Hry a základní herní strategie

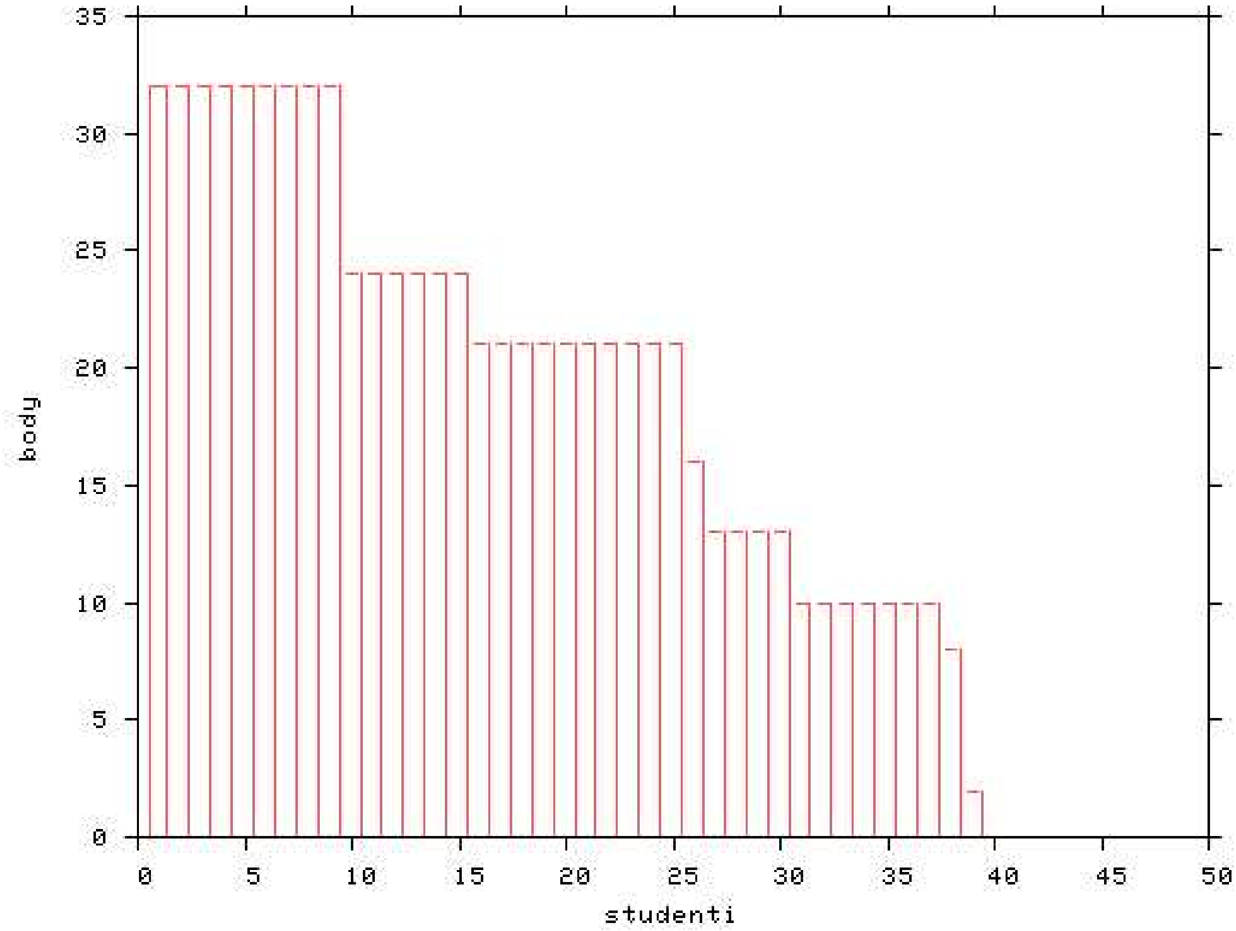
Aleš Horák

E-mail: `hales@fi.muni.cz`

`http://nlp.fi.muni.cz/uui/`

Obsah:

Statistické výsledky průběžné písemky



průběžná písemka PB016

46 studentů

Body	Počet studentů
32	9
24	6
21	10
16	1
13	4
10	7
8	1
2	1
0	7

Hry × Prohledávání stavového prostoru

Multiagentní prostředí:

- agent musí brát v úvahu akce jiných agentů $\longrightarrow$ jak ovlivní jeho vlastní prospěch
- vliv ostatních agentů – prvek náhody
- kooperativní × soupeřící multiagentní prostředí (MP)

Hry:

- matematická teorie her (odvětví ekonomie) – kooperativní i soupeřící MP, kde vliv všech agentů je *významný*
- hra v UI = obv. deterministické MP, 2 střídající se agenti, výsledek hry je vzájemně opačný nebo shoda

Algoritmy soupeřícího prohledávání (*adversarial search*):

- oponent dělá dopředu neurčitelné tahy $\longrightarrow$ řešením je strategie, která počítá se všemi možnými tahy protivníka
- časový limit $\Rightarrow$ zřejmě nenajdeme optimální řešení $\longrightarrow$ hledáme lokálně optimální řešení

HRY A UI – HISTORIE

- Babbage, 1846 – počítač porovnává přínos různých herních tahů
- von Neumann, 1944 – algoritmy perfektní hry
- Zuse, Wiener, Shannon, 1945–50 – přibližné vyhodnocování
- Turing, 1951 – první šachový program (jen na papíře)
- Samuel, 1952–57 – strojové učení pro zpřesnění vyhodnocování
- McCarthy, 1956 – prořezávání pro možnost hlubšího prohledávání

Hry a UI – historie

- Babbage, 1846 – počítač porovnává přínos různých herních tahů
- von Neumann, 1944 – algoritmy perfektní hry
- Zuse, Wiener, Shannon, 1945–50 – přibližné vyhodnocování
- Turing, 1951 – první šachový program (jen na papíře)
- Samuel, 1952–57 – strojové učení pro zpřesnění vyhodnocování
- McCarthy, 1956 – prořezávání pro možnost hlubšího prohledávání

řešení her je zajímavým předmětem studia $\leftarrow$ je obtížné:

průměrný faktor větvení v šachách $b = 35$

pro 50 tahů 2 hráčů ... prohledávací strom $\approx 35^{100} \approx 10^{154}$ uzlů ($\approx 10^{40}$ stavů)

Hry a UI – aktuální výsledky

- **dáma** – 1994 program *Chinook* porazil světovou šampionku Marion Tinsley. Používá úplnou databázi tahů pro ≤ 8 figur (443 748 401 247 pozic).
- **šachy** – 1997 porazil stroj *Deep Blue* světového šampiona Gary Kasparova. Stroj počítá 200 mil pozic/s, sofistikované vyhodnocování a nezveřejněné metody pro prozkoumávání některých tahů až do hloubky 40 tahů.
- **Othello** – světoví šampioni odmítají hrát s počítači, protože stroje jsou příliš dobré
- **Go** – světoví šampioni odmítají hrát s počítači, protože stroje jsou příliš slabé. V Go je $b > 300$, takže počítače mohou používat pouze znalostní bázi vzorových her.

Typy her

	deterministické	*s náhodou*
perfektní znalosti	šachy, dáma, Go, Othello	backgammon, monopoly
nepřesné znalosti		bridge, poker, scrabble

