CSCM603130: Sistem Cerdas
Adversarial Search

Fariz Darari, Aruni Yasmin Azizah

Fakultas Ilmu Komputer
Universitas Indonesia

2019/2020 • Semester Ganjil

Outline

Outline

Competitive multiagent enviroment

- State space search: agent berinteraksi dengan environment.
- Terkadang environment berisi agent lain (multiagent environment): cooperative, competitive
- Competitive environment
  - agent-agent memiliki goal yang bertentangan (conflicting goals)
  - adversarial search problems $\rightarrow$ game
- Latar belakang: game theory (matematika, ekonomi)

Jenis-jenis game

	deterministic	stochastic
perfect information	chess, checkers, go, othello	monopoly, backgammon
imperfect information	battleship	bridge, poker, nuclear war

Dalam sejarah AI, *game* yang biasanya jadi bahan riset:

- Deterministic
- Perfect information (fully observable)
- 2 pemain, *turn-taking*
- Zero-sum game (nilai utility dari kedua pemain pada akhir game adalah sama, hanya berbeda polaritas)

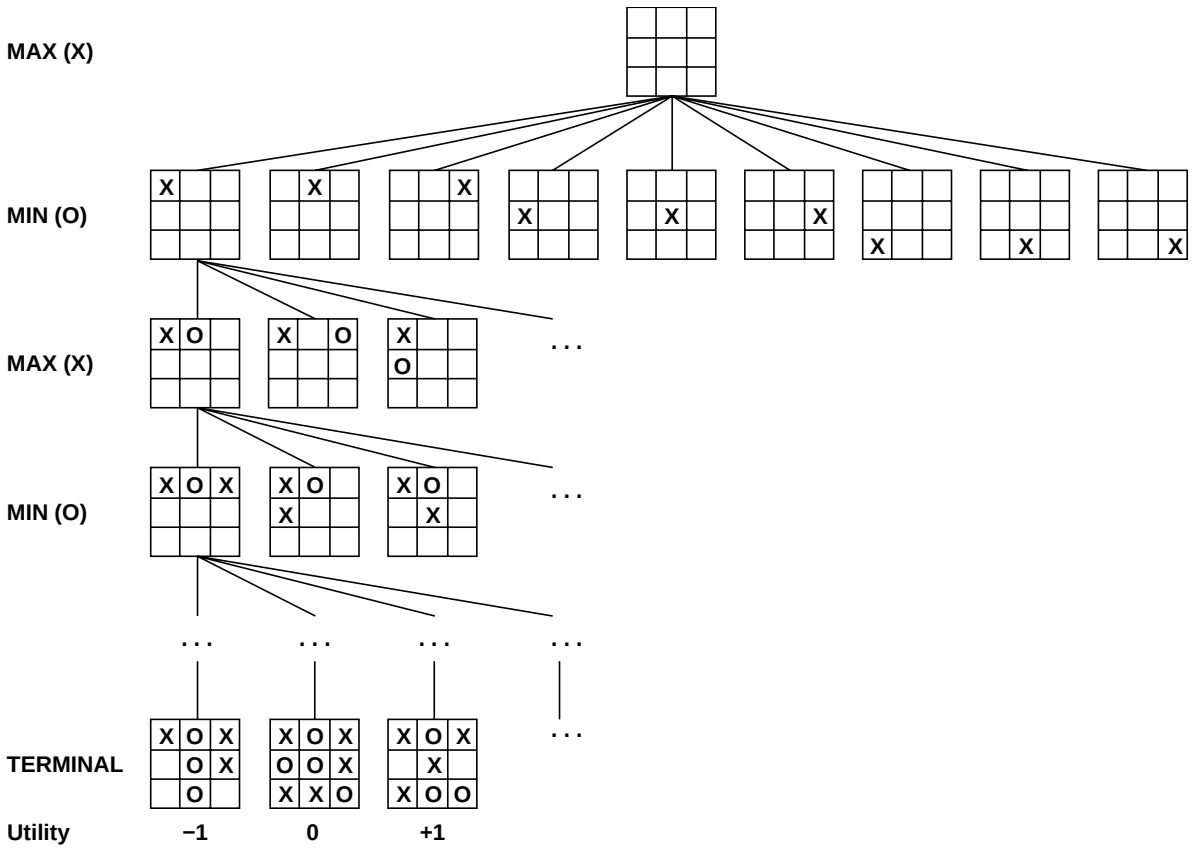
Outline

Game sebagai search

- State: konfigurasi permainan
  - Initial state (S_0): konfigurasi di awal permainan
- PLAYER(s): pemain (agent) yang melakukan tindakan (action) pada state s
- ACTIONS(s): himpunan tindakan yang diijinkan pada state s
- RESULT(s, a): model transisi yang mendefinisikan state hasil dari tindakan a pada state s
- TERMINAL-TEST(s): menentukan apakah permainan sudah selesai pada state s (mencapai terminal state)
- UTILITY(s, p): utility function/payoff function yg mendefinisikan nilai numerik terhadap terminal state s untuk pemain p. Mis: menang (+1), seri (0), kalah (-1).

Initial state, fungsi ACTIONS, fungsi PLAYER, dan fungsi RESULT mendefinisikan sebuah game tree.

Contoh game tree: tic-tac-toe



Game tree tic-tac-toe relatif "kecil" (kurang dari $9! = 362K$ terminal nodes).
Pada catur bisa mencapai 10^{40} terminal nodes.

Outline

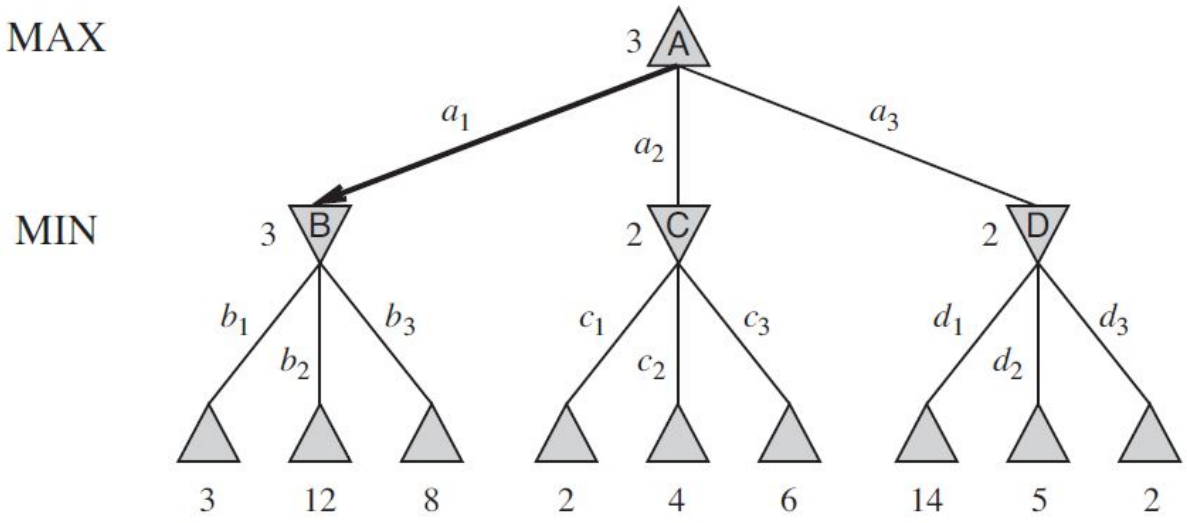
Solusi optimal dalam sebuah game

- Andaikan sebuah permainan antara 2 pemain: MAX (agent) dan MIN
- Kondisi awal dimulai dari giliran MAX, kemudian MIN, dst. sampai permainan selesai
- 1 langkah = 2 ply/layer (MAX jalan, MIN jalan)
- Search pada game tree: cari path sehingga mencapai terminal state di mana MAX menang.
  - Tapi langkah MIN mempengaruhi search yang dilakukan MAX!
  - nilai utility yang tinggi pada terminal state adalah baik untuk MAX dan buruk untuk MIN
- Solusi berupa *contingent strategy* untuk setiap kemungkinan langkah MIN.

Solusi optimal game 2 pemain: Fungsi MINIMAX

MINIMAX(s)=

UTILITY(s)	jika TERMINAL-TEST(s)
$max_{a \in Actions(s)}$ MINIMAX(RESULT(s, a))	jika PLAYER(s) = MAX
$min_{a \in Actions(s)}$ MINIMAX(RESULT(s, a))	jika PLAYER(s) = MIN



Tindakan a_1 adalah pilihan optimal bagi MAX: minimax decision pada root.

- Ini strategi optimal, atau *optimal play*: memberikan hasil terbaik melawan musuh yang diasumsikan optimal.

Algoritma Minimax

Algoritma Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*

return $\mathit{argmax}_{a \in \mathit{Actions}(s)}$ 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*

Algoritma Minimax

- Definisi algoritma ini rekursif, dengan *base case* pada *terminal state*
- Untuk menghitung MinimaxValue pada *initial state*, harus *depth-first search* pada game tree!
- Untuk tree dengan kedalaman m dan banyak aksi adalah b:
  - Complete? Ya, kalau game tree-nya *finite*
  - Optimal? Ya, asumsi lawan musuh optimal juga. (Kalau tidak? "Lebih optimal"!)
  - Time complexity? $O(b^m)$
  - Space complexity? $O(bm)$ (atau $O(m)$ dgn. backtracking)

Algoritma Minimax

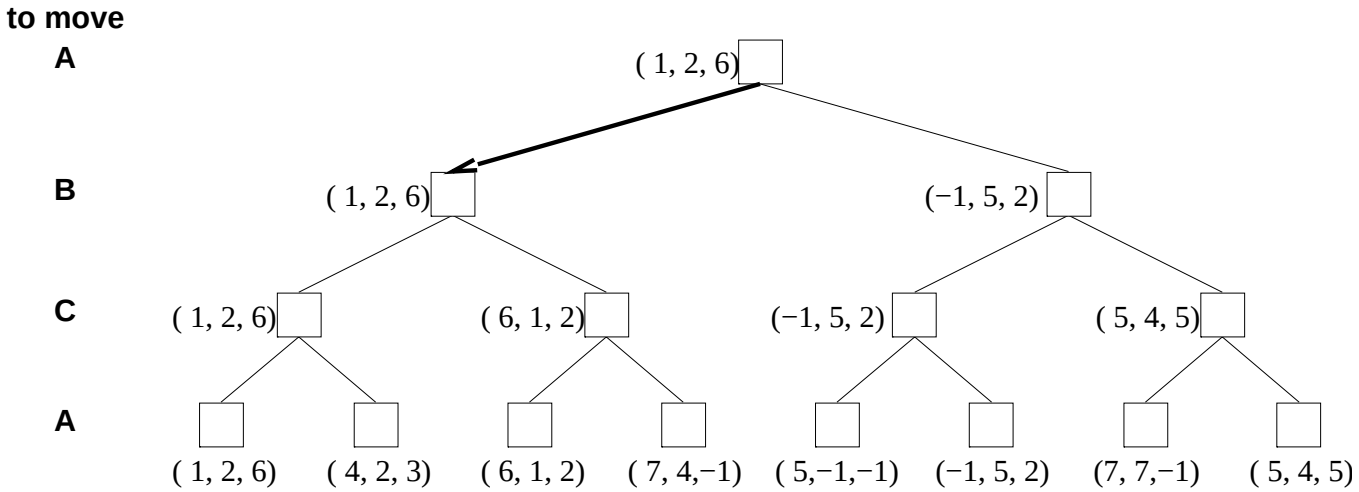
- Definisi algoritma ini rekursif, dengan *base case* pada *terminal state*
- Untuk menghitung MinimaxValue pada *initial state*, harus *depth-first search* pada game tree!
- Untuk tree dengan kedalaman m dan banyak aksi adalah b:
  - Complete? Ya, kalau game tree-nya *finite*
  - Optimal? Ya, asumsi lawan musuh optimal juga. (Kalau tidak? "Lebih optimal"!)
  - Time complexity? $O(b^m)$
  - Space complexity? $O(bm)$ (atau $O(m)$ dgn. backtracking)

Teori sih OK...

