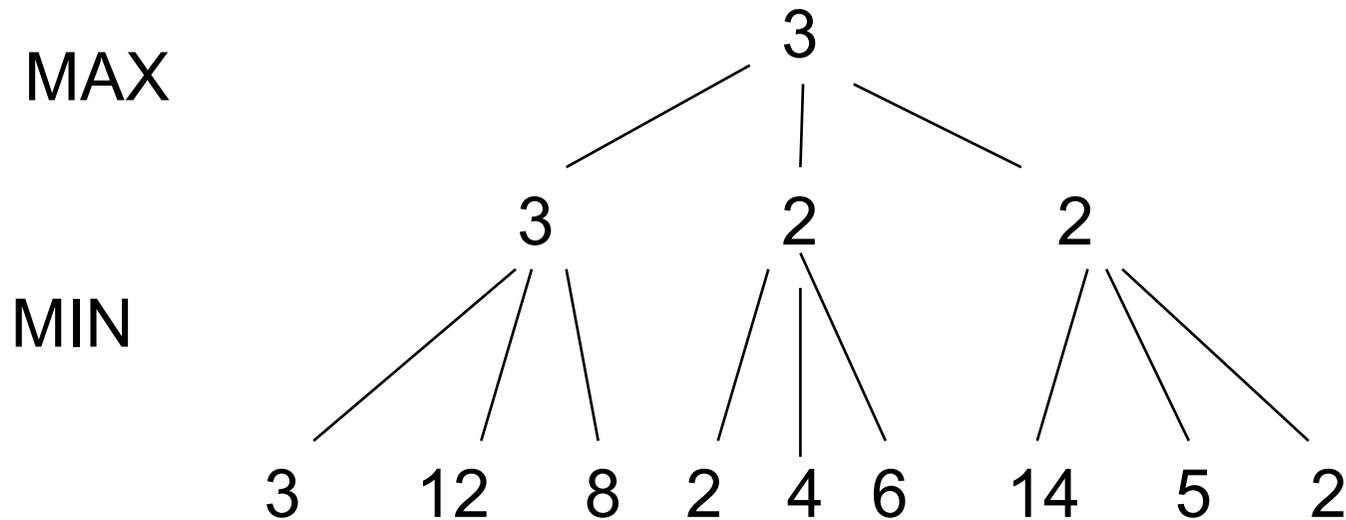
---



- Minimiere Gewinnmöglichkeiten für den Gegner
- Maximiere eigene Gewinnmöglichkeiten

# Minimax

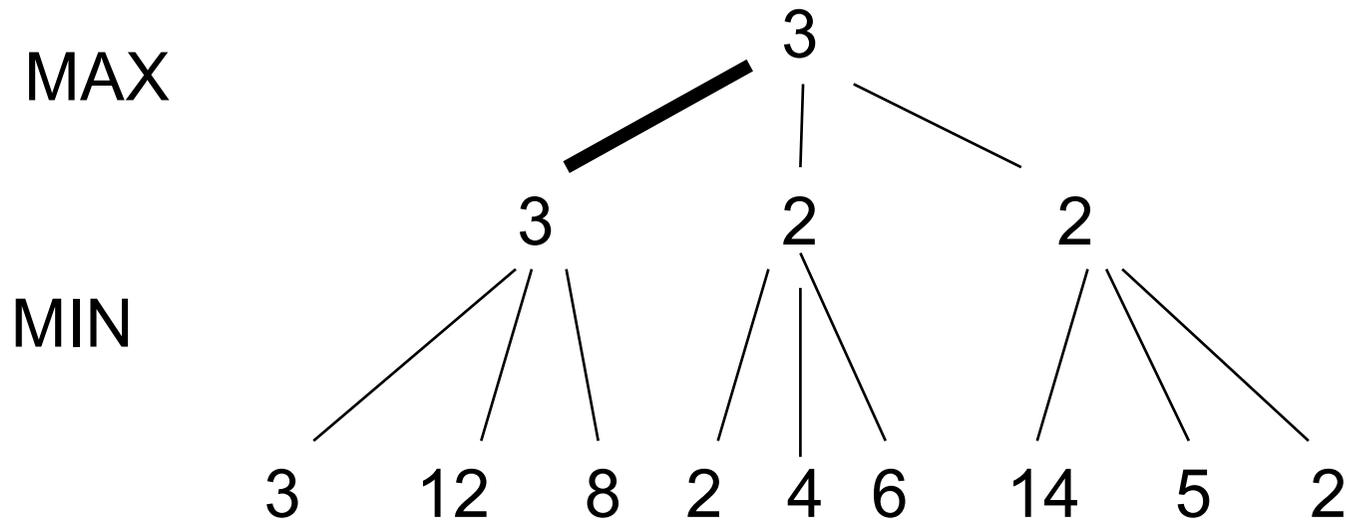
---



- 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**  $\arg \max_{a \in \text{ACTIONS}(\textit{state})} \text{MIN-VALUE}(\text{RESULT}(\textit{state}, a))$

---

**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**  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\textit{state}, a)))$   
**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**  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\textit{state}, a)))$   
**return** *v*

**Vollständig: ?**  
**Optimal: ?**

**Zeitkomplexität: ?**  
**Platzkomplexität: ?**



# Minimax: Rekursive Implementation

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

---

**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**  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\textit{state}, a)))$   
**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**  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\textit{state}, a)))$   
**return** *v*

**Vollständig:** Ja, für endl. Zust.raum    **Zeitkomplexität:**  $O(b^m)$   
**Optimal:** Ja                                **Platzkomplexität:**  $O(bm)$  (= DFS)



# Spiele 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<sup>50</sup> 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**

Suchbaumabschnidungen 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



# 1. $\alpha$ - $\beta$ -Pruning

---

- **Pruning:** Elimination eines Zweigs des Suchbaums, ohne vollständige Untersuchung jedes Knotens darunter (vgl. auch A\*)

- **$\alpha$ - $\beta$ -Pruning:** beschneide Teile des Suchbaums, die den Nützlichkeitswerte eines Max- oder Min-Knotens nicht verbessern können

(wobei nur bisher betrachtete Knotennützlichkeitswerte eine Rolle spielen)

- Geht das? Ja (s.u.).
  - Der Verzweigungsfaktor wird in etwa von  $b$  auf  $\sqrt{b}$  reduziert
  - Die Suchtiefe wird gegenüber Minimax ca. verdoppelt