Hledání optimálního tahu

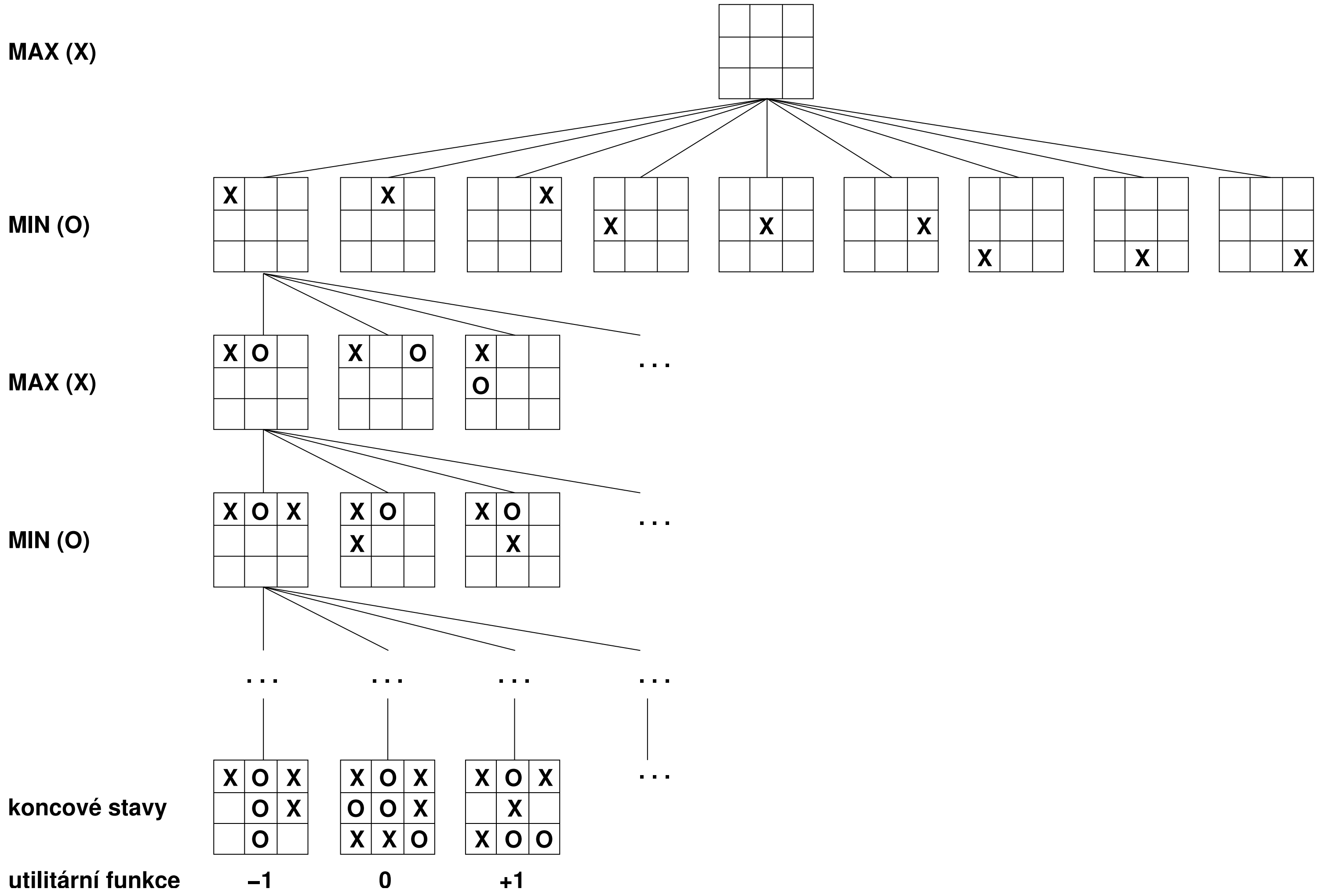
2 hráči – MAX a MIN, MAX je první na tahu a pak se střídají až do konce hry

hra = prohledávací problém:

- **počáteční stav** – počáteční herní situace + kdo je na tahu
- **přechodová funkce** – vrací dvojice (legální tah, výsledný stav)
- **ukončovací podmínka** – určuje, kdy hra končí, označuje koncové stavy
- **utilitární funkce** – numerické ohodnocení koncových stavů

HLEDÁNÍ OPTIMÁLNÍHO TAHU pokrač.

počáteční stav a přechodová funkce definují herní strom:



ALGORITMUS MINIMAX

MAX ($\triangle$) musí *prohledat* herní strom pro zjištění nejlepšího tahu proti MIN ($\bigtriangledown$)

$\longrightarrow$ zjistit nejlepší hodnotu minimax – zajišťuje *nejlepší výsledek* proti *nejlepšímu protivníkovi*

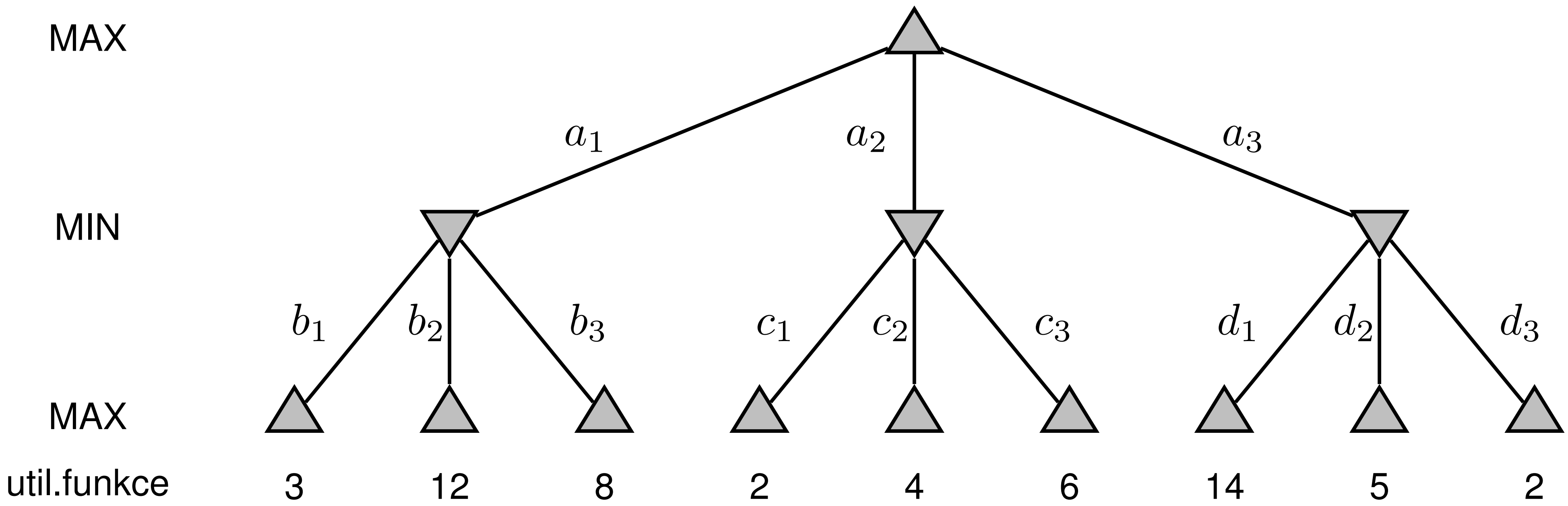
$$\text{Hodnota minimax}(n) = \begin{cases} \text{utility}(n) & \text{pro koncový stav } n \\ \max_{s \in \text{moves}(n)} \text{Hodnota minimax}(s) & \text{pro MAX uzel } n \\ \min_{s \in \text{moves}(n)} \text{Hodnota minimax}(s) & \text{pro MIN uzel } n \end{cases}$$

Algoritmus Minimax pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)

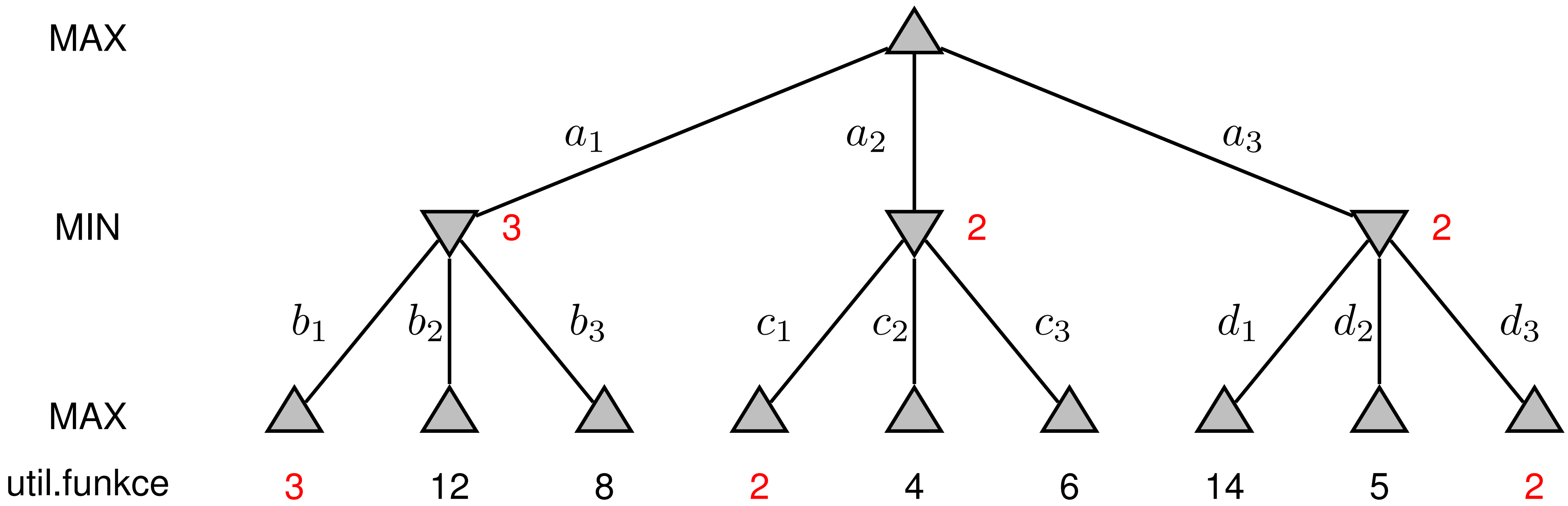
ALGORITMUS MINIMAX pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