Untuk catur: $b \approx 35$, $m \approx 100 \rightarrow$ pencarian strategi optimal berdasarkan Minimax tidak feasible!

Game dengan 3 (atau lebih) pemain

- Intinya sama dengan minimax: setiap pemain berlaku optimal
- Nilai setiap node berupa *vektor* dengan n nilai
  Mis: untuk 3 pemain $A, B, C \rightarrow < v_A, v_B, v_C >$
- Pada terminal state: nilai *utility* untuk setiap pemain



Ternyata dengan mengikuti strategi optimal ini bisa muncul aliansi, mis: A & B sama-sama lemah, lawan C.

- Aliansi tidak perlu berlanjut setelah C dikalahkan.

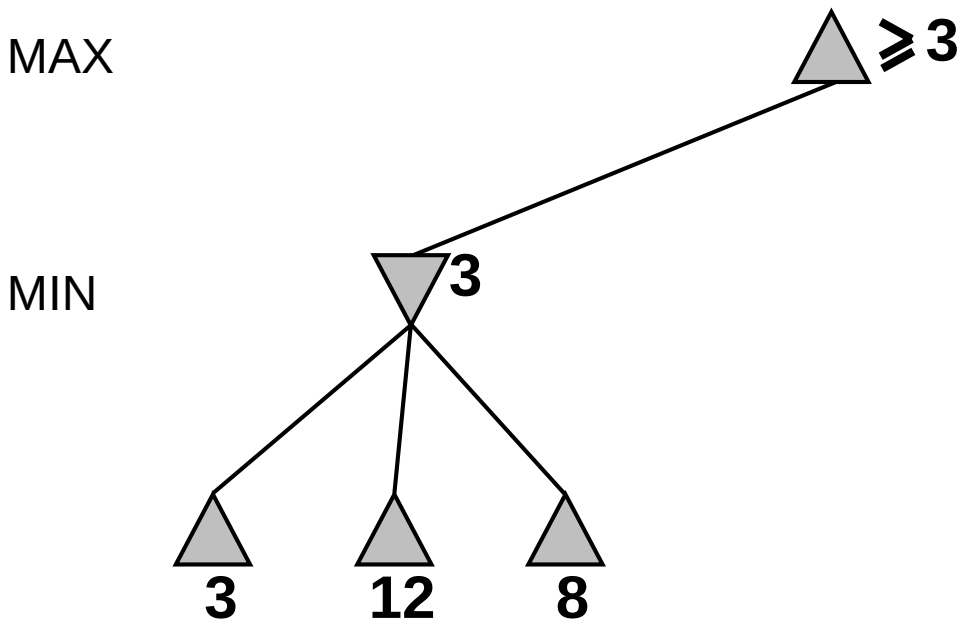
Outline

Outline

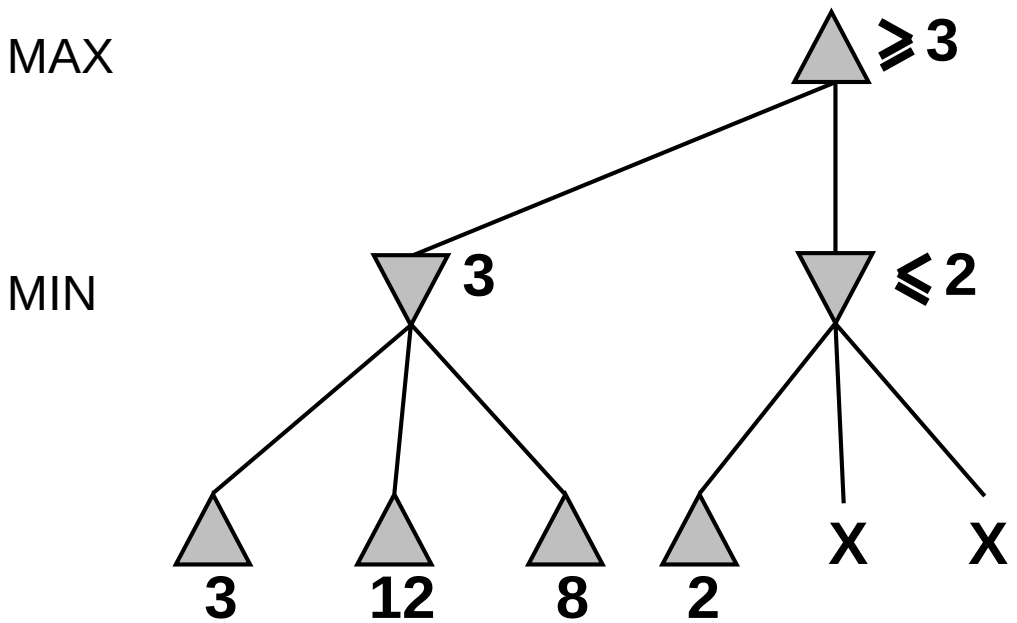
Pruning (memangkas) game tree

- Kinerja MINIMAX masih bisa diperbaiki dengan pruning (memangkas) game tree.
- Prinsipnya: node (subtree) yang tidak mungkin mempengaruhi hasil akhir tidak perlu ditelusuri.
- Pruning demikian dilakukan oleh algoritma alpha-beta pruning