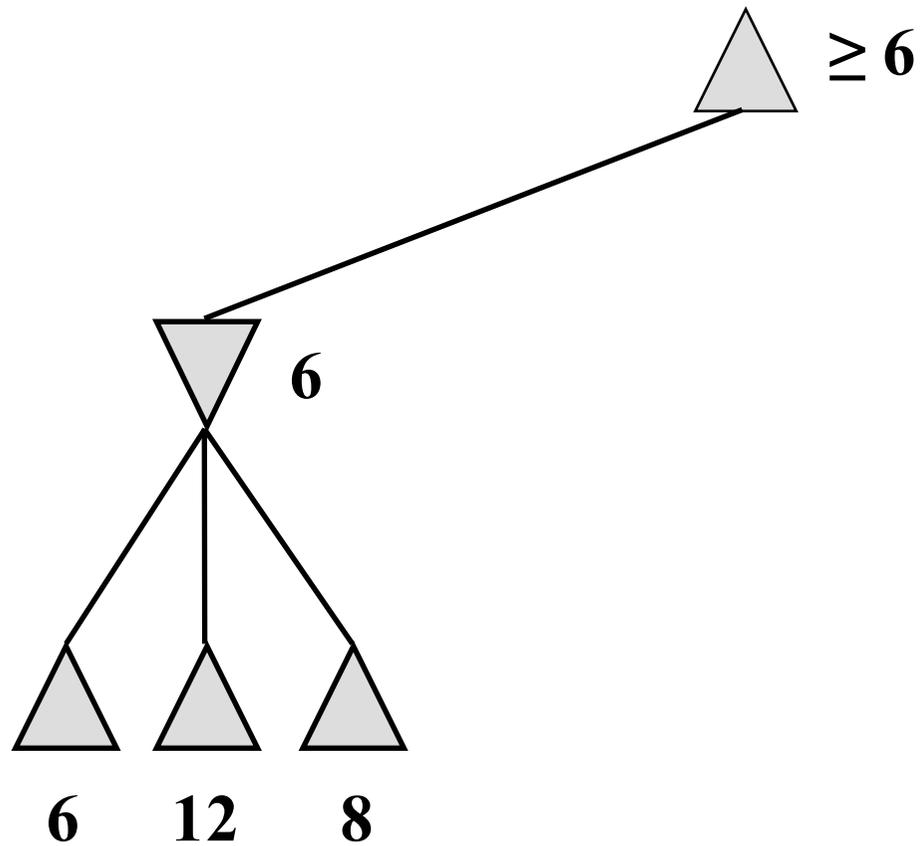


# $\alpha$ - $\beta$ -Pruning: Beispiel

---

**MAX**

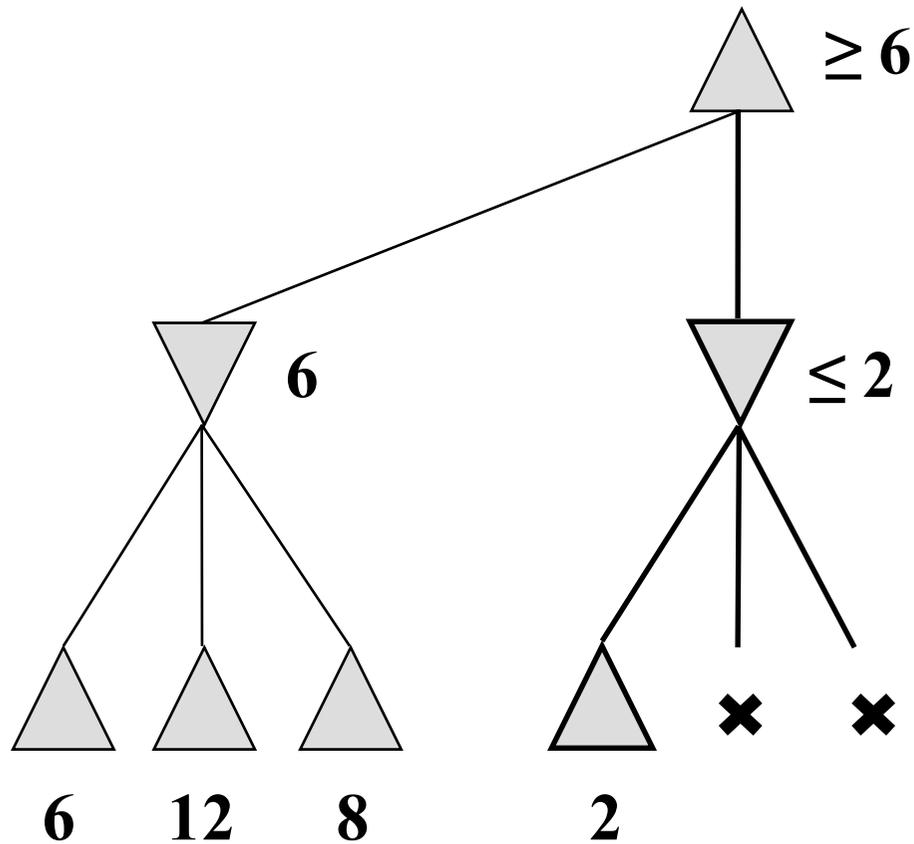
**MIN**



# $\alpha$ - $\beta$ -Pruning: Beispiel

MAX

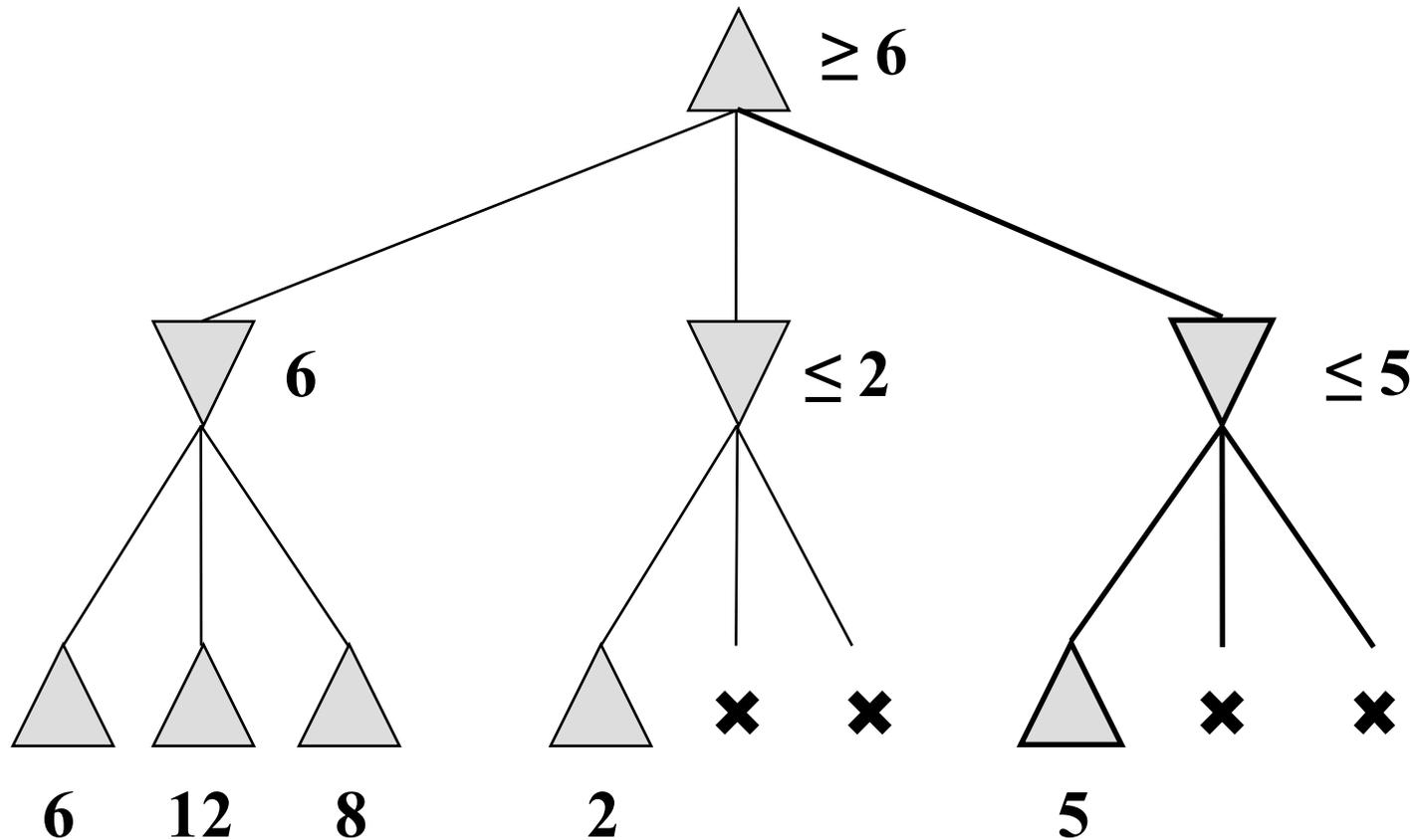
MIN



# $\alpha$ - $\beta$ -Pruning: Beispiel

**MAX**

**MIN**



MINIMAX(root)

$$= \max(\min(6, 12, 8), \min(2, a, b), \min(5, b, d))$$

$$= \max(6, z, y) \quad \text{wobei } z = \min(2, a, b) \leq 2 \quad \text{und } y = \min(5, b, d) \leq 5$$

$$= 6$$



# $\alpha$ - $\beta$ -Pruning: Algorithmus

---

- Weil Minimax Tiefensuche betreibt, betrachten wir die Knoten auf einem gegebenen Pfad im Baum
- Für jeden Pfad, speichern wir:
  - $\alpha$ : Bewertung der bislang besten Wahl für MAX
  - $\beta$ : Bewertung der bislang besten Wahl für MIN