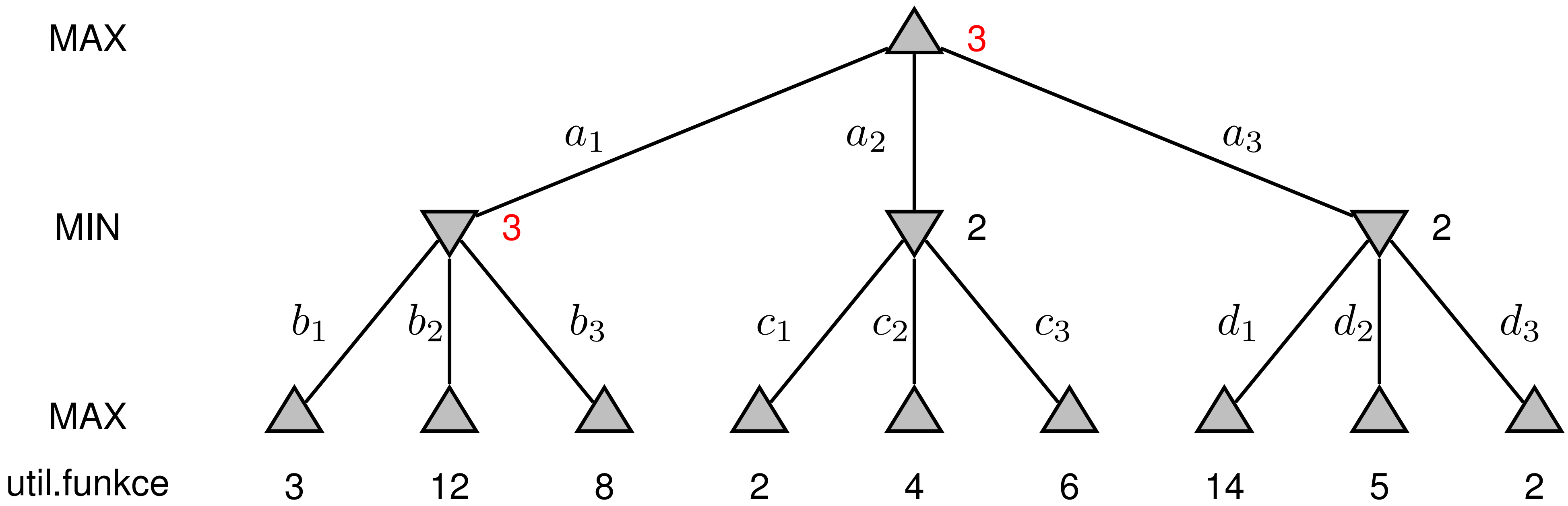
ALGORITMUS MINIMAX pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



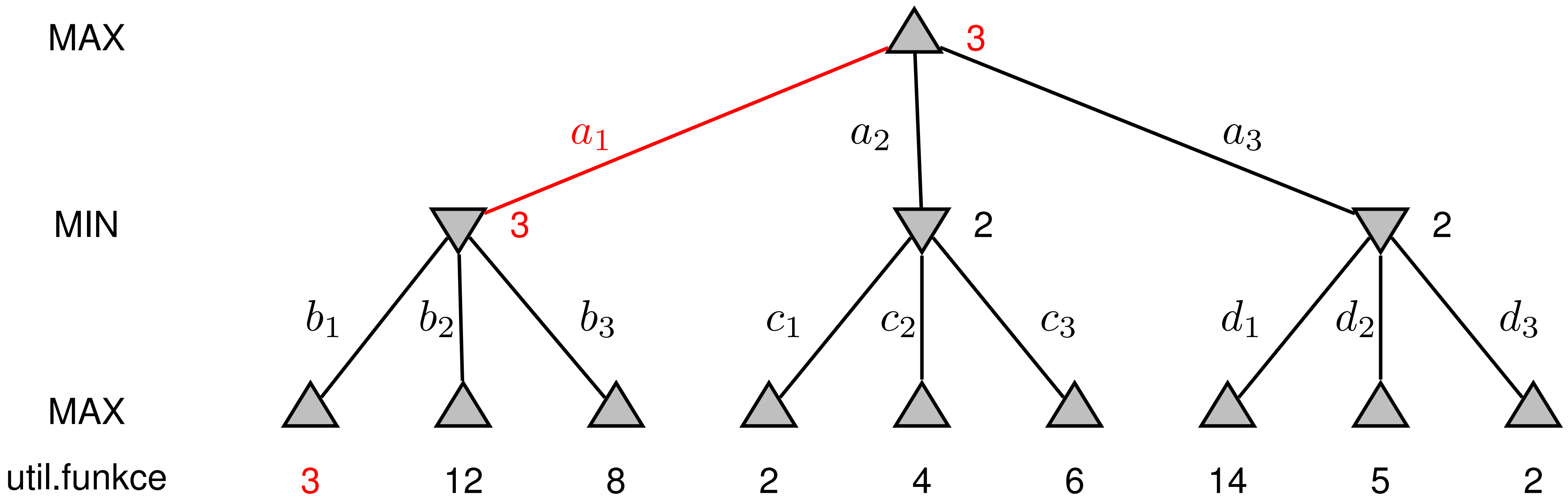
Algoritmus Minimax pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



ALGORITMUS MINIMAX pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



ALGORITMUS MINIMAX pokrač.

```
% minimax( Pos , BestSucc , Val ):
%    Pos is a position , Val is its minimax value ;
%    best move from Pos leads to position BestSucc
minimax( Pos, BestSucc, Val) :-
    moves( Pos, PosList), ! ,                 % Legal moves in Pos produce PosList
    best( PosList , BestSucc, Val)
    ;
    staticval ( Pos, Val ).                   % Pos has no successors : evaluate statically

best ( [ Pos], Pos, Val)  :-
    minimax( Pos, _, Val ), ! .
best ( [ Pos1 | PosList ], BestPos, BestVal) :-
    minimax( Pos1, _, Val1 ),
    best( PosList , Pos2, Val2),
    betterof ( Pos1, Val1 , Pos2, Val2 , BestPos, BestVal).
betterof ( Pos0, Val0, Pos1, Val1 , Pos0, Val0)  :-     % Pos0 better than Pos1
    min_to_move( Pos0),                                  % MIN to move in Pos0
    Val0 > Val1 , !                                      % MAX prefers the greater value
    ;
    max_to_move( Pos0),                                  % MAX to move in Pos0
    Val0 < Val1 , ! .                                    % MIN prefers the lesser value
betterof ( Pos0, Val0 , Pos1, Val1 , Pos1, Val1 ).      % Otherwise Pos1 better than Pos0
```

Algoritmus Minimax – vlastnosti

úplnost

optimálnost

časová složitost

prostorová složitost

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	
časová složitost	
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	$O(bm)$, prohledávání do hloubky

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	$O(bm)$, prohledávání do hloubky