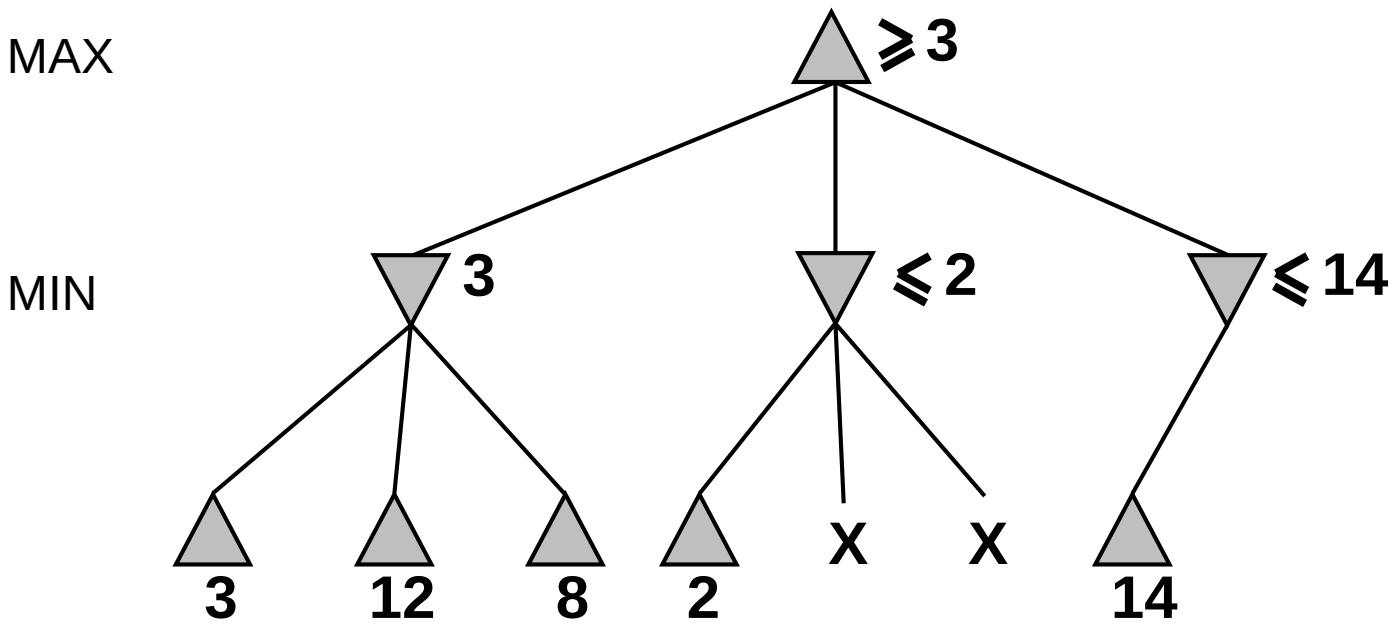
Contoh $\alpha - \beta$ pruning



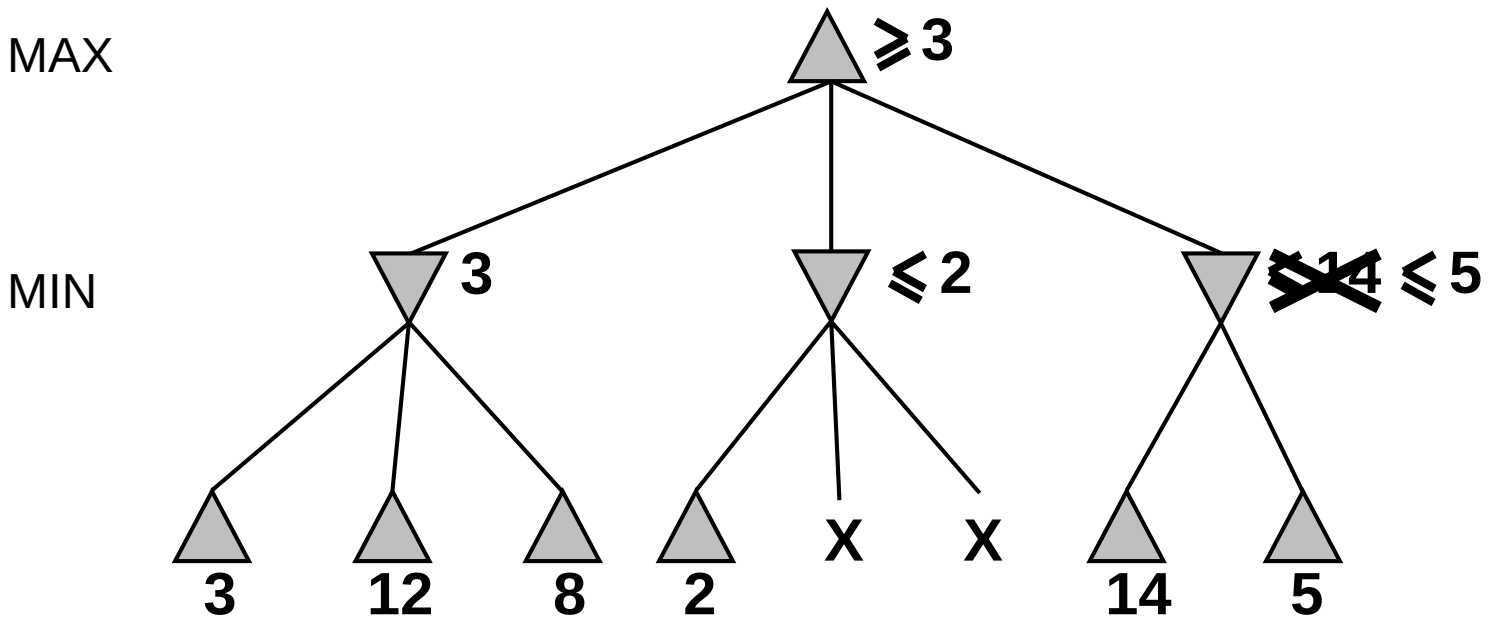
Contoh $\alpha - \beta$ pruning



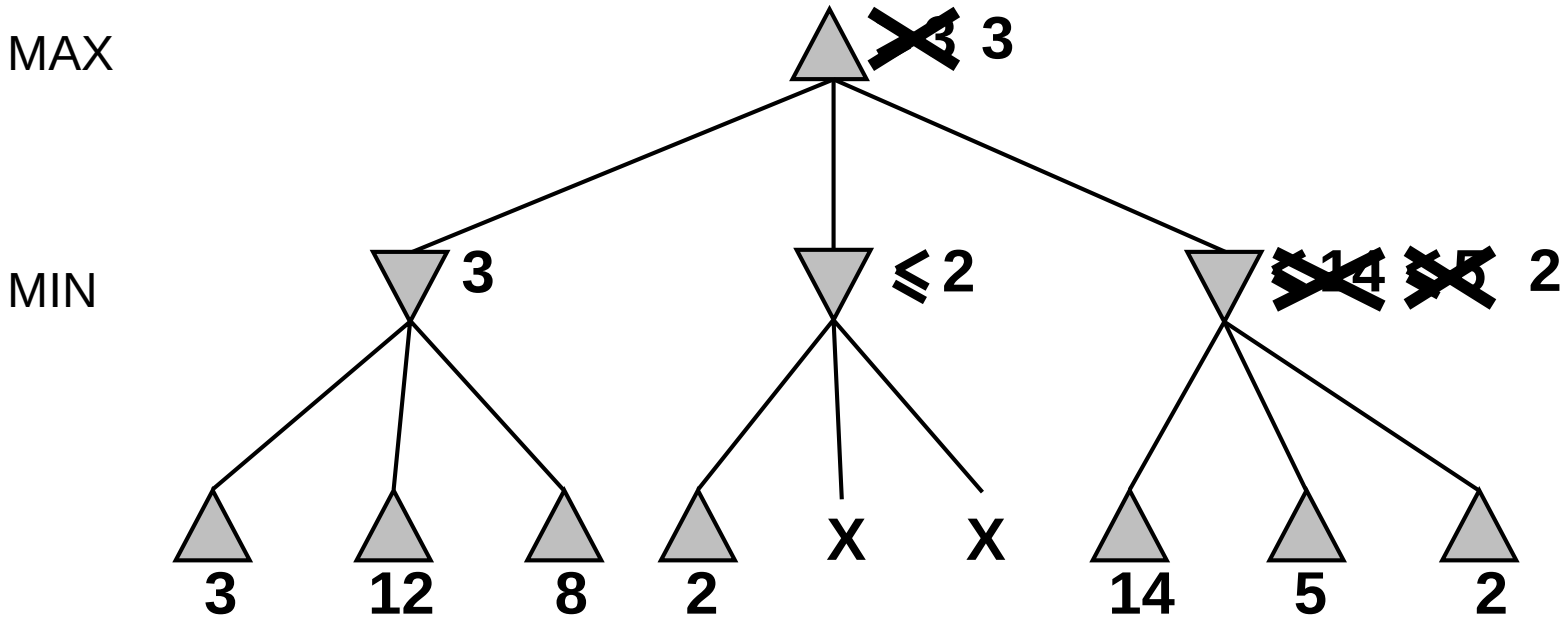
Contoh $\alpha - \beta$ pruning



Contoh $\alpha - \beta$ pruning

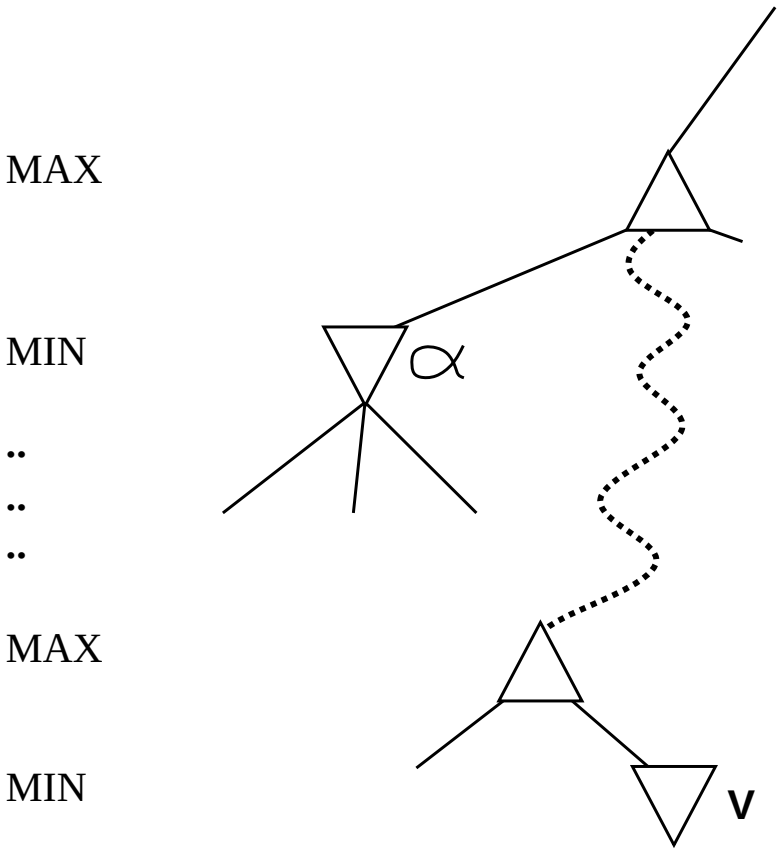


Contoh $\alpha - \beta$ pruning



$$\begin{aligned} \textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\ &= \max(3, \min(2,x,y), 2) \\ &= 3 \end{aligned}$$

Prinsip dasar $\alpha - \beta$ pruning



Pruning dengan α

α adalah nilai terbesar (terbaik untuk MAX) sementara yang sudah diketahui. Jika nilai $V < \alpha$, MAX tidak pernah akan memilihnya $\rightarrow V$ bisa dipangkas.

Pruning dengan β

β adalah nilai terkecil (terbaik untuk MIN) sementara yang sudah diketahui. Jika nilai $V > \beta$, MIN tidak pernah akan memilihnya $\rightarrow V$ bisa dipangkas.

Algoritma Alpha-Beta Pruning

Algoritma Alpha-Beta Pruning

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

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

Sifat alpha-beta pruning

- Alpha-beta pruning tidak mempengaruhi hasil akhir algoritma minimax
- Urutan penelusuran nilai (atau move ordering) mempengaruhi kinerja $\rightarrow$ coba pilih nilai yang "terbaik" dulu
- Dengan urutan yang ideal $\rightarrow O(b^{m/2})$
  - Effective branching factor: $\sqrt{b}$. Untuk catur, dengan $b = 35$ menjadi sekitar 6.
  - Pencarian 2x lebih dalam daripada algoritma minimax untuk waktu yang sama.

Outline

Keterbatasan sumber daya

- Biasanya dalam suatu permainan ada batasan waktu
- Andaikan ada agent bermain catur yang diberi 100 detik untuk "berpikir" tiap langkah.
- Mis. bisa memroses 10^4 node/detik $\rightarrow 10^6 node/langkah$
- Kita bisa melakukan aproksimasi menggunakan:
  - Evaluation function: *prediksi* dari nilai utility (tidak perlu sampai ke terminal state)
  - Cutoff: batasi depth yang diproses ($\approx$ IDS), bisa juga quiescence search

Evaluation function

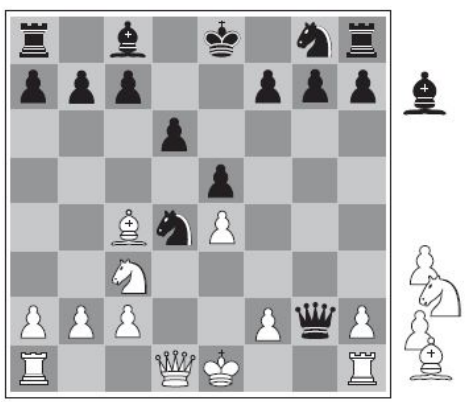
Biasanya, evaluation function berupa kombinasi linier dari fitur-fitur state:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

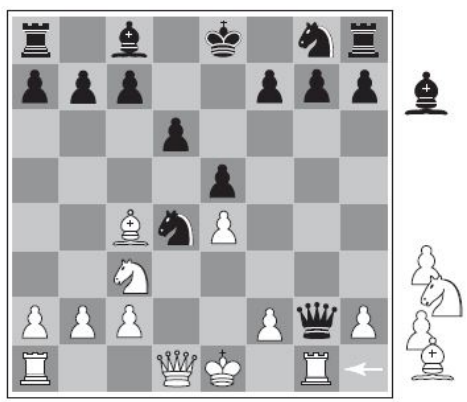
di mana w_i: bobot, f_i: fitur dari posisi.

Mis. untuk catur: w_i bisa merepresentasikan nilai material bidak (pion=1, kuda=3, benteng=5, ratu=9) dan f_i "advantage" bidak tersebut terhadap lawannya.

- $w_1 = 1$, f_1 = jumlah pion putih - jumlah pion hitam di papan
- $w_2 = 3$, f_2 = jumlah gajah putih - jumlah gajah hitam di papan

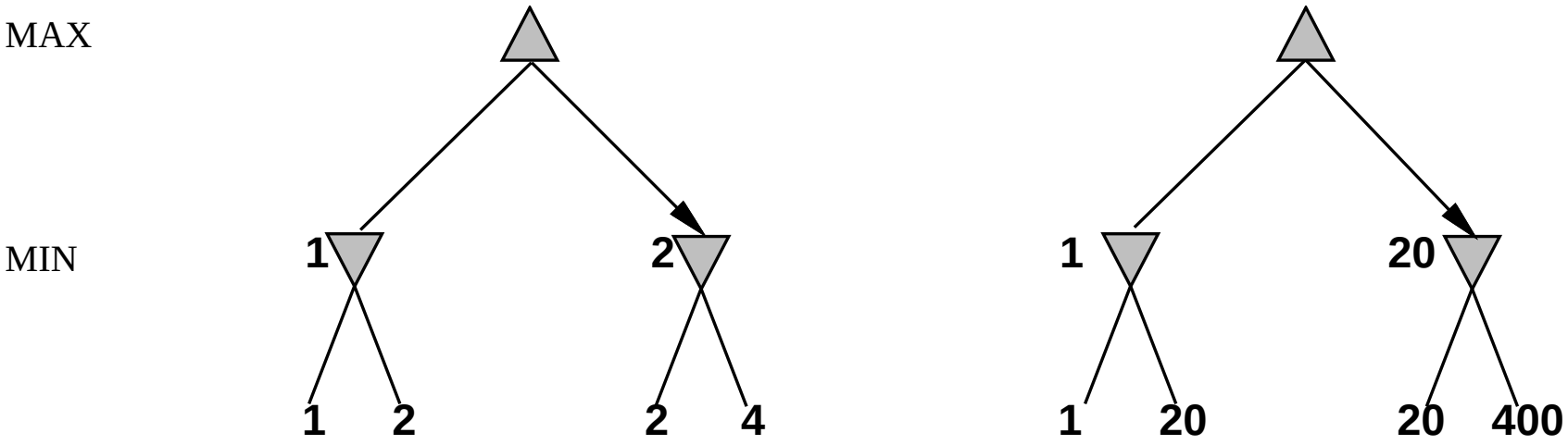


(a) White to move



(b) White to move

Perhatikan: nilai persisnya tidak penting



- Jika *Eval* diubah secara monotonik, hasil strategi tidak berubah
- Evaluation function pada game yang deterministic adalah fungsi *ordinal* (urutan/prioritas)

Melakukan search dengan cutoff

Pada state s dan di kedalaman d:

H-MINIMAX(s, d)=

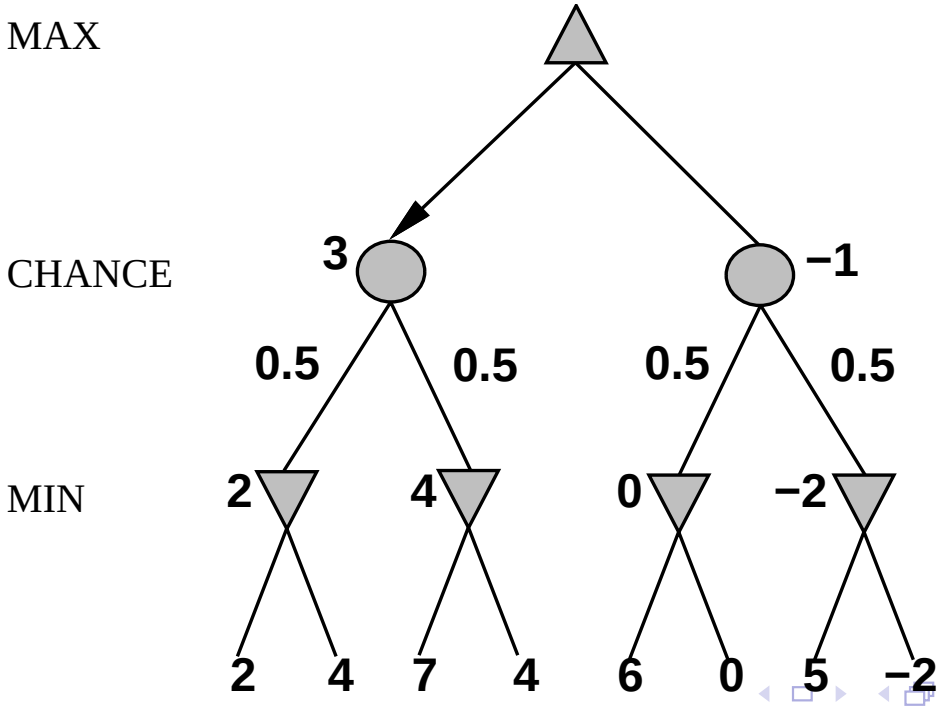
$$\begin{cases} \text{EVAL}(s) & \text{jika CUTOFF-TEST}(s, d) \\ max_{a \in Actions(s)} \text{ H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{jika PLAYER}(s) = \text{MAX} \\ min_{a \in Actions(s)} \text{ H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{jika PLAYER}(s) = \text{MIN} \end{cases}$$

- Untuk contoh catur: $b^m = 10^6, b = 35 \rightarrow m = 4$
- 4-ply lookahead $\approx$ pemain manusia pemula
- 8-ply lookahead $\approx$ pemain manusia "master", catur komputer rata-rata
- 14-ply lookahead $\approx$ Deep Blue, Garry Kasparov, ...

Outline

Bermain dengan probabilitas

- Ada banyak permainan yang memiliki unsur stokastik (kemungkinan/*chance*/*random*): lemparan dadu, koin, kocokan kartu.
- Game tree perlu ditambahkan *chance node*.
- Perkiraan nilai (*expected value*) untuk semua kemungkinan.



Generalisasi Fungsi Minimax dengan Chance Node

ExpectiMinimax(s)=

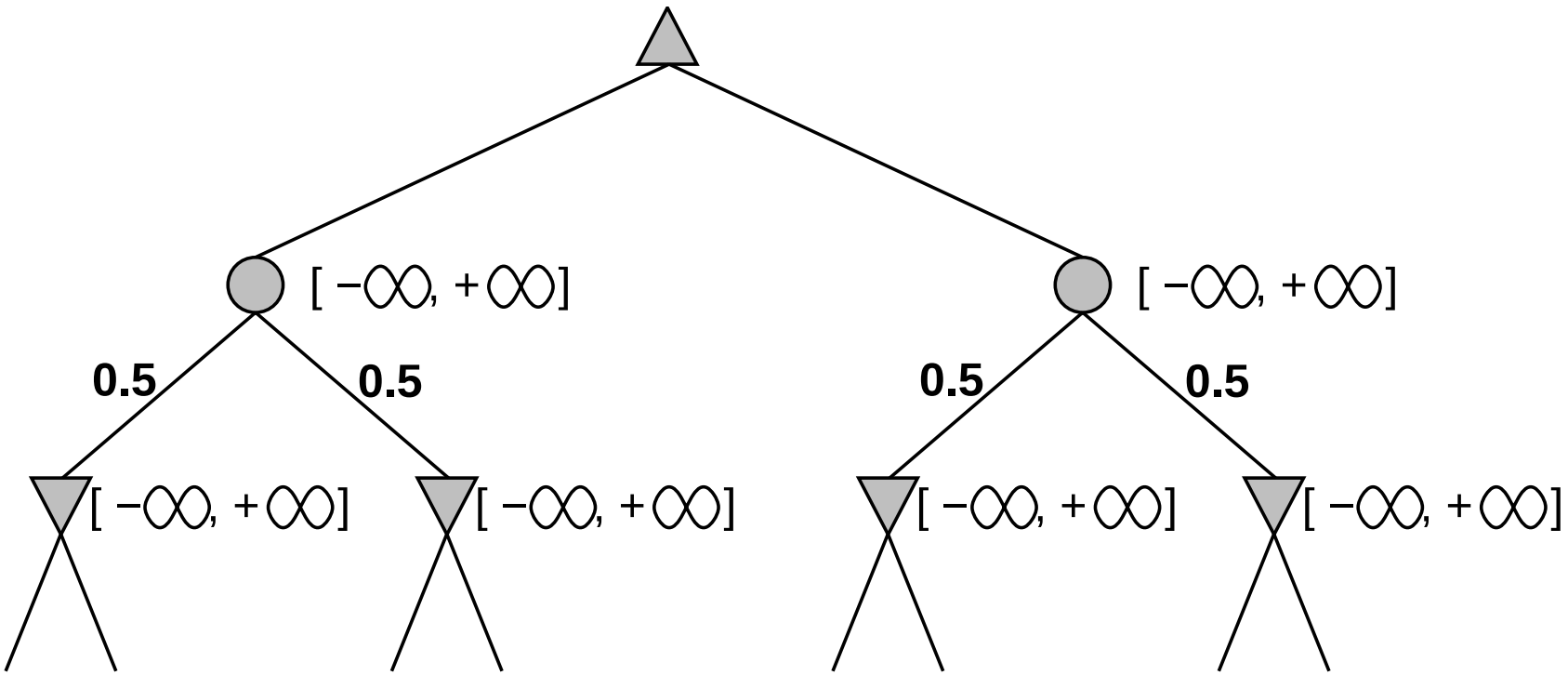
Utility(s)	jika Terminal-Test(s)
max_a ExpectiMinimax(Result(s, a))	jika Player(s) = Max
min_a ExpectiMinimax(Result(s, a))	jika Player(s) = Min
$\sum_r P(r) \times$ ExpectiMinimax(Result(s, r))	jika Player(s) = Chance

- Ide ExpectiMinimax sama dengan Minimax ditambah penanganan Chance.

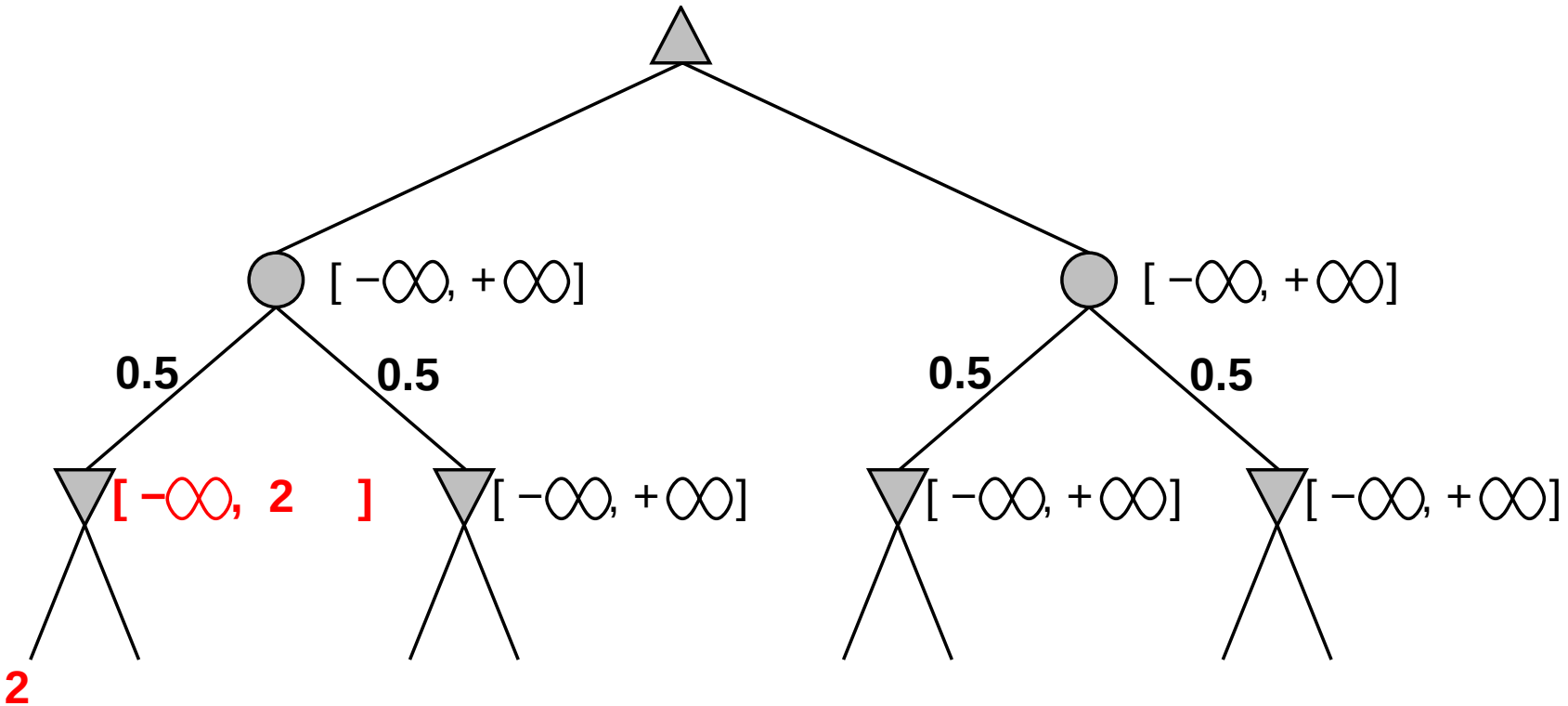
Algoritma ExpectiMinimax

- Time complexity MINIMAX: $O(b^m)$
- Dengan tambahan chance node, EXPECTIMINIMAX: $O(b^m \times n^m)$ (n=jumlah kemungkinan yang berbeda)
- Melempar 1 dadu: $n = 6$. Melempar 2 dadu: $n = 21$.
- Dengan lookahead melihat d langkah ke depan: $O(b^d \times n^d)$
- Karena stokastik, lookahead kurang efektif.