# Suche mit Pfadbeschneidung

---

function ALPHA-BETA-SEARCH (state)

$\alpha := -\infty ; \beta := \infty$

if TERMINAL-TEST(state) then

return  $\perp$

else

return

$\operatorname{argmax}_{a \in \text{ACTIONS}(\text{state})} \{ v := \text{MIN-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta)$   
 $\alpha := \max(\alpha, v)$   
 $v \}$

# MAX-VALUE und MIN-VALUE mit $\alpha$ - $\beta$ -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  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\textit{state}, a), \alpha, \beta))$   
    if  $v \geq \beta$  then return v  
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
  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  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\textit{state}, a), \alpha, \beta))$   
    if  $v \leq \alpha$  then return v  
     $\beta \leftarrow \text{MIN}(\beta, v)$   
  return v
```

Darstellung entnommen aus:  
Russell, St., Norvig, P., Artificial Intelligence:  
A Modern Approach, 3<sup>rd</sup> Ed., 2010

IM FOCUS DAS LEBEN



# $\alpha$ - $\beta$ -Suchalgorithmus

In Min-Value:

```
 $v \leftarrow +\infty$   
for each  $a$  in  $ACTIONS(state)$  do  
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \leq \alpha$  then return  $v$   
   $\beta \leftarrow \text{MIN}(\beta, v)$   
return  $v$ 
```

MAX

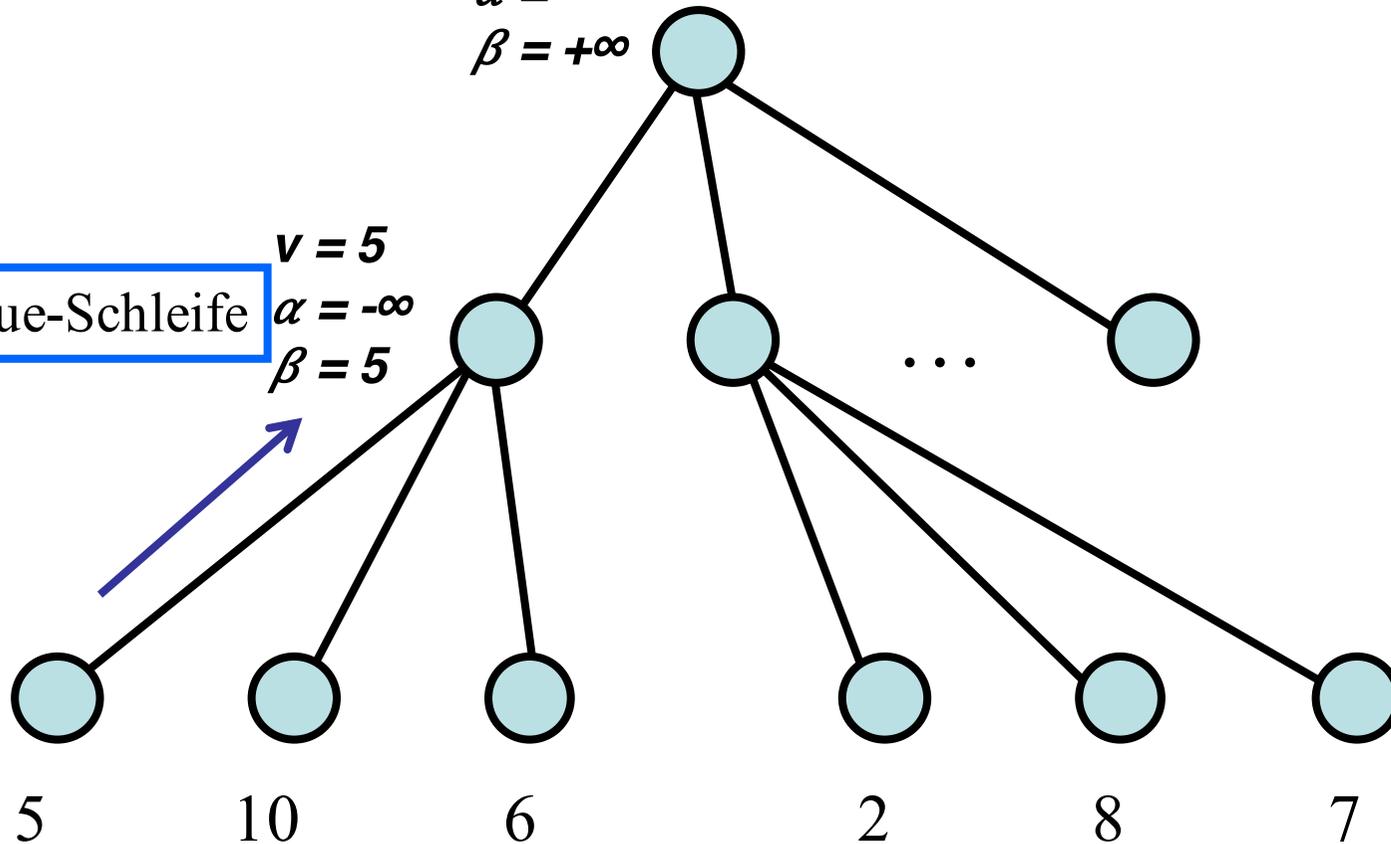
$v = -\infty$   
 $\alpha = -\infty$   
 $\beta = +\infty$

MIN

Min-Value-Schleife

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Min-Value:

```
 $v \leftarrow +\infty$   
for each  $a$  in  $ACTIONS(state)$  do  
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \leq \alpha$  then return  $v$   
   $\beta \leftarrow \text{MIN}(\beta, v)$   
return  $v$ 
```

MAX

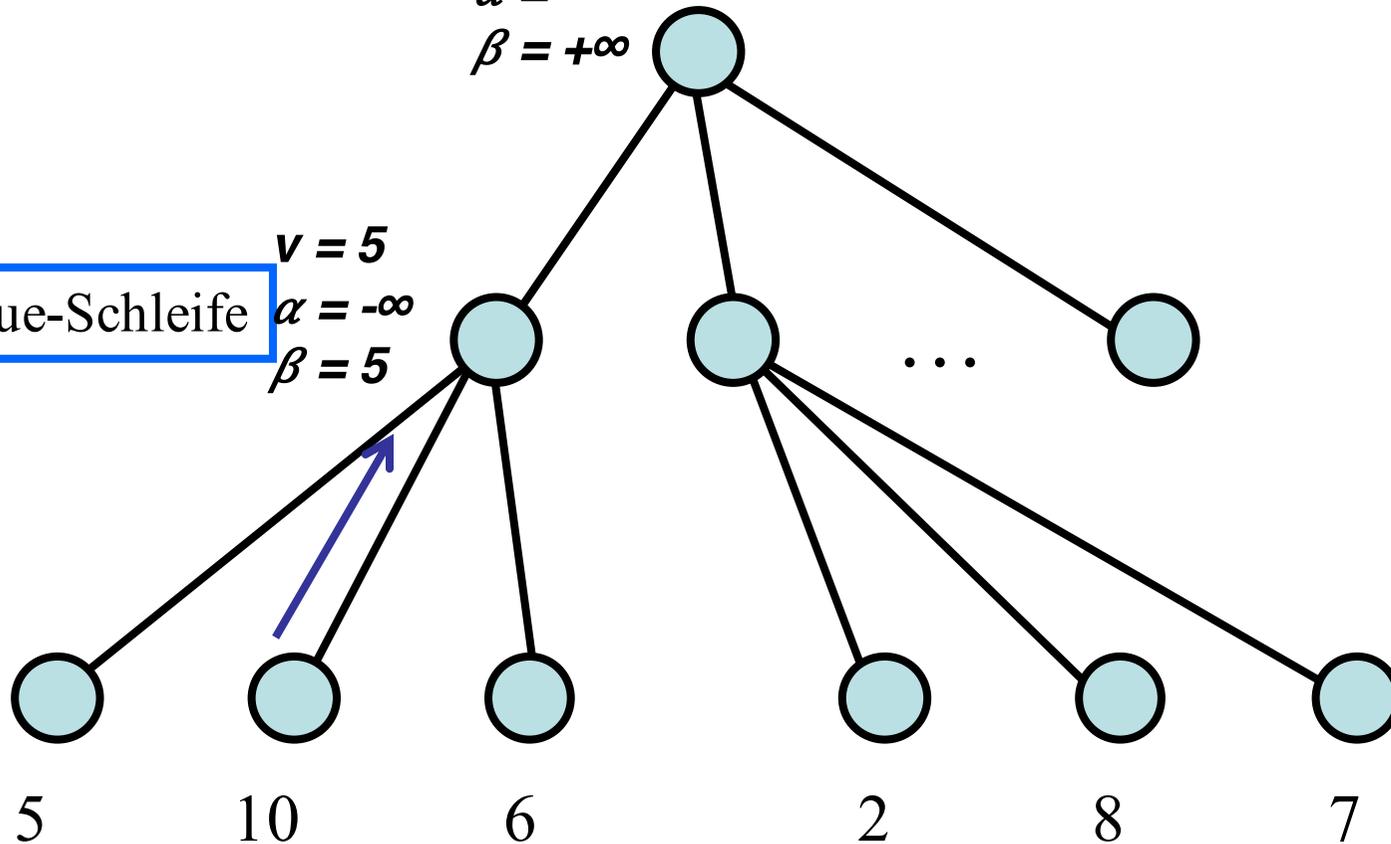
$v = -\infty$   
 $\alpha = -\infty$   
 $\beta = +\infty$

MIN

Min-Value-Schleife

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Min-Value:

```
 $v \leftarrow +\infty$   
for each  $a$  in ACTIONS( $state$ ) do  
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \leq \alpha$  then return  $v$   
   $\beta \leftarrow \text{MIN}(\beta, v)$   
return  $v$ 
```

MAX

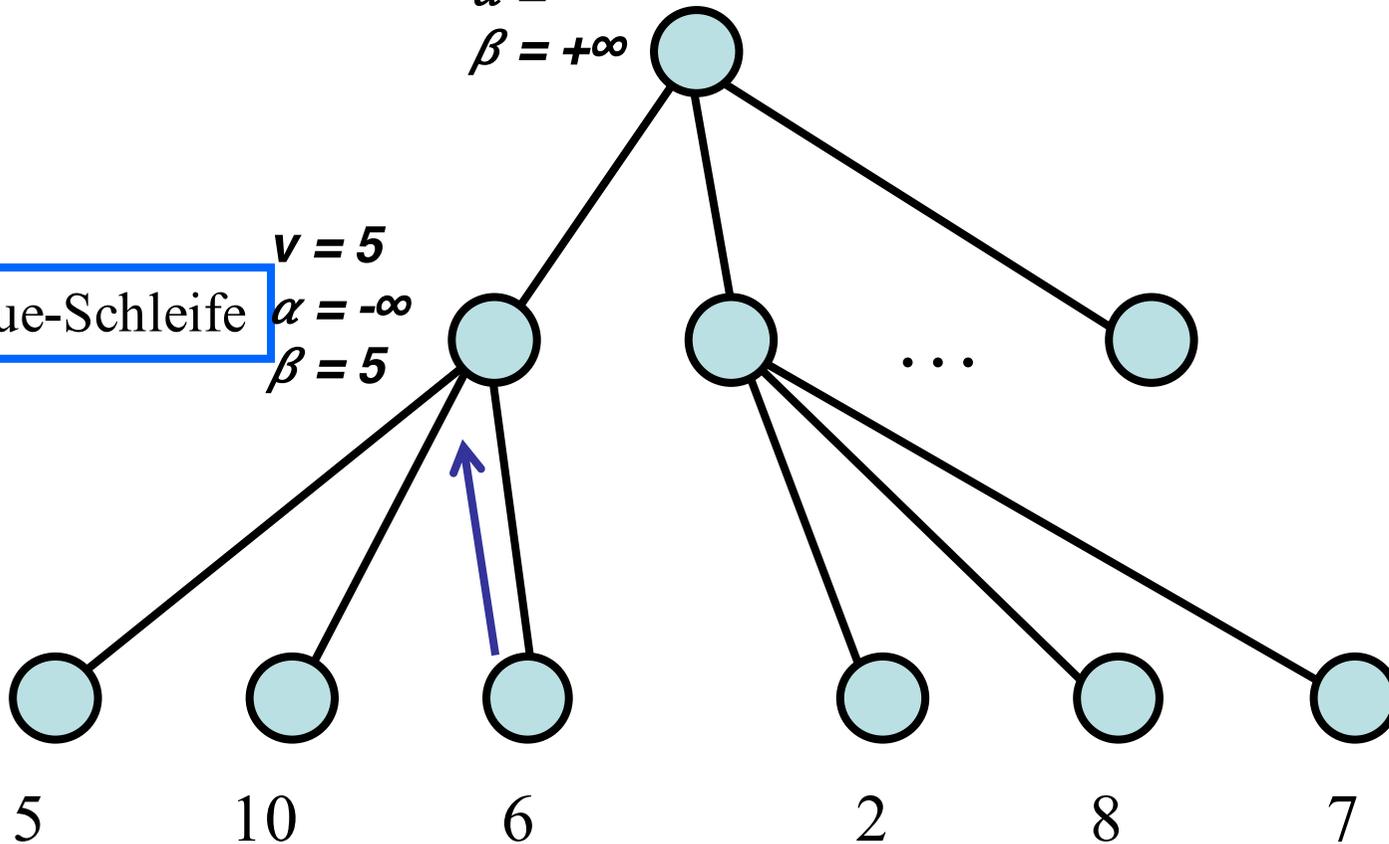
MIN

Min-Value-Schleife

$v = -\infty$   
 $\alpha = -\infty$   
 $\beta = +\infty$

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Max-Value:

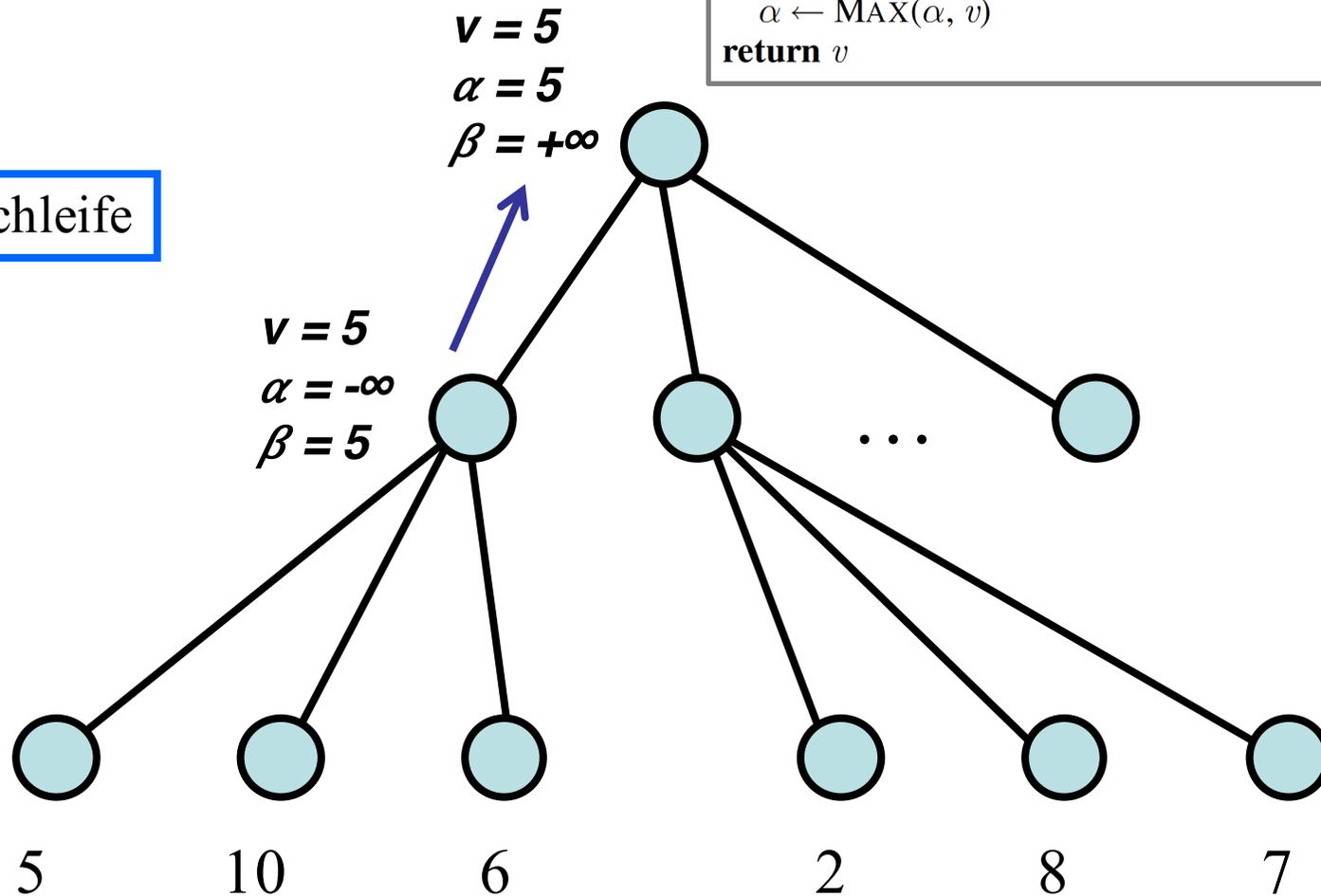
```
 $v \leftarrow -\infty$   
for each  $a$  in  $ACTIONS(state)$  do  
   $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \geq \beta$  then return  $v$   
   $\alpha \leftarrow \text{MAX}(\alpha, v)$   
return  $v$ 
```

MAX

Max-Value-Schleife

MIN

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Max-Value:

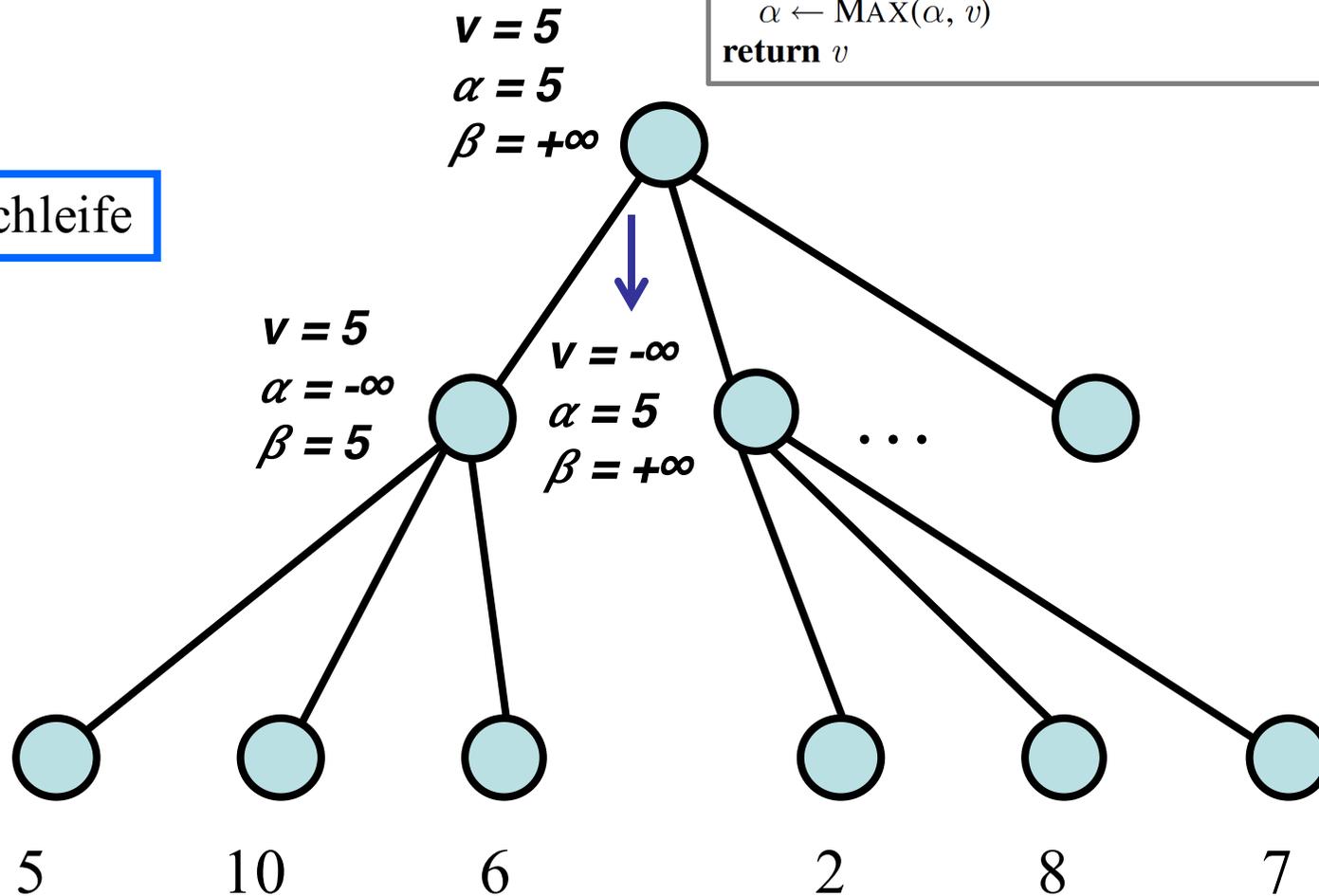
```
 $v \leftarrow -\infty$   
for each  $a$  in ACTIONS( $state$ ) do  
   $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \geq \beta$  then return  $v$   
   $\alpha \leftarrow \text{MAX}(\alpha, v)$   
return  $v$ 
```

MAX

Max-Value-Schleife

MIN

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Min-Value:

```
 $v \leftarrow +\infty$   
for each  $a$  in ACTIONS( $state$ ) do  
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \leq \alpha$  then return  $v$   
   $\beta \leftarrow \text{MIN}(\beta, v)$   
return  $v$ 
```

MAX

MIN

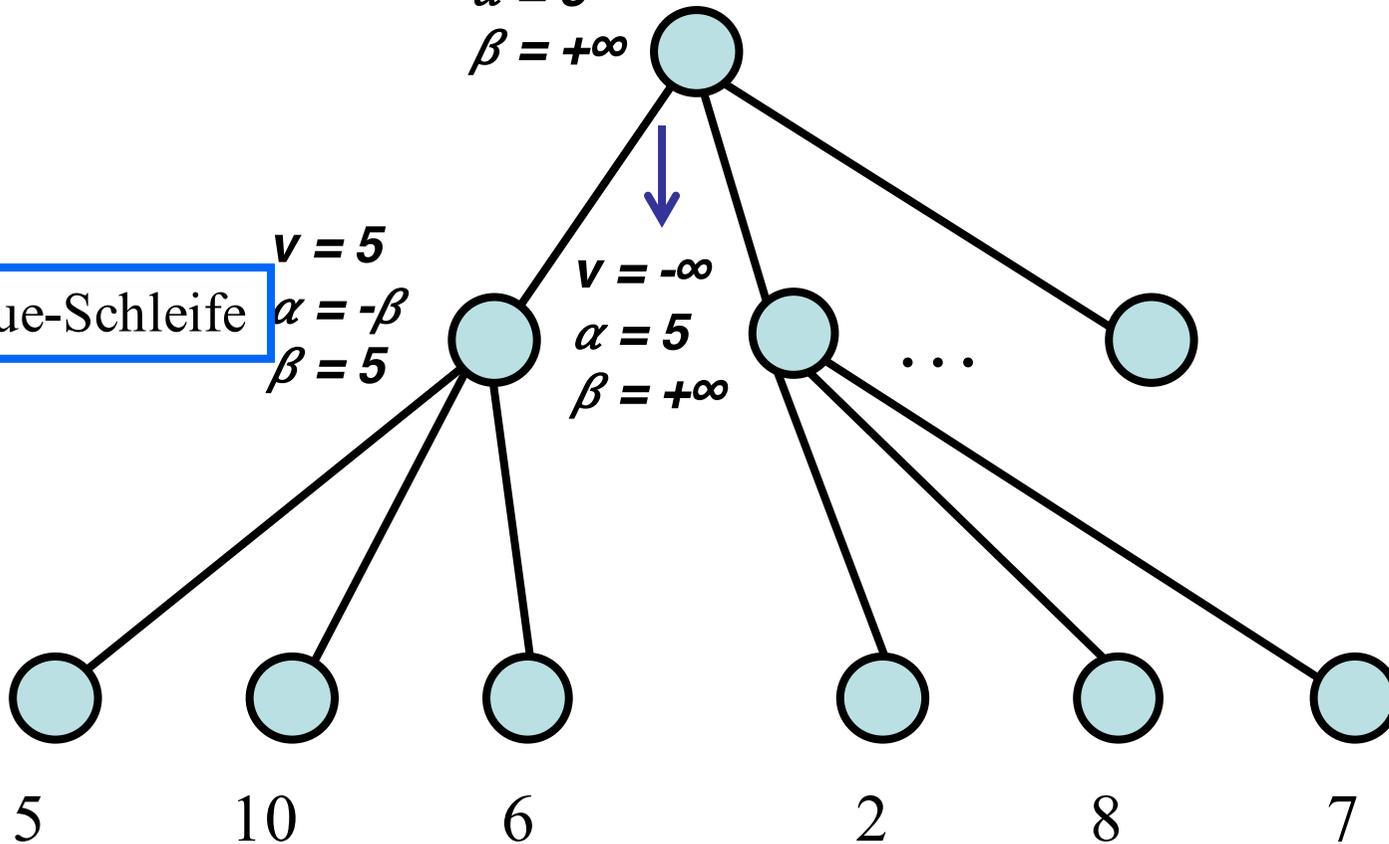
Min-Value-Schleife

$v = 5$   
 $\alpha = 5$   
 $\beta = +\infty$

$v = 5$   
 $\alpha = -\beta$   
 $\beta = 5$

$v = -\infty$   
 $\alpha = 5$   
 $\beta = +\infty$

MAX



# $\alpha$ - $\beta$ -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
    
```

MAX

MIN

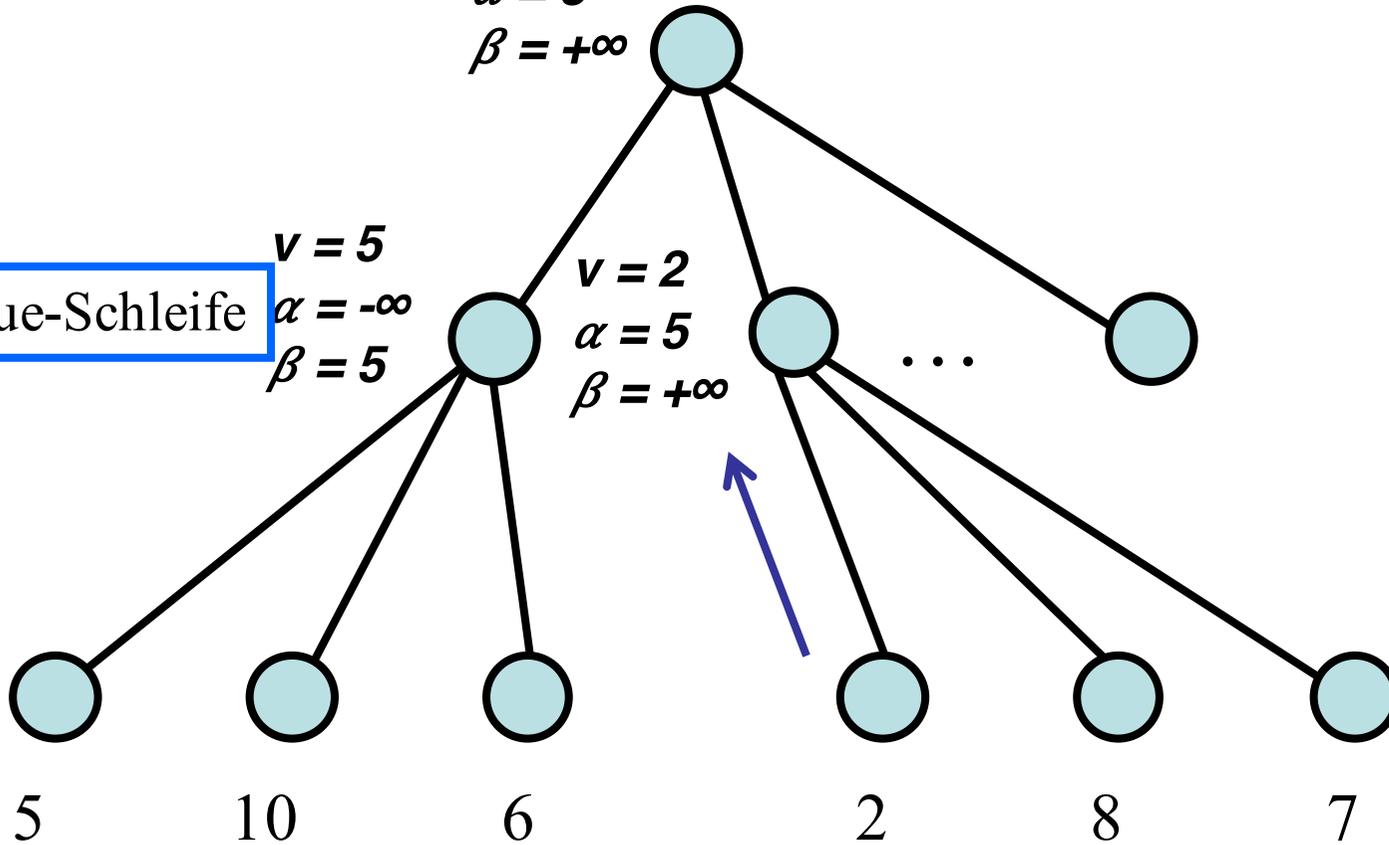
Min-Value-Schleife

$v = 5$   
 $\alpha = 5$   
 $\beta = +\infty$

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

$v = 2$   
 $\alpha = 5$   
 $\beta = +\infty$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Min-Value:

```
 $v \leftarrow +\infty$   
for each  $a$  in ACTIONS( $state$ ) do  
   $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \leq \alpha$  then return  $v$   
   $\beta \leftarrow \text{MIN}(\beta, v)$   
return  $v$ 
```