šachy . . . $b \approx 35, m \approx 100 \Rightarrow$ přesné řešení není možné

$b^m = 10^6, b = 35 \Rightarrow m \approx 4$

4-tahy $\approx$ člověk-nováček

8-tahů $\approx$ člověk-mistr, typické PC

12-tahů $\approx$ Deep Blue, Kasparov

Algoritmus Alfa-Beta prořezávání

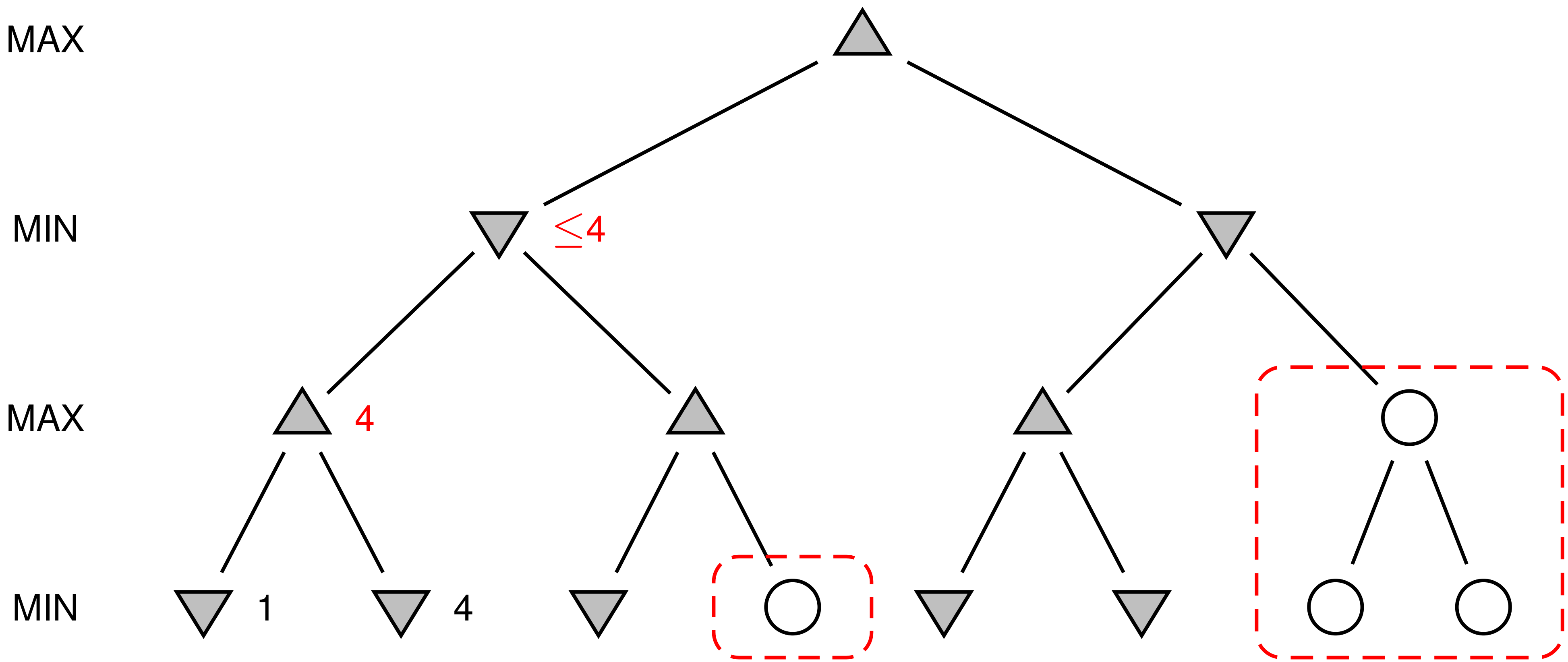
Příklad stromu, který zpracuje predikát **minmax**

Alfa-Beta odřízne expanzi některý uzlů $\Rightarrow$ Alfa-Beta procedura je efektivnější variantou minimaxu

Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**

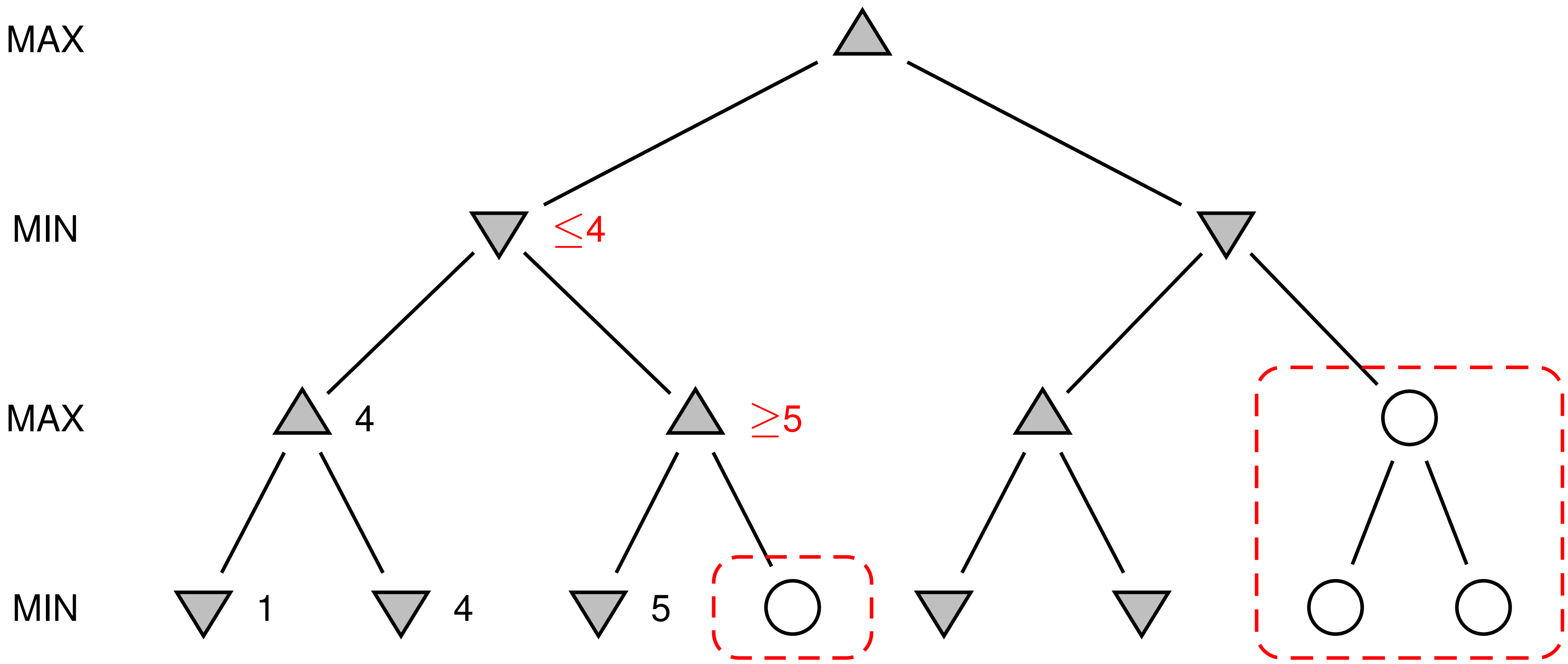
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**

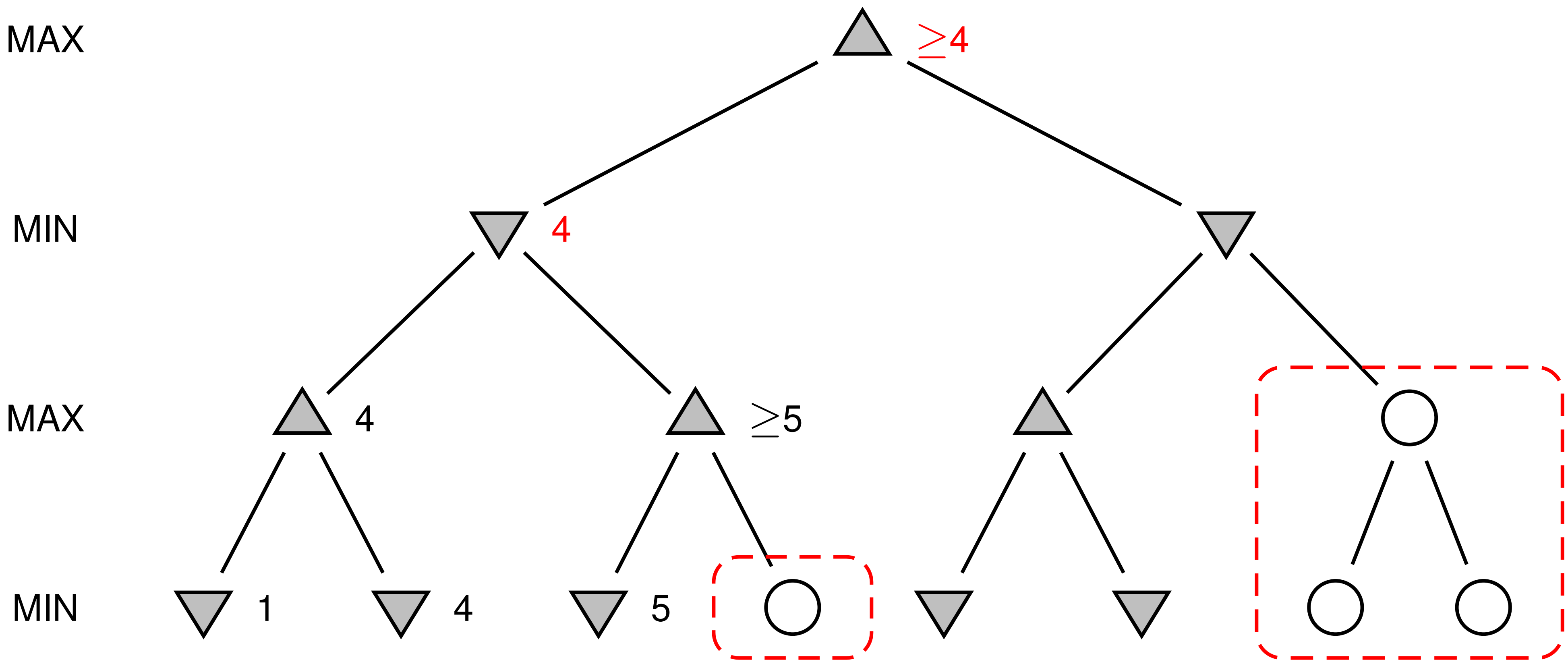
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**