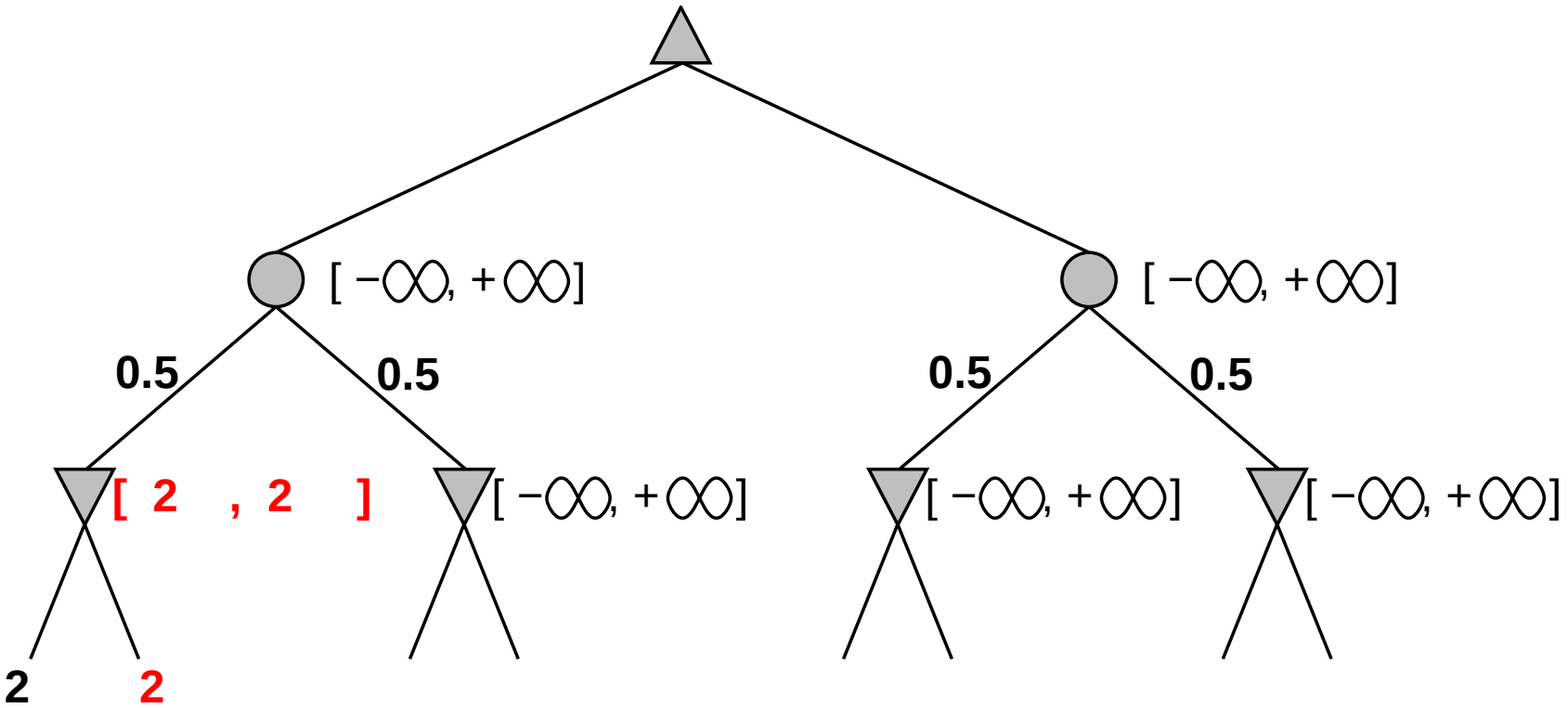
$\alpha - \beta$ pruning pada ExpectiMinimax



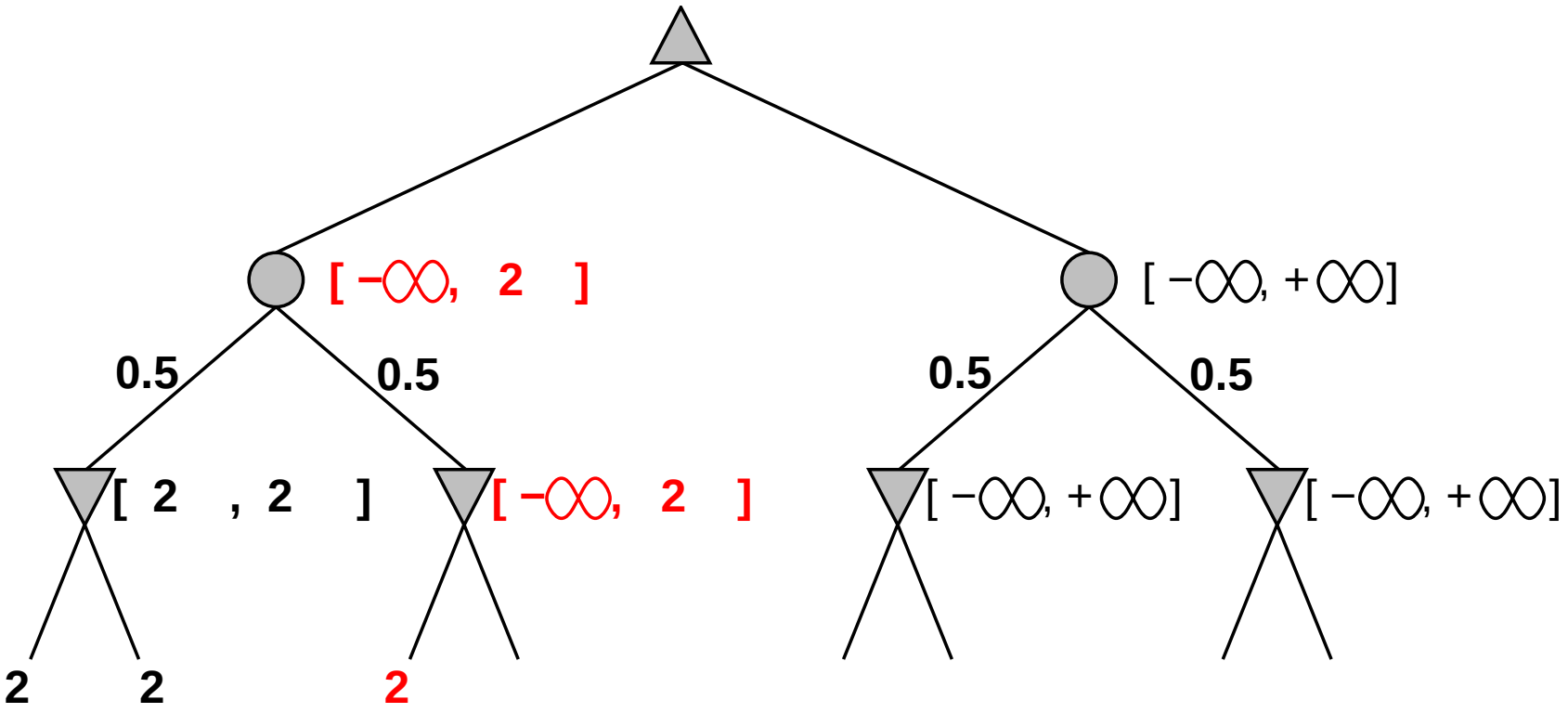
$\alpha - \beta$ pruning pada ExpectiMinimax



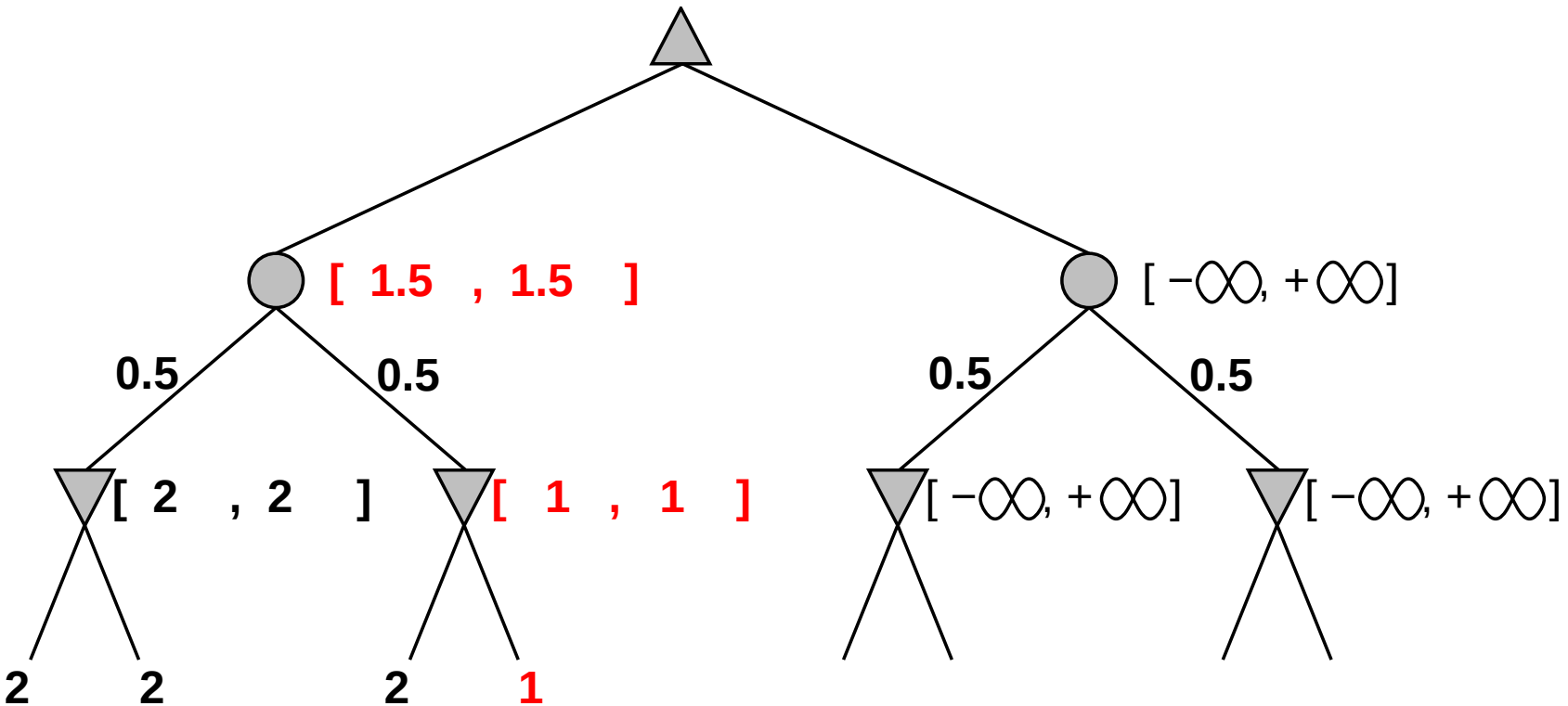
$\alpha - \beta$ pruning pada ExpectiMinimax



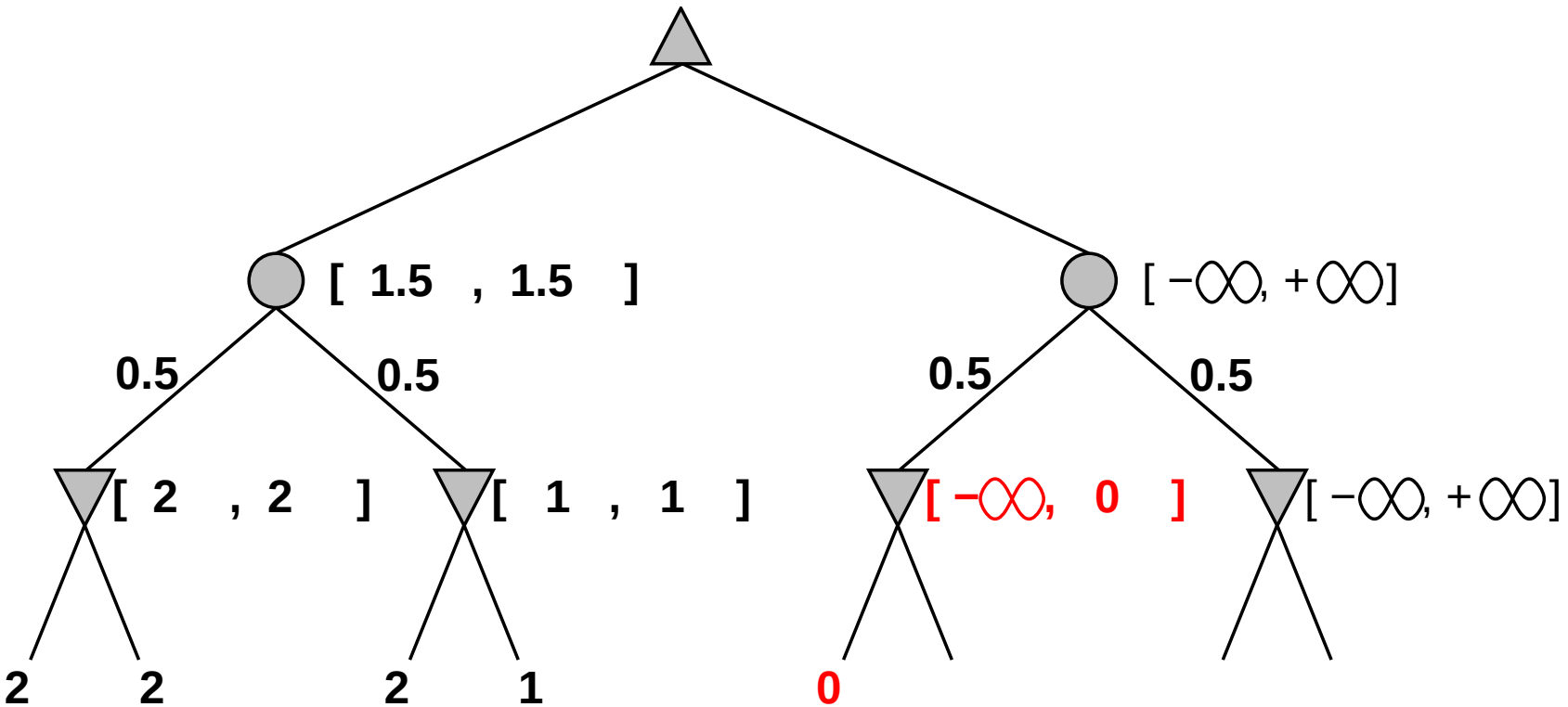
$\alpha - \beta$ pruning pada ExpectiMinimax