MAX

MIN

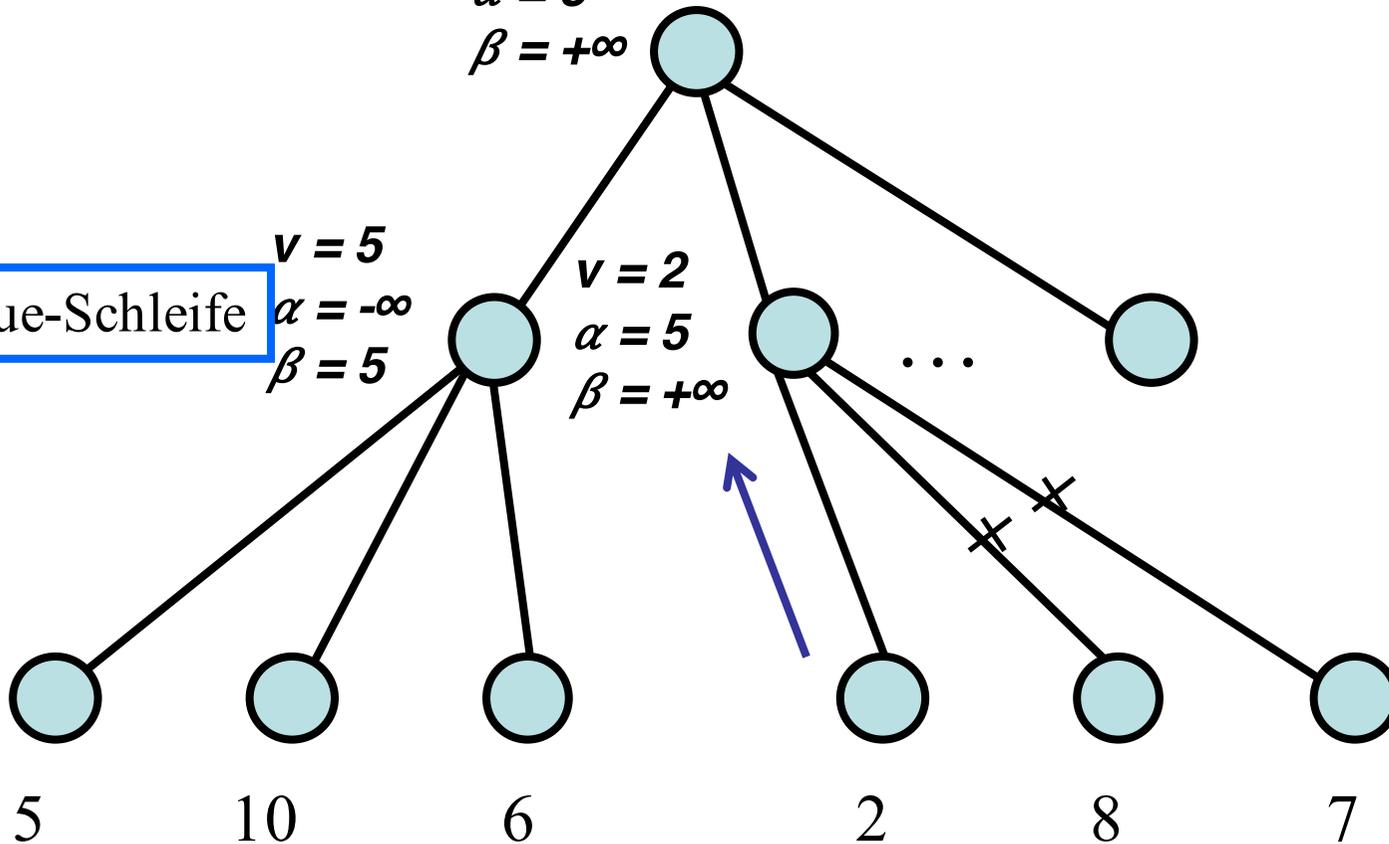
Min-Value-Schleife

$v = 5$   
 $\alpha = 5$   
 $\beta = +\infty$

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

$v = 2$   
 $\alpha = 5$   
 $\beta = +\infty$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus

In Max-Value:

```
 $v \leftarrow -\infty$   
for each  $a$  in ACTIONS( $state$ ) do  
   $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
  if  $v \geq \beta$  then return  $v$   
   $\alpha \leftarrow \text{MAX}(\alpha, v)$   
return  $v$ 
```

MAX Max-Value-Schleife

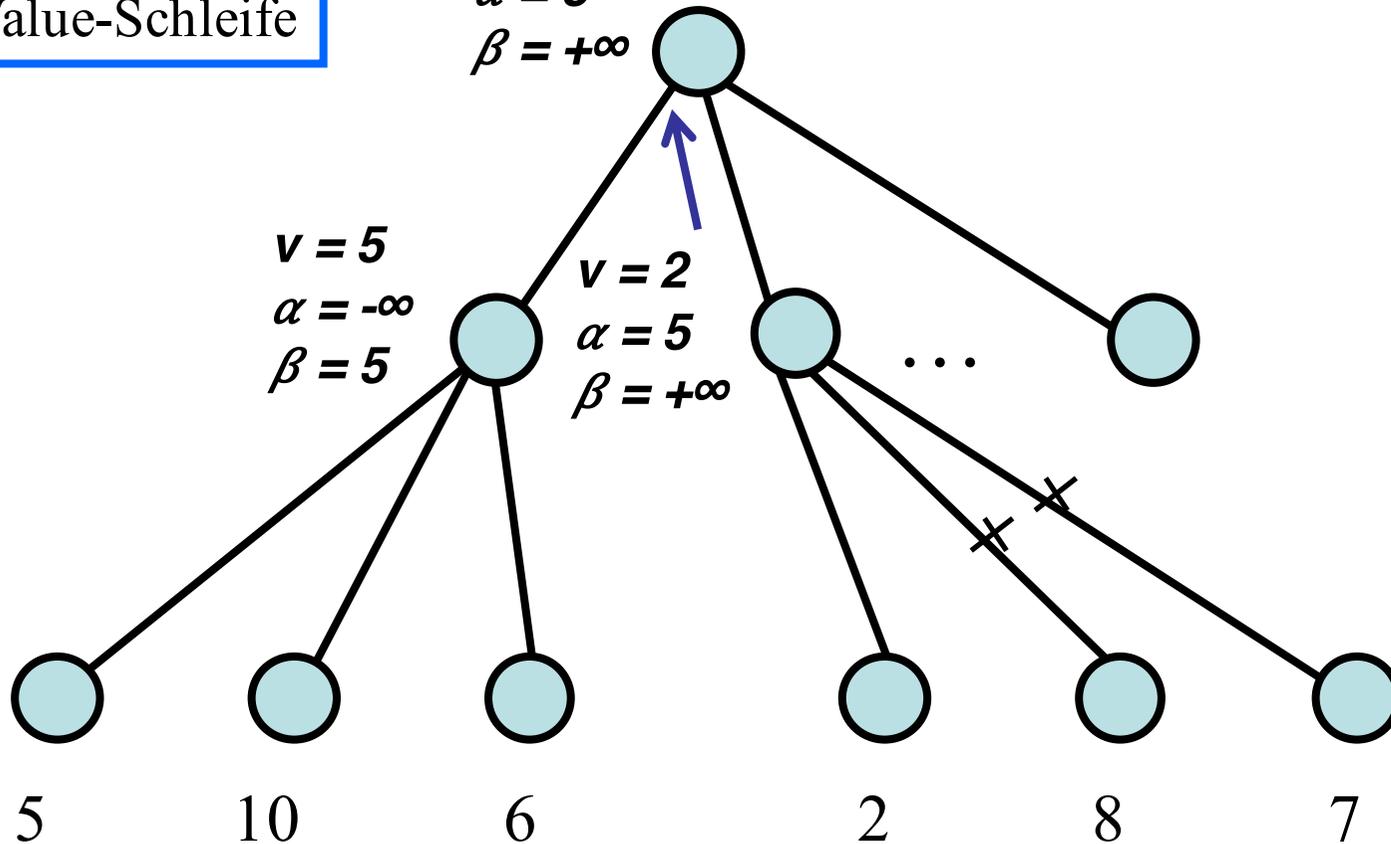
$v = 5$   
 $\alpha = 5$   
 $\beta = +\infty$