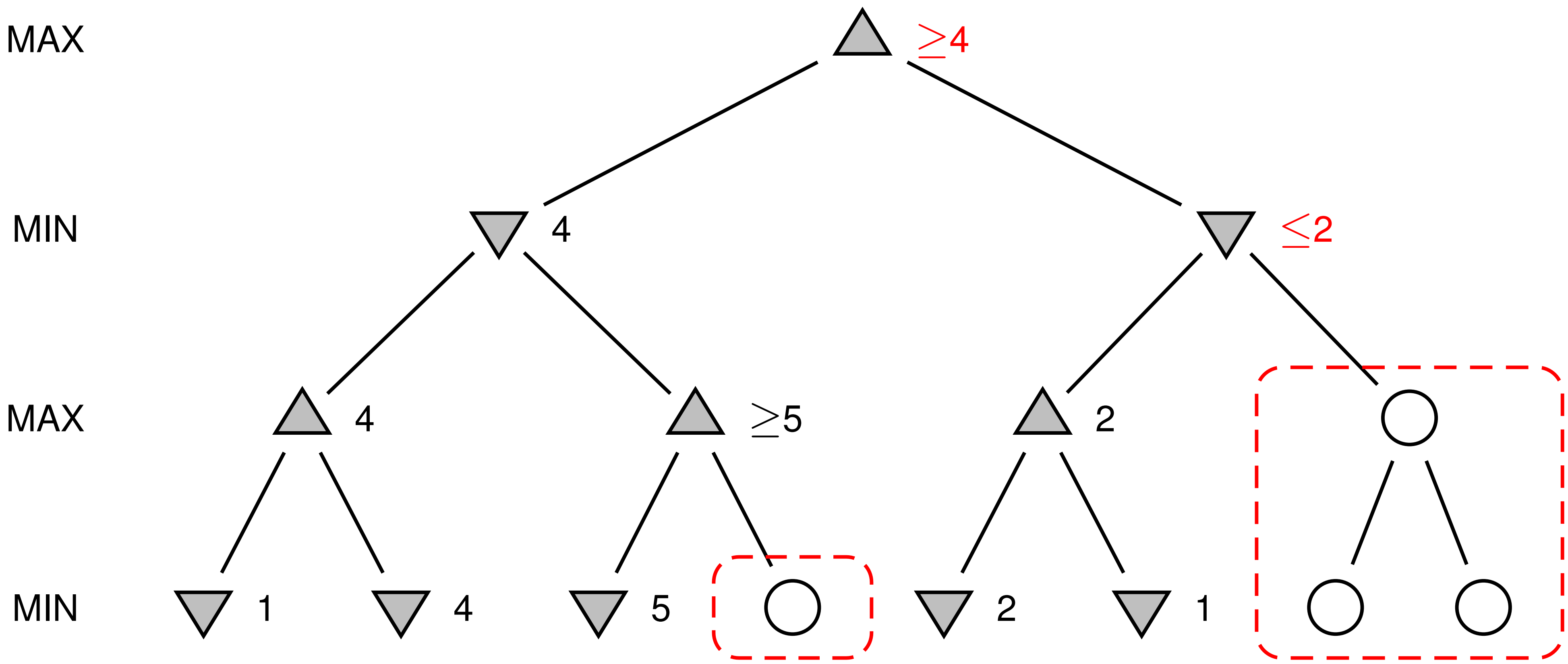
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**

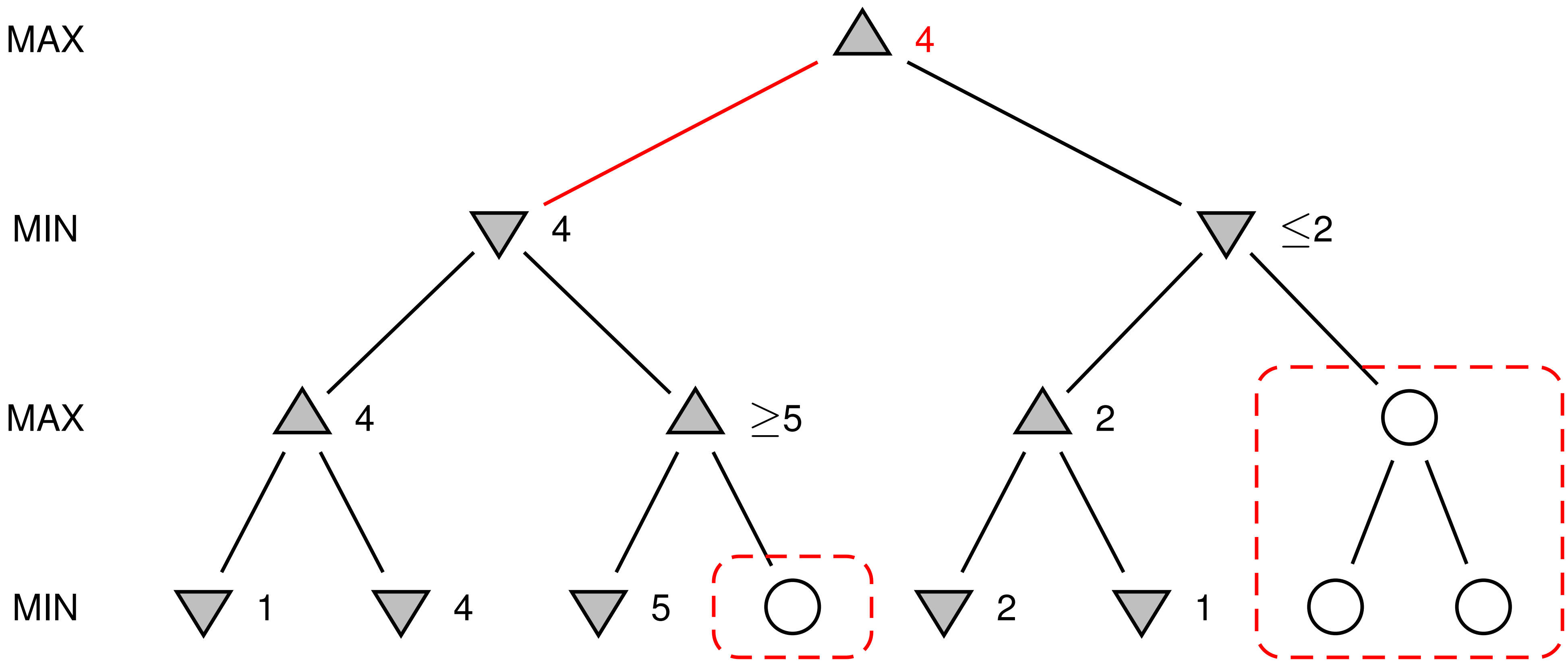
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**

Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta – vlastnosti

- prořezávání neovlivní výsledek $\Rightarrow$ je stejný jako u minimaxu
- dobré uspořádání přechodů (možných tahů) ovlivní efektivitu prořezávání
- v případě “nejlepšího” uspořádání časová složitost$= O(b^{m/2})$