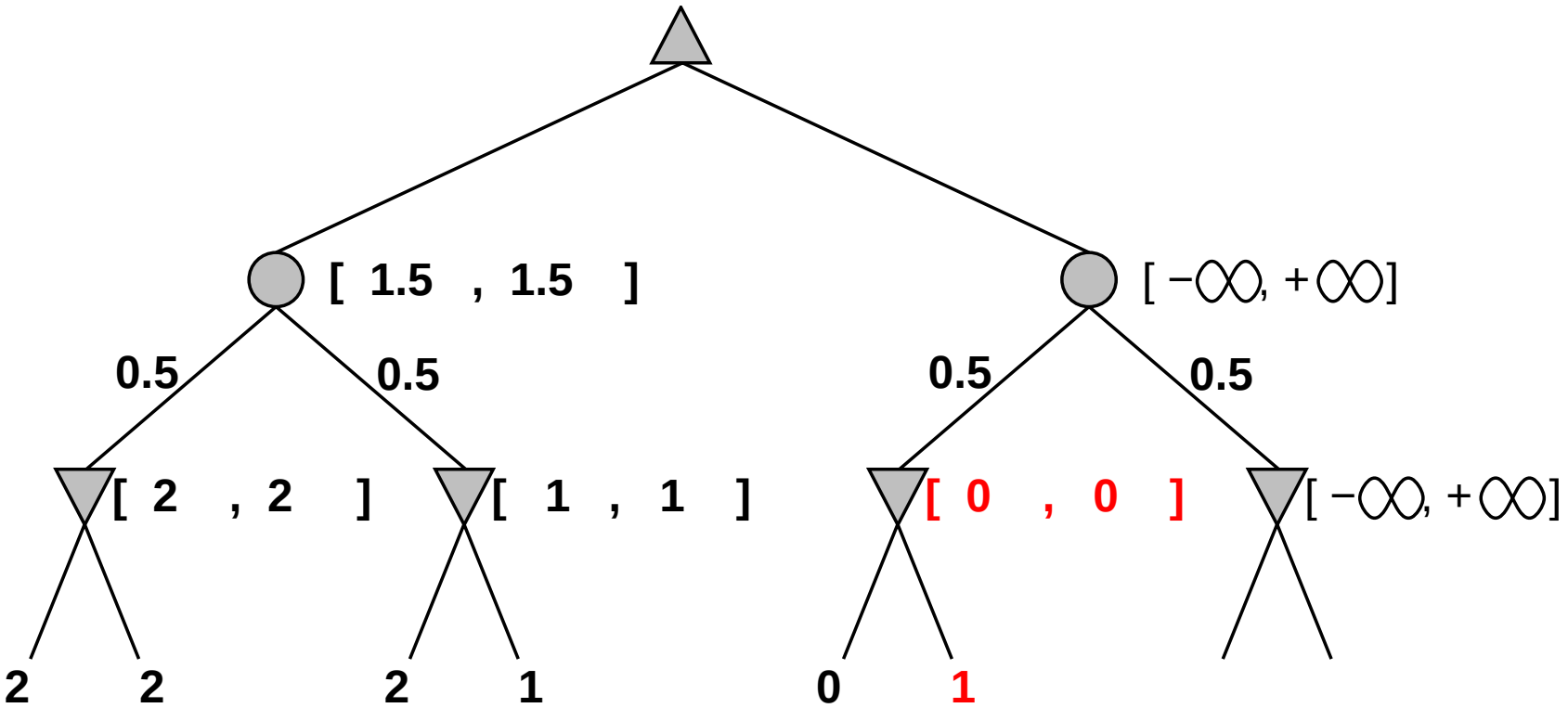
$\alpha - \beta$ pruning pada ExpectiMinimax



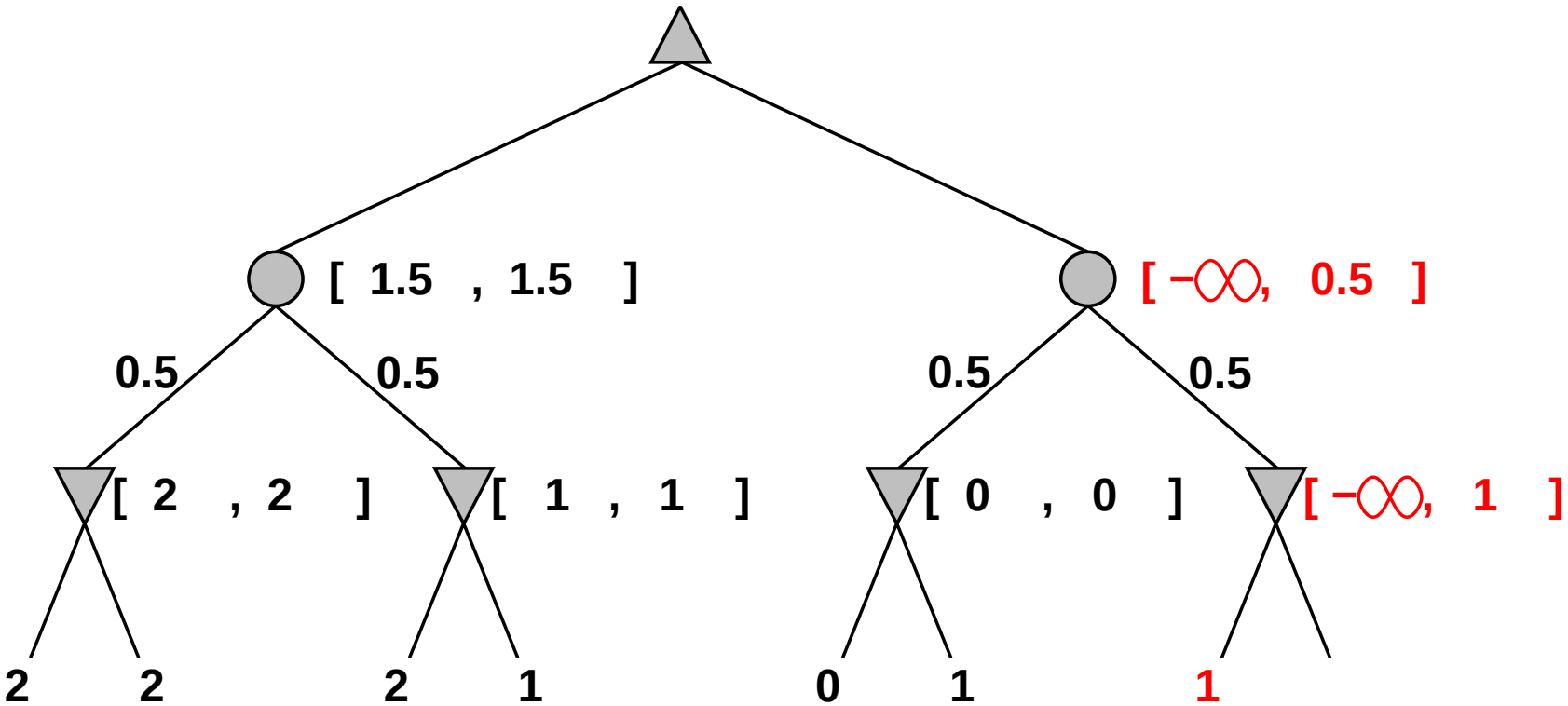
$\alpha - \beta$ pruning pada ExpectiMinimax