MIN

$v = 5$   
 $\alpha = -\infty$   
 $\beta = 5$

$v = 2$   
 $\alpha = 5$   
 $\beta = +\infty$

MAX



# $\alpha$ - $\beta$ -Suchalgorithmus: Eigenschaften

---

- Pruning hat keinen Einfluss auf das Endergebnis!!!
- Eine gute Reihenfolge der Betrachtung möglicher Züge erhöht die Effektivität vom Pruning
- Mit *perfekter Ordnung*, Zeitkomplexität =  $O(b^{m/2})$ 
  - Verdopplung der Suchtiefe
  - Gute Heuristik zur Ordnung nötig
  - Tiefe 8 erreichbar => guter Schachspieler

## 2. Zugbewertung ohne erschöpfende Suche

---

- Der Minimax-Algorithmus generiert den vollständigen Spielsuchraum, während der  $\alpha$ - $\beta$ -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

---

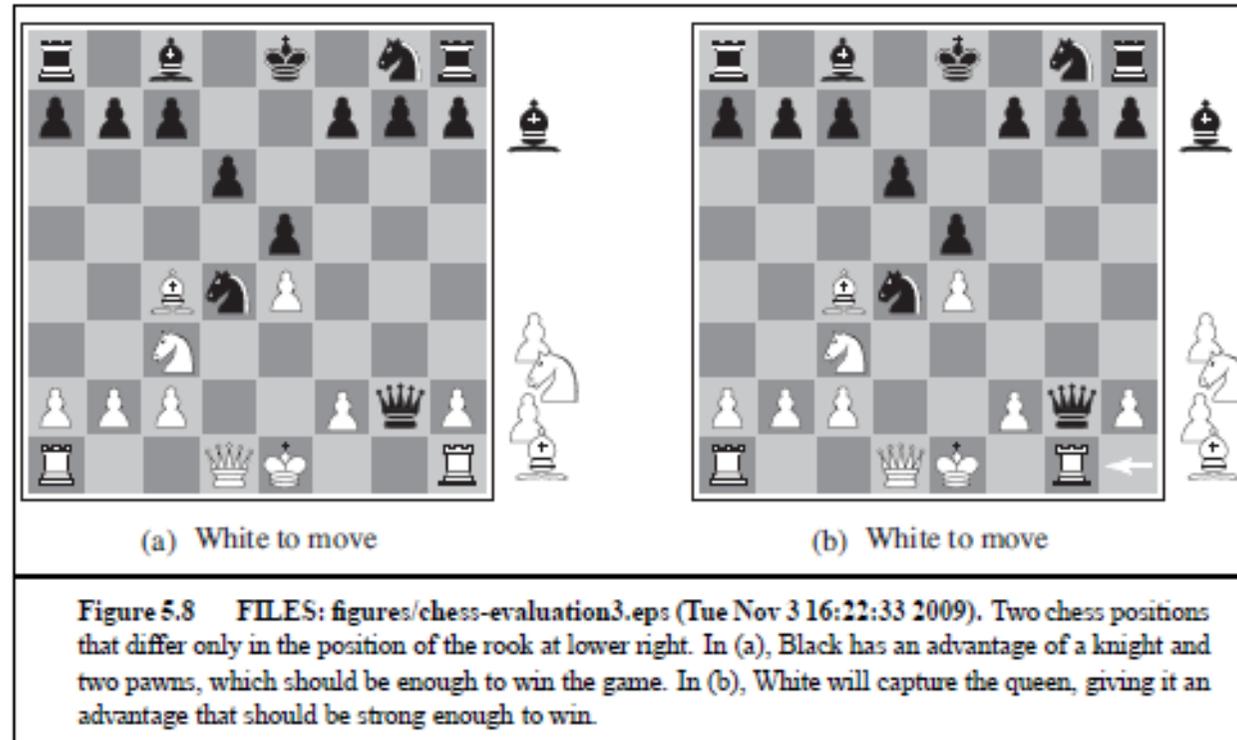
- Die Bewertungsfunktion sollt die Endzustände in der gleichen Weise ordnen wie die Nützlichkeitsfunktion ( $a < b < c \dots$ ).
- Die Berechnung der Funktion sollte schnell erfolgen
- Für Nicht-Endzustände sollte die Bewertungsfunktion die aktuelle Gewinnaussicht (Nützlichkeit) korrekt abschätzen.

# Bewertungsfunktion: Beispiel

---

- Merkmalsberechnungen (z.B. Anzahl der Bauern, Türme, ...)
- Es ergeben sich Kategorien mit Gewinn, Verlust, Unentschieden
- Für jede Merkmalskombination werden die Gewinnchancen abgeschätzt (gelernt)
- Beispiel für eine Merkmalskombination:  
72% Gewinn (+1), 20% Verlust (-1), 8% Unentschieden (0)
- Erwarteter Nutzen:  
 $(0,72 * +1) + (0,20 * -1) + (0,08 * 0) = 0,52$

# Bewertungsfunktion: Beispiel



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

$$f = w_1f_1 + w_2f_2 + \dots + w_nf_n$$

- 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**  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$   
    **if**  $v \geq \beta$  **then return**  $v$   
     $\alpha \leftarrow \text{MAX}(\alpha, v)$   
**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?

$$b^m = 10^6, b = 35 \rightarrow m = 4$$

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



# Zusammenfassung

---

- Wir haben z.B. erkannt, dass für einen Graphen  $G=(V, E)$  mit  $n = |V|$  und  $m = |E|$  gilt:
  - $T_{\text{Tiefensuche}} \in O(n+m)$
  - $T_{\text{Dijkstra}} \in O(n \log n + m)$  mit Fibonacci-Heaps
  - Folgerung: Optimierungsprobleme sind linear bzw. linear-logarithmisch lösbar
- Bei Spielsuchverfahren (oder auch  $A^*$ ) können wir den Graphen nicht sinnvoll vorher "erzeugen" und als Eingabe verwenden
  - Eingabe ist nur der Startknoten (Startzustand) plus Funktion zur Erzeugung von Knotennachfolgern (nächste Zustände)
  - Wir haben Verzweigungen im Suchraum: Probleme sind mit exponentiellem Aufwand lösbar
  - Folgerung: Grenzen des Einsatzes von Computern erkennbar