  $\Rightarrow$ zdvojí hloubku prohledávání

  $\Rightarrow$ může snadno dosáhnout hloubky 8 v šachu, což už je použitelná úroveň

Algoritmus Alfa-Beta – vlastnosti

- prořezávání neovlivní výsledek $\Rightarrow$ je stejný jako u minimaxu
- dobré uspořádání přechodů (možných tahů) ovlivní efektivitu prořezávání
- v případě "nejlepšího" uspořádání časová složitost$= O(b^{m/2})$

  $\Rightarrow$ zdvojí hloubku prohledávání

  $\Rightarrow$ může snadno dosáhnout hloubky 8 v šachu, což už je použitelná úroveň

označení $\alpha - \beta$:

- $\alpha \ldots$ doposud nejlepší hodnota pro MAXe
- $\beta \ldots$ doposud nejlepší hodnota pro MINa
- $<\alpha, \beta> \ldots$ interval ohodnocovací funkce v průběhu výpočtu (na začátku $<-\infty, \infty>$)
-

minimax $\ldots V(P)$	$\alpha - \beta \ldots V(P, \alpha, \beta)$
když $V(P) \leq \alpha$	$V(P, \alpha, \beta) = \alpha$
když $\alpha < V(P) < \beta$	$V(P, \alpha, \beta) = V(P)$
když $V(P) \geq \beta$	$V(P, \alpha, \beta) = \beta$

Algoritmus Alfa-Beta prořezávání

```
alphabeta( Pos, Alpha, Beta, GoodPos, Val) :- moves( Pos, PosList), !,
    boundedbest( PosList, Alpha, Beta, GoodPos, Val);
    staticval ( Pos, Val ).                     % Static value of Pos

boundedbest( [Pos | PosList], Alpha, Beta, GoodPos, GoodVal) :-
    alphabeta( Pos, Alpha, Beta, _, Val),
    goodenough( PosList, Alpha, Beta, Pos, Val, GoodPos, GoodVal).

goodenough( [], _, _, Pos, Val, Pos, Val) :- ! .   % No other candidate
goodenough( _, Alpha, Beta, Pos, Val, Pos, Val) :-
    min_to_move( Pos), Val > Beta, !            % Maximizer attained upper bound
    ; max_to_move( Pos), Val < Alpha, !.        % Minimizer attained lower bound
goodenough( PosList, Alpha, Beta, Pos, Val, GoodPos, GoodVal) :-
    newbounds( Alpha, Beta, Pos, Val, NewAlpha, NewBeta), % Refine bounds
    boundedbest( PosList, NewAlpha, NewBeta, Pos1, Val1),
    betterof ( Pos, Val , Pos1, Val1 , GoodPos, GoodVal).

newbounds( Alpha, Beta, Pos, Val, Val, Beta) :-
    min_to_move( Pos), Val > Alpha, !.          % Maximizer increased lower bound
newbounds( Alpha, Beta, Pos, Val, Alpha, Val) :-
    max_to_move( Pos), Val < Beta, !.           % Minimizer decreased upper bound
newbounds( Alpha, Beta, _, _, Alpha, Beta).     % Otherwise bounds unchanged

betterof ( Pos, Val , Pos1, Val1 , Pos, Val) :- min_to_move( Pos), Val > Val1, !
    ; max_to_move( Pos), Val < Val1, !.         % Pos better than Pos1
betterof ( _, _, Pos1, Val1 , Pos1, Val1 ).     % Otherwise Pos1 better
```

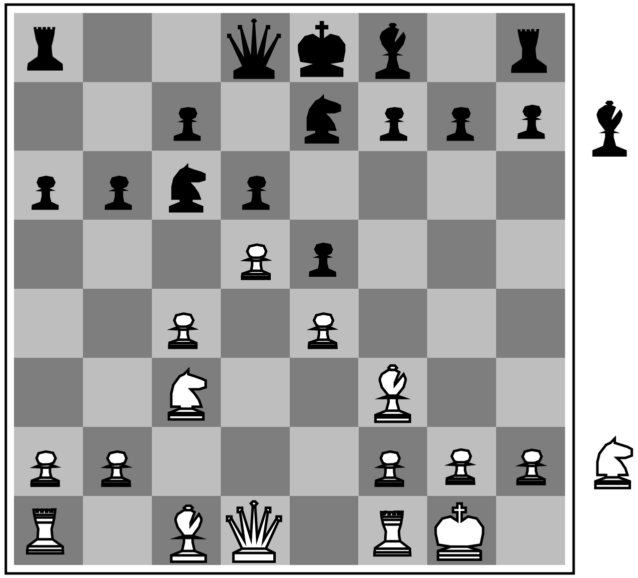
Časové omezení

předpokládejme, že máme 100 sekund + prozkoumáme 10^4 uzlů/s $\Rightarrow$ 10^6 uzlů na 1 tah

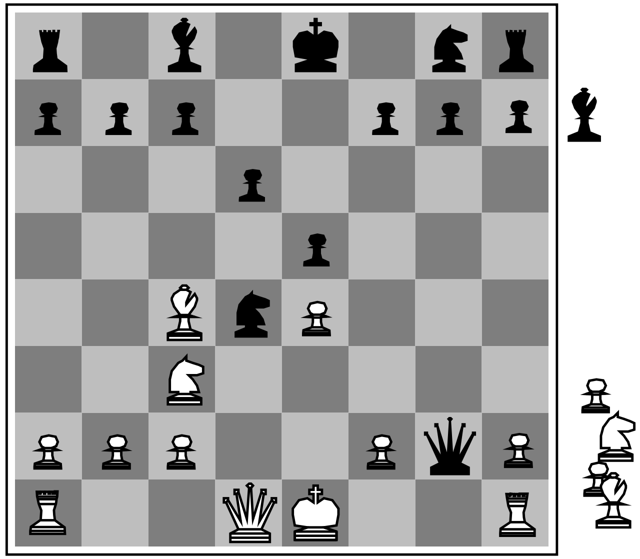
řešení:

- **ohodnocovací funkce** odhad přínosu pozice
- **ořezávací test** (*cutoff test*) – např. hloubka nebo hodnota ohodnocovací funkce