$\alpha - \beta$ pruning pada ExpectiMinimax

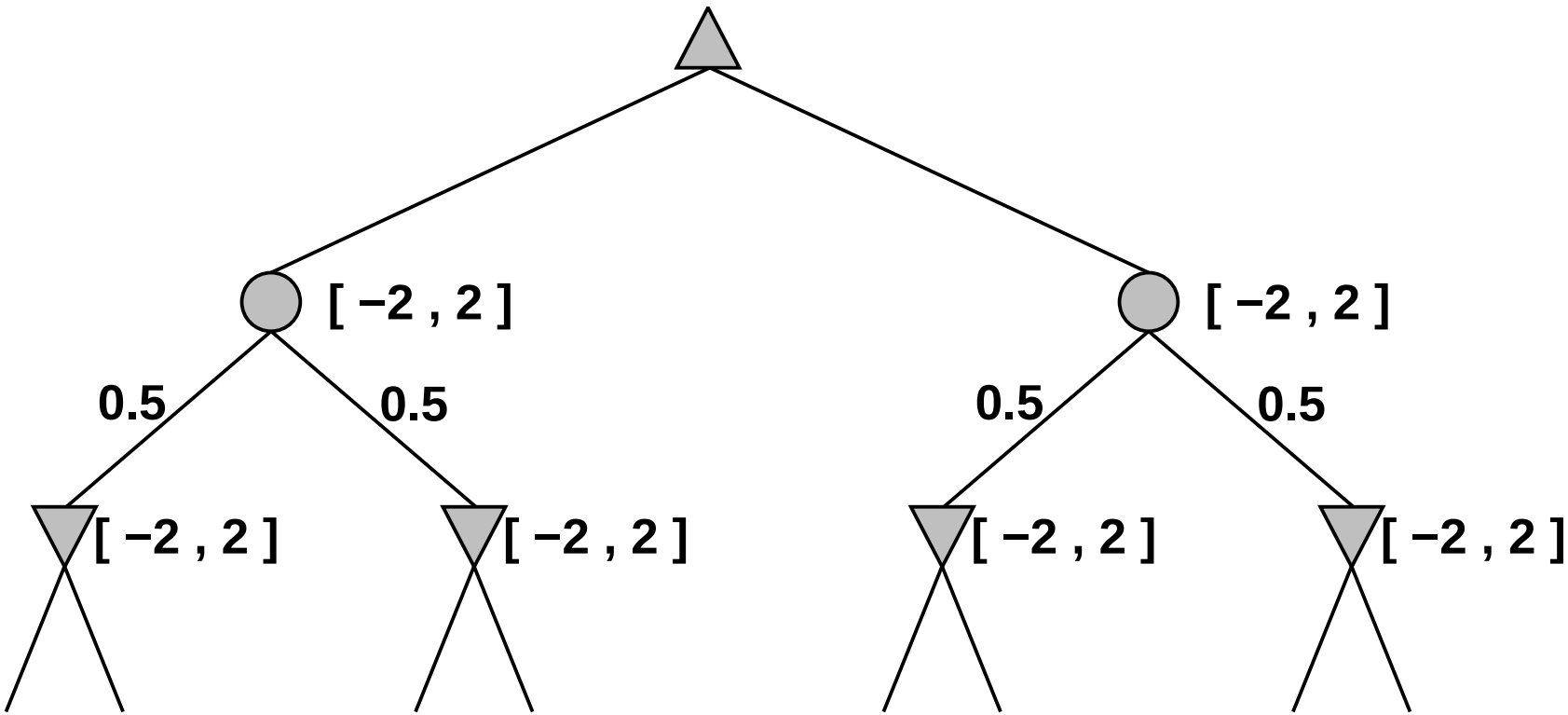


$\alpha - \beta$ pruning pada ExpectiMinimax



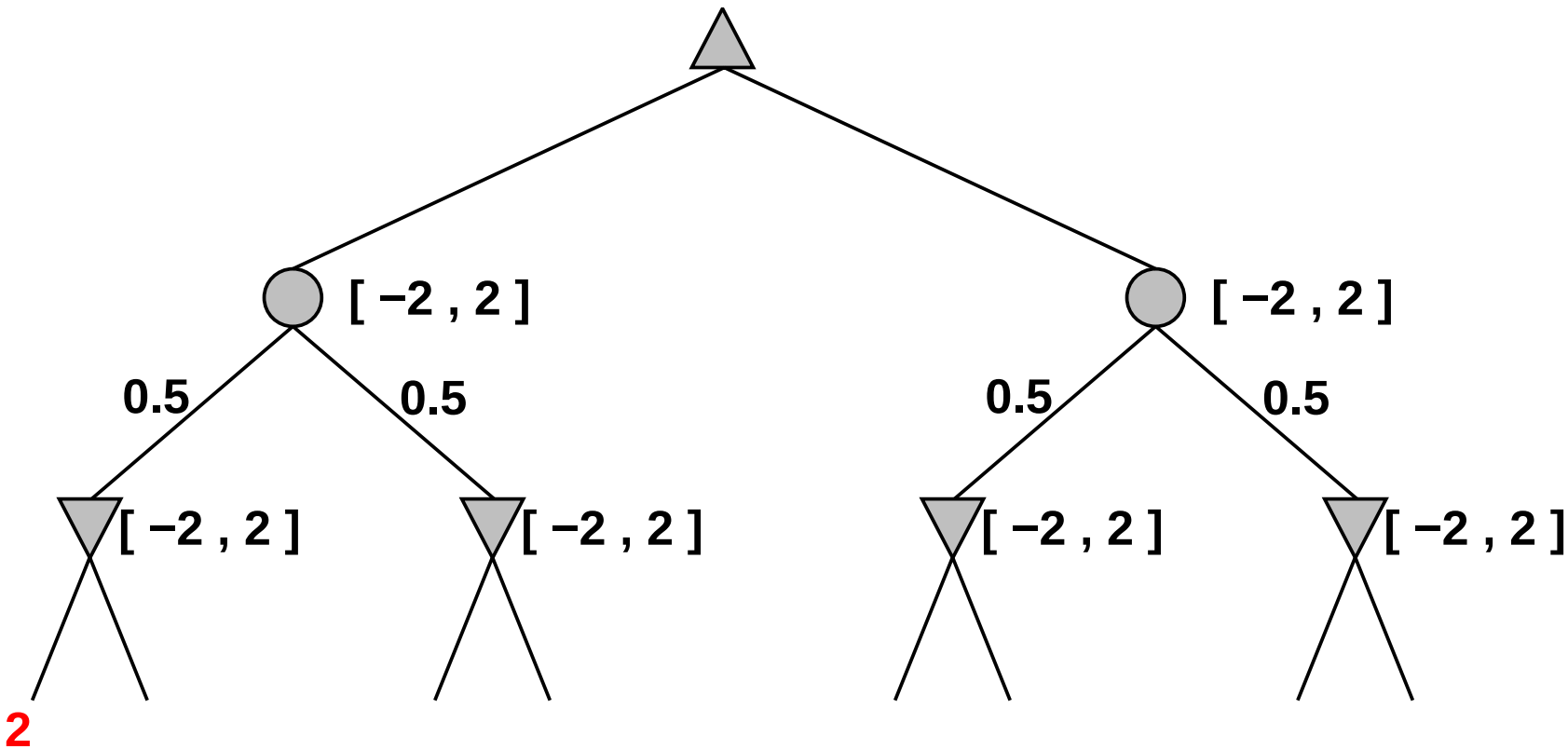
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



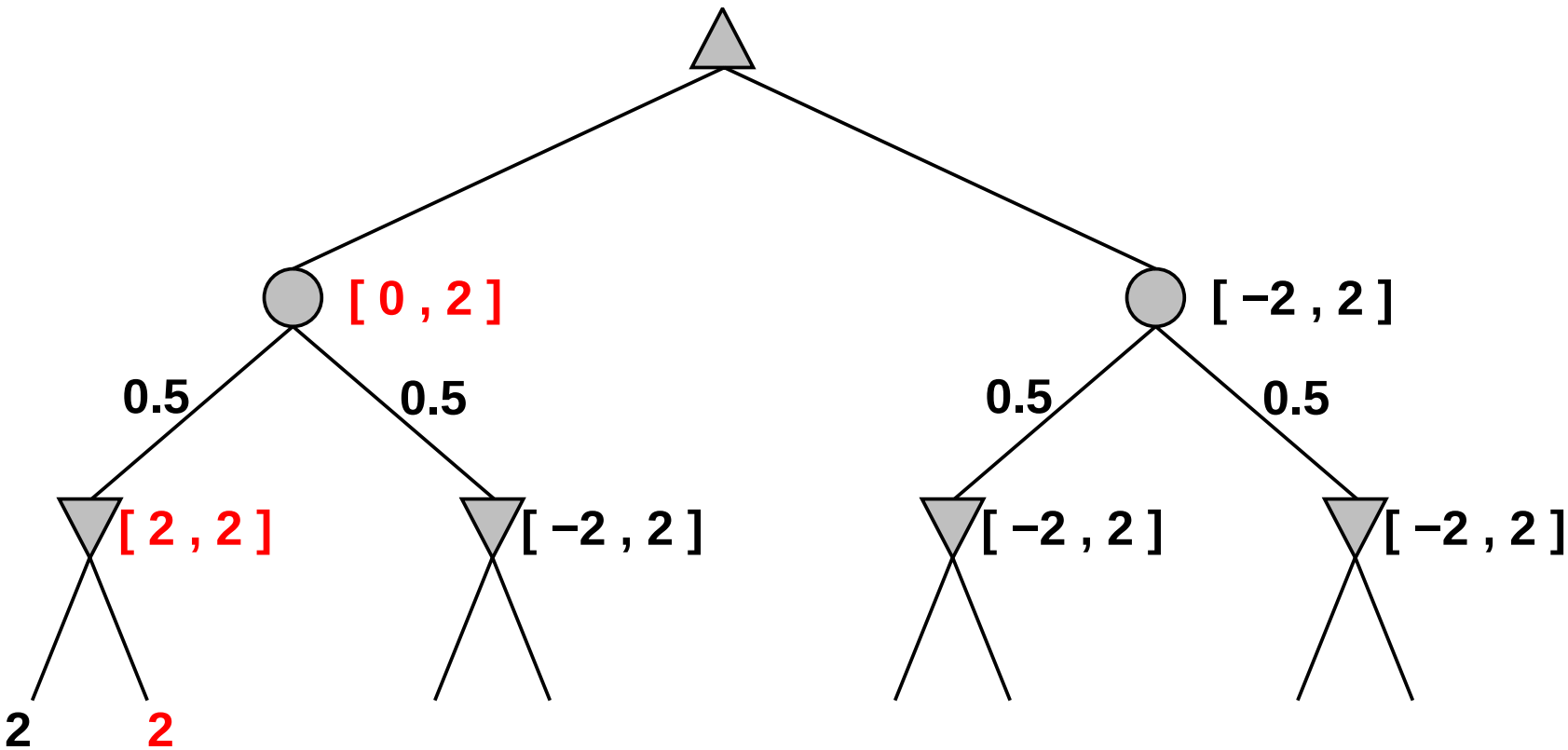
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



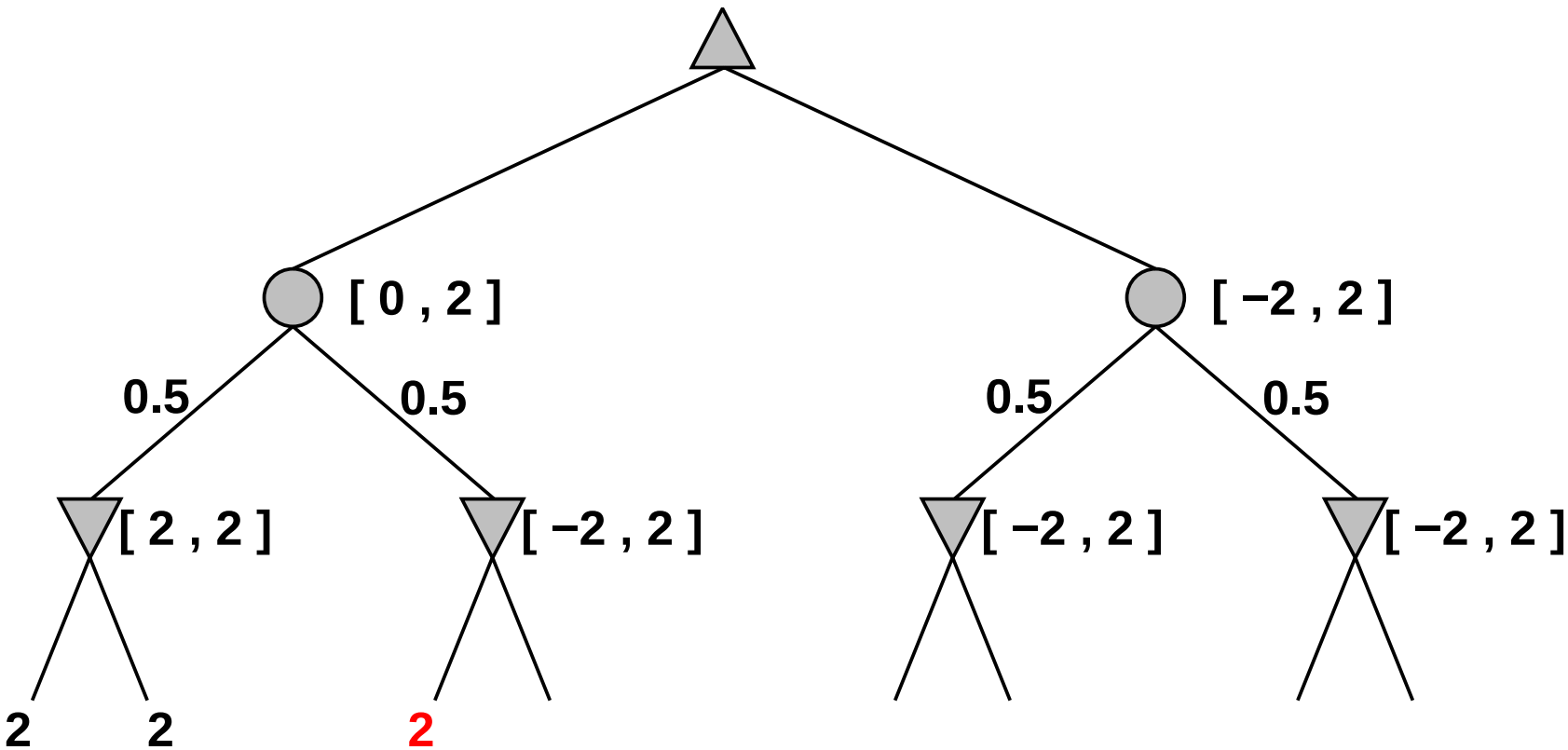
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



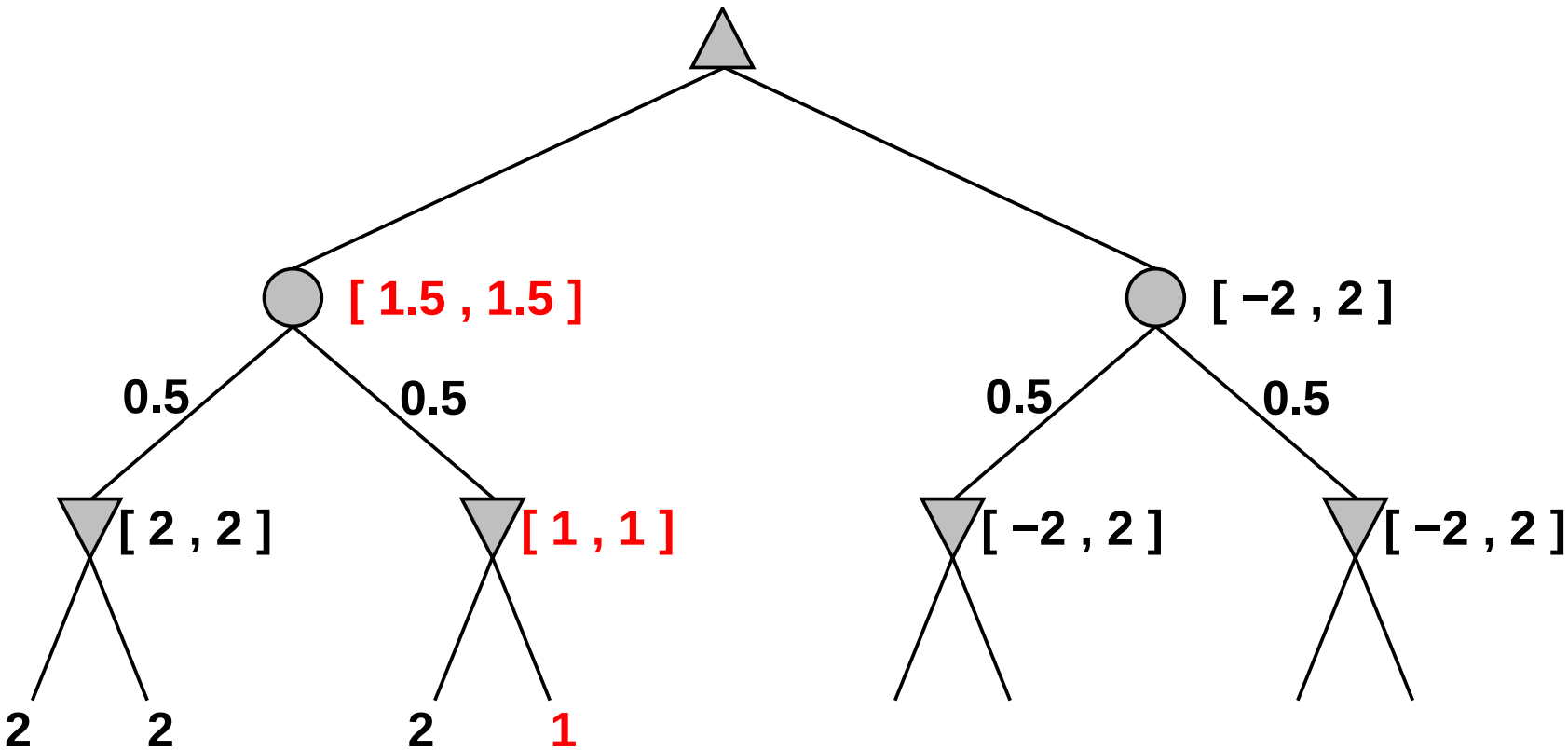
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



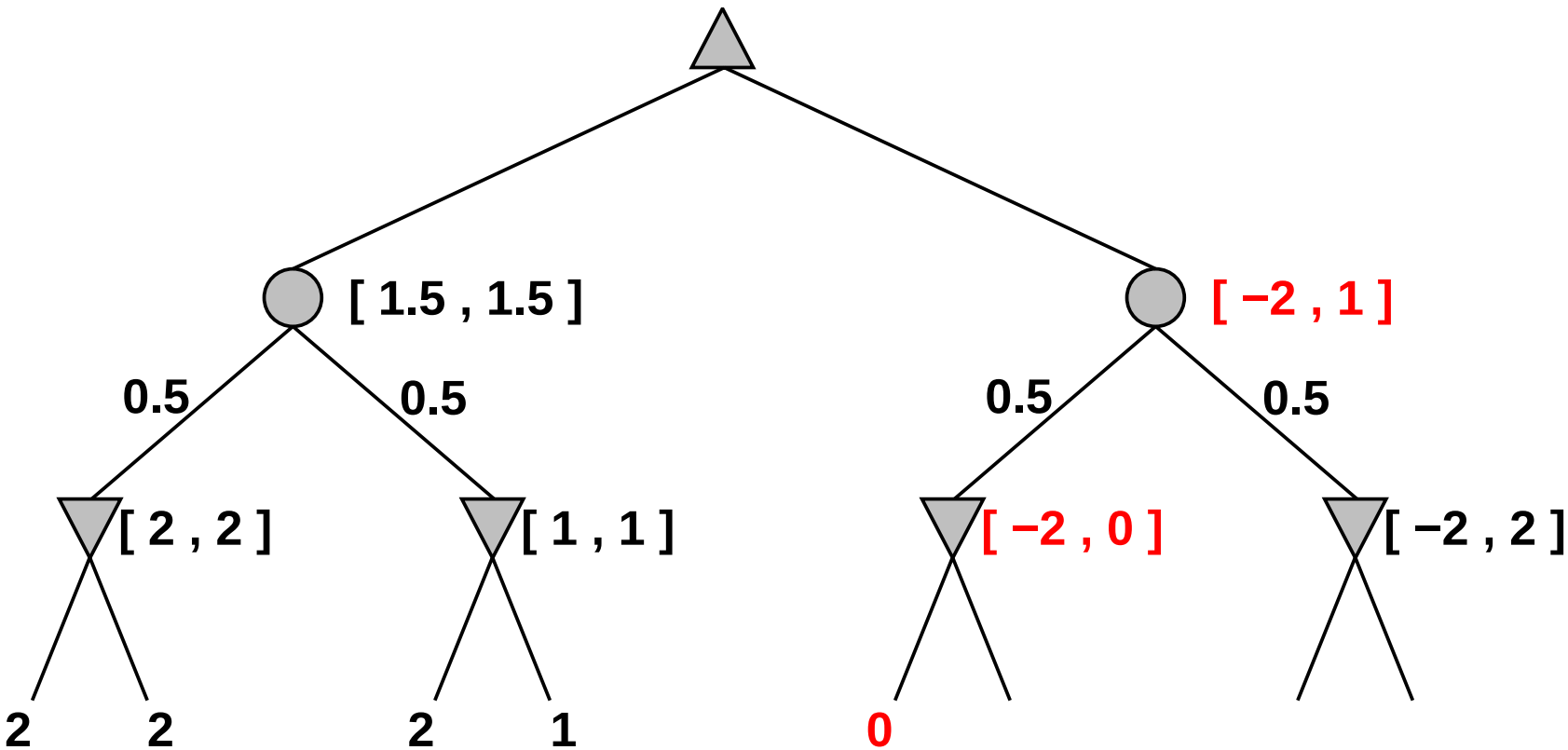
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



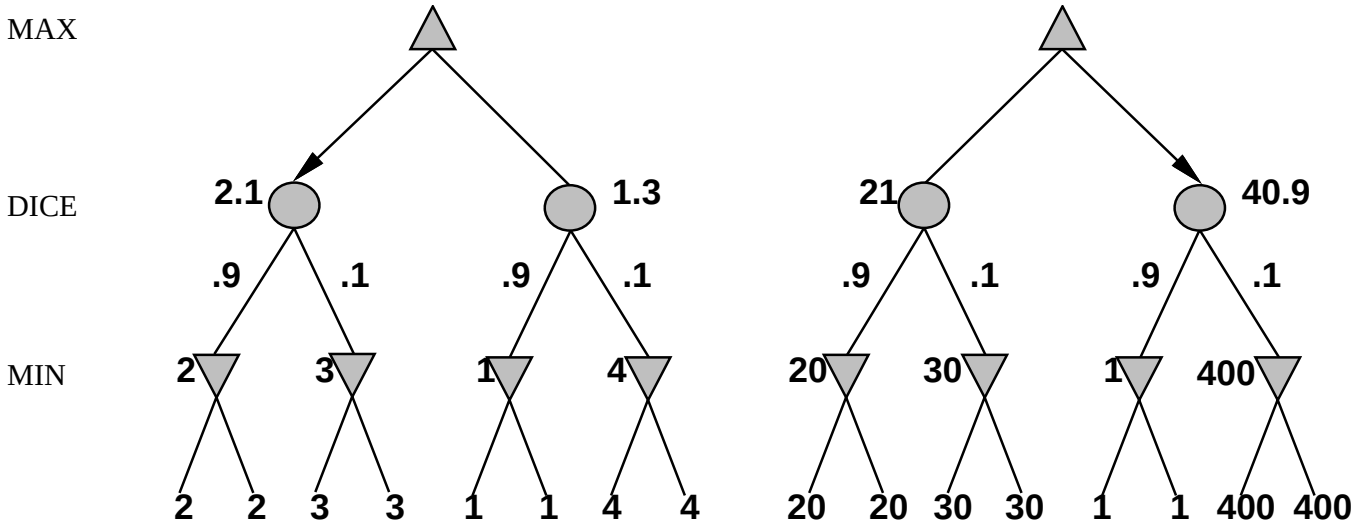
$\alpha - \beta$ pruning pada ExpectiMinimax

Pruning lebih efektif jika nilai *utility function* dibatasi:



Cutoff dan Evaluation Function pada ExpectiMinimax

ExpectiMinimax dapat diaproksimasi dengan cutoff dan evaluation function.



- Nilai absolut *Eval* dapat mempengaruhi hasil
- *Evaluation function* pada stochastic game harus proporsional dengan *expected utility* dari posisi.
- Tantangan buat AI: perbaiki evaluation function-nya, mis: menggunakan machine learning.
- Pendekatan lain: jangan coba semua kemungkinan, tapi cukup di-sample (Monte-Carlo).

Outline

State of the art

- Checkers: 1994, Chinook mengalahkan juara dunia selama 40 tahun, Marion Tinsley. Sejak 2007, Chinook mampu bermain (perfect play) dengan alpha-beta search yang dikombinasikan dengan database dari 39 triliun posisi endgame.
- Catur: 1997, Deep Blue mengalahkan juara dunia Gary Kasparov dalam pertarungan 6-ronde. Bisa memroses 200 juta node/detik, evaluation function canggih, dan metode lain sehingga lookahead bisa mencapai 40-ply.
- Othello: 1997, program Logistello mengalahkan juara dunia, Takeshi Murakami, 6-0.
- Go: Branching factor $b \geq 361$. October 2015, Google AplhaGo program mengalahkan juara Eropa (Fan Hui).
  9 Maret 2016, AlphaGo memenangkan game atas juara dunia (Lee Se-dol): 4-1.

Ringkasan

- Adversarial search adalah masalah search melawan agent lain dengan conflicting goal → *game*
- Optimal play: strategi optimal yang memberikan hasil terbaik, asumsi lawan musuh optimal.
- Algoritma MINIMAX: secara teoritis memberikan strategi optimal. Dalam kenyataannya terlalu mahal *computational cost*-nya
- Pruning (memangkas) tree dengan alpha-beta pruning
- Aproksimasi dengan evaluation function dan cutoff: lihat *m* langkah ke depan, perkiraan nilai *utility* dengan *evaluation function*
- Unsur *random* bisa diselesaikan dengan EXPECTIMINIMAX: lebih lambat dari MINIMAX → pruning dan aproksimasi