Ohodnocovací funkce

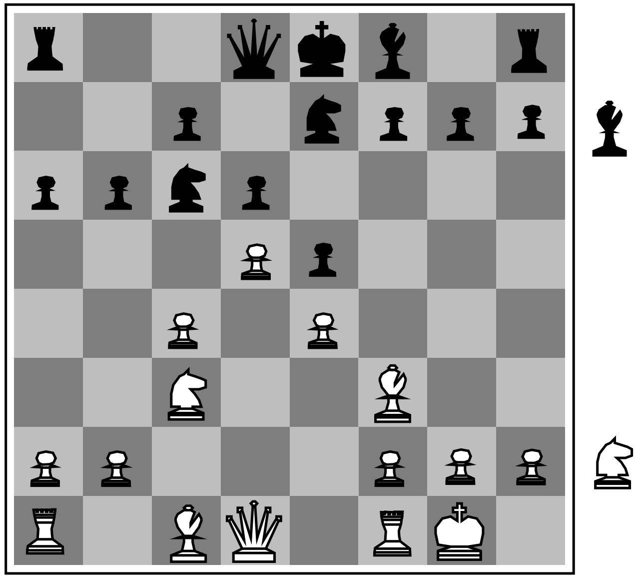


Černý na tahu
Bílý ma o něco lepší pozici

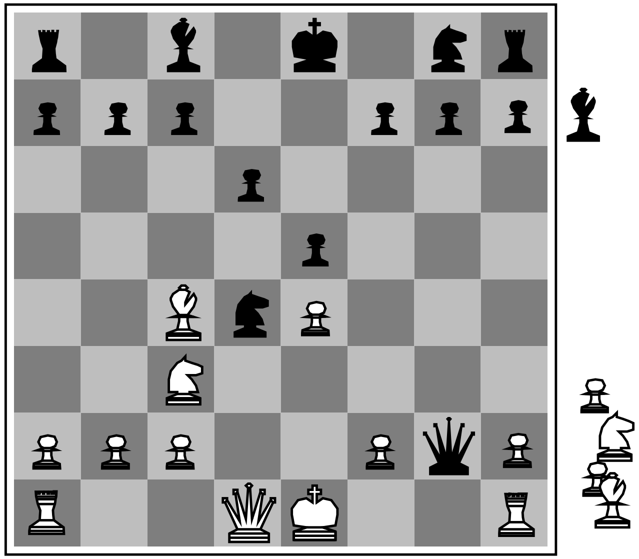


Bílý na tahu
Černý vítězí

Ohodnocovací funkce



Černý na tahu
Bílý ma o něco lepší pozici



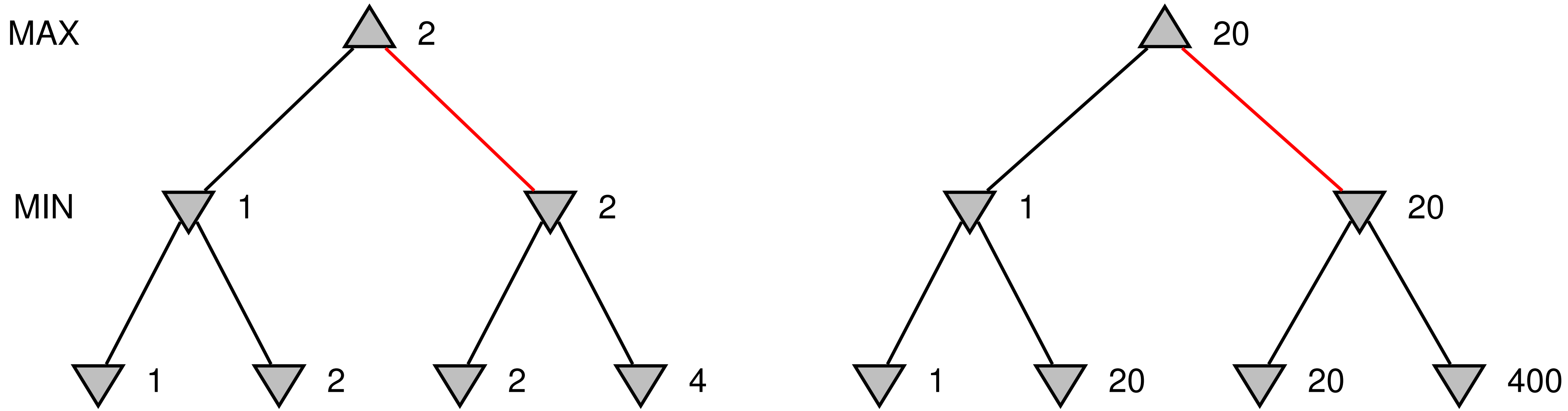
Bílý na tahu
Černý vítězí

Pro šachy typicky lineární vážený součet rysů

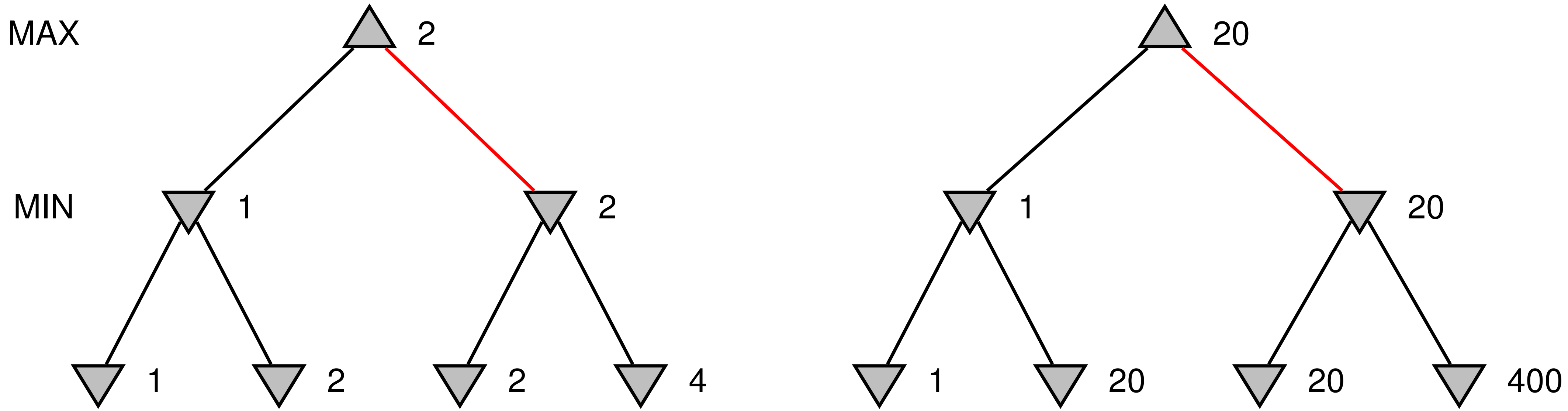
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

např. $w_1 = 9$
$f_1(s) =$ (počet bílých královen) $-$ (počet černých královen)
...

Ohodnocovací funkce – odchylky



Ohodnocovací funkce – odchylky



chová se stejně pro libovolnou monotónní transformaci fuknce $Eval$

záleží pouze na uspořádání $\longrightarrow$ ohodnocení v deterministické hře funguje jako ordinální funkce

Prohledávání s ořezávacím testem

minimax_cutoff je stejný jako **minimax** kromě:

1. koncový test ⟶ *ořezávací test*
2. utilitární funkce ⟶ *ohodnocovací funkce*

Prohledávání s ořezávacím testem

minimax_cutoff je stejný jako **minimax** kromě:

1. koncový test $\longrightarrow$ *ořezávací test*
2. utilitární funkce $\longrightarrow$ *ohodnocovací funkce*

další možnosti vylepšení:

➜ vyhodnocovat pouze klidné stavy (quiescent search)

➜ při vyhodnocování počítat s efektem horizontu – zvraty mimo prohledanou oblast

➜ dopředné ořezávání – některé stavy se ihned zahazují
bezpečné např. pro symetrické tahy nebo pro tahy hluboko ve stromu

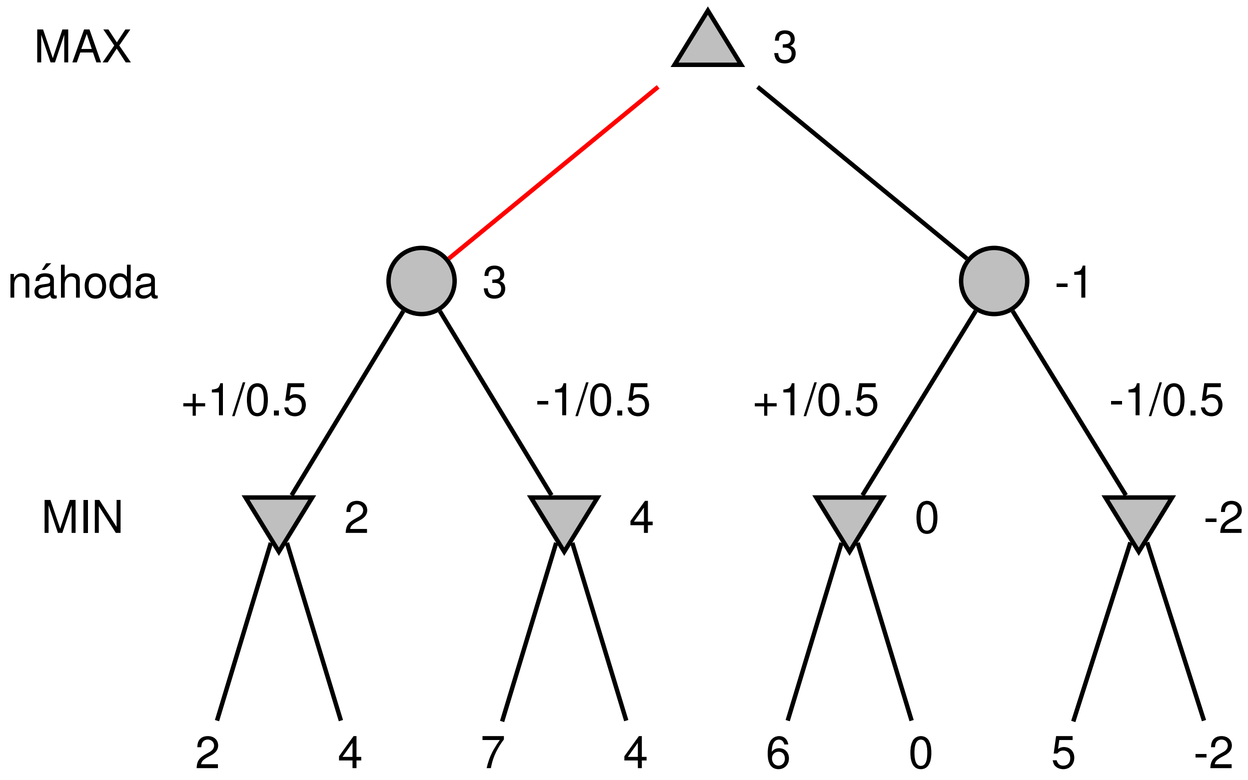
Nedeterministické hry

náhoda $\longleftarrow$ hod kostkou, hod mincí, míchání karet

Nedeterministické hry

náhoda ⟵ hod kostkou, hod mincí, míchání karet

příklad – 1 tah s házení mincí:



Algoritmus Minimax pro nedeterministické hry

expect_minimax … počítá perfektní hru s přihlédnutím k náhodě

rozdíl je pouze v započítání uzlů *náhoda*:

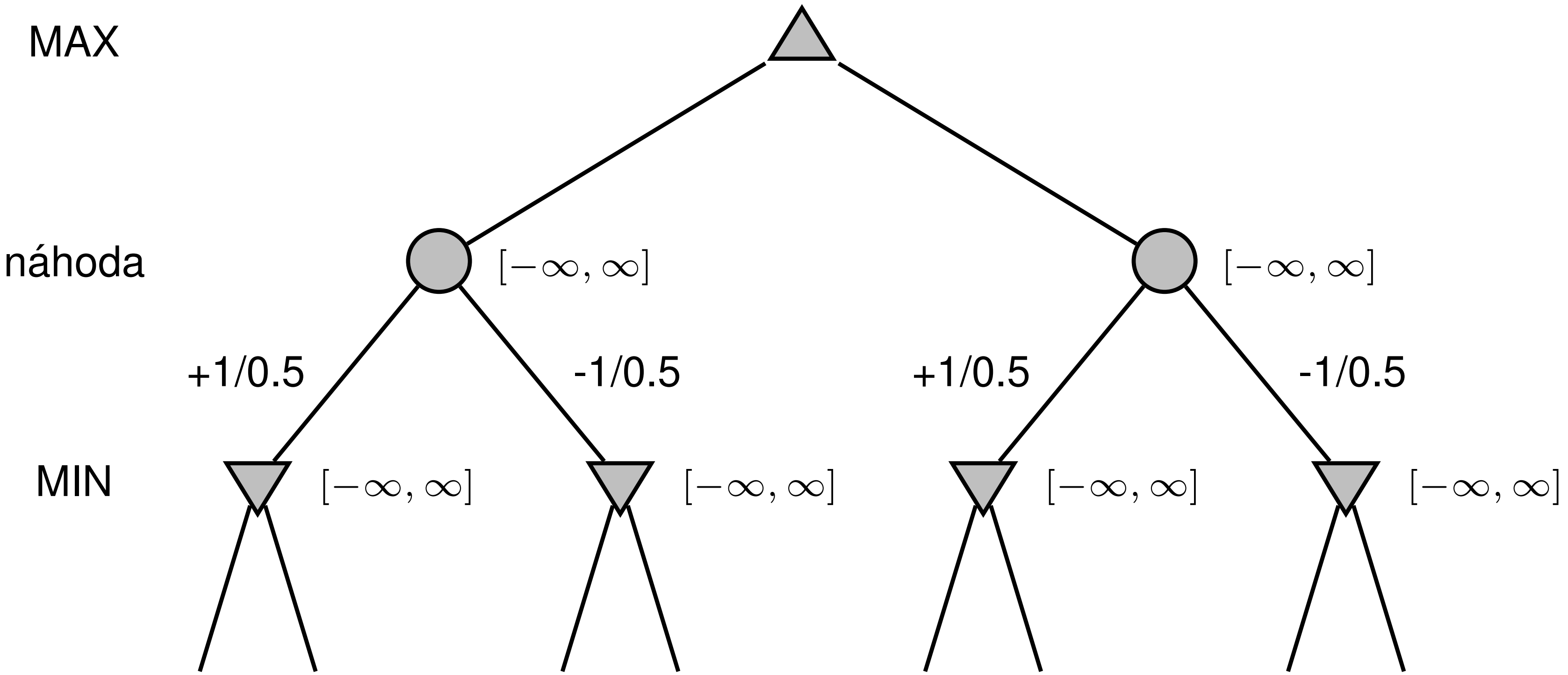
$$\text{expect\_minimax}(n) = \begin{cases} \text{utility}(n) & \text{pro koncový stav } n \\ \max_{s \in \text{moves}(n)} \text{expect\_minimax}(s) & \text{pro MAX uzel } n \\ \min_{s \in \text{moves}(n)} \text{expect\_minimax}(s) & \text{pro MIN uzel } n \\ \sum_{s \in \text{moves}(n)} P(s) \cdot \text{expect\_minimax}(s) & \text{pro uzel náhody } n \end{cases}$$

Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání

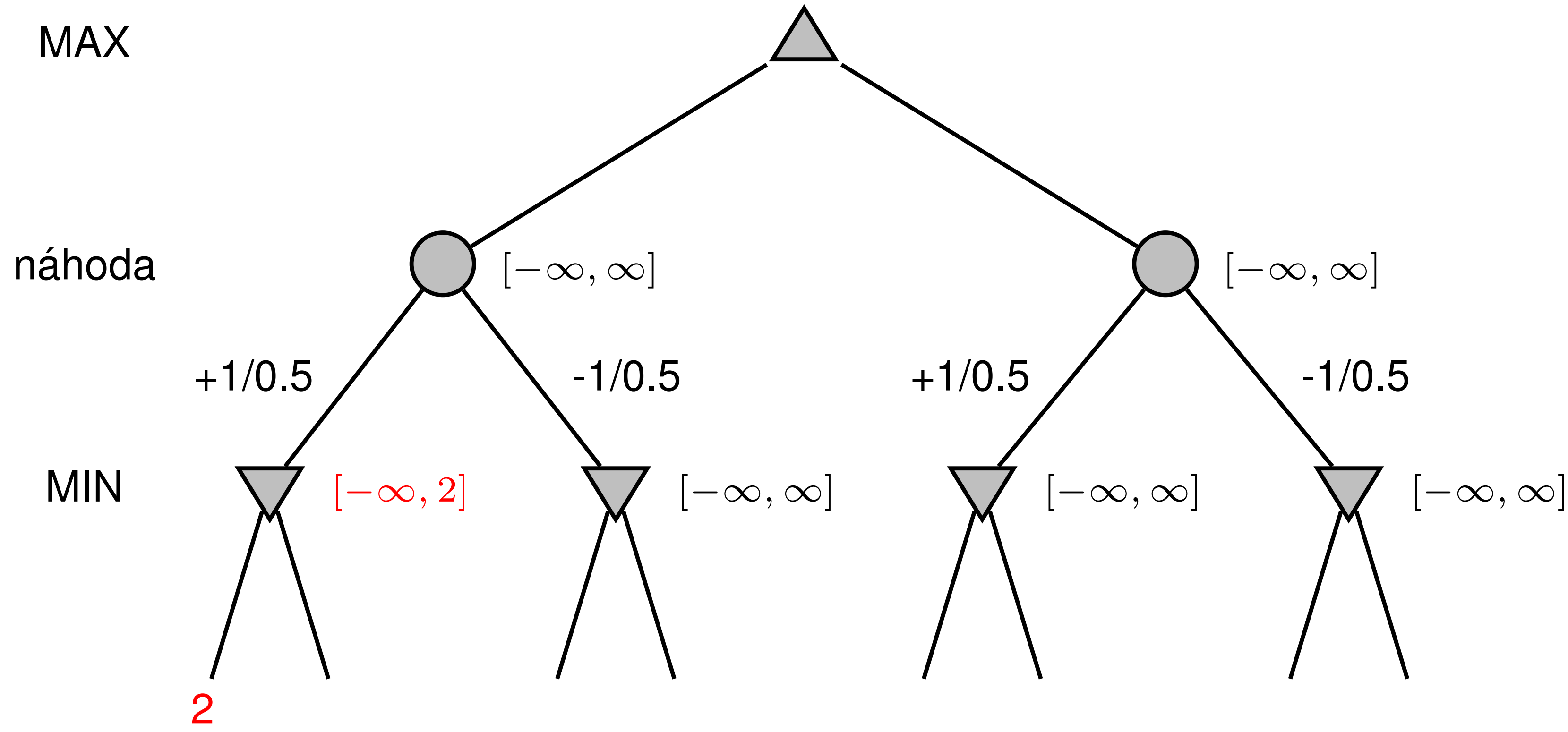
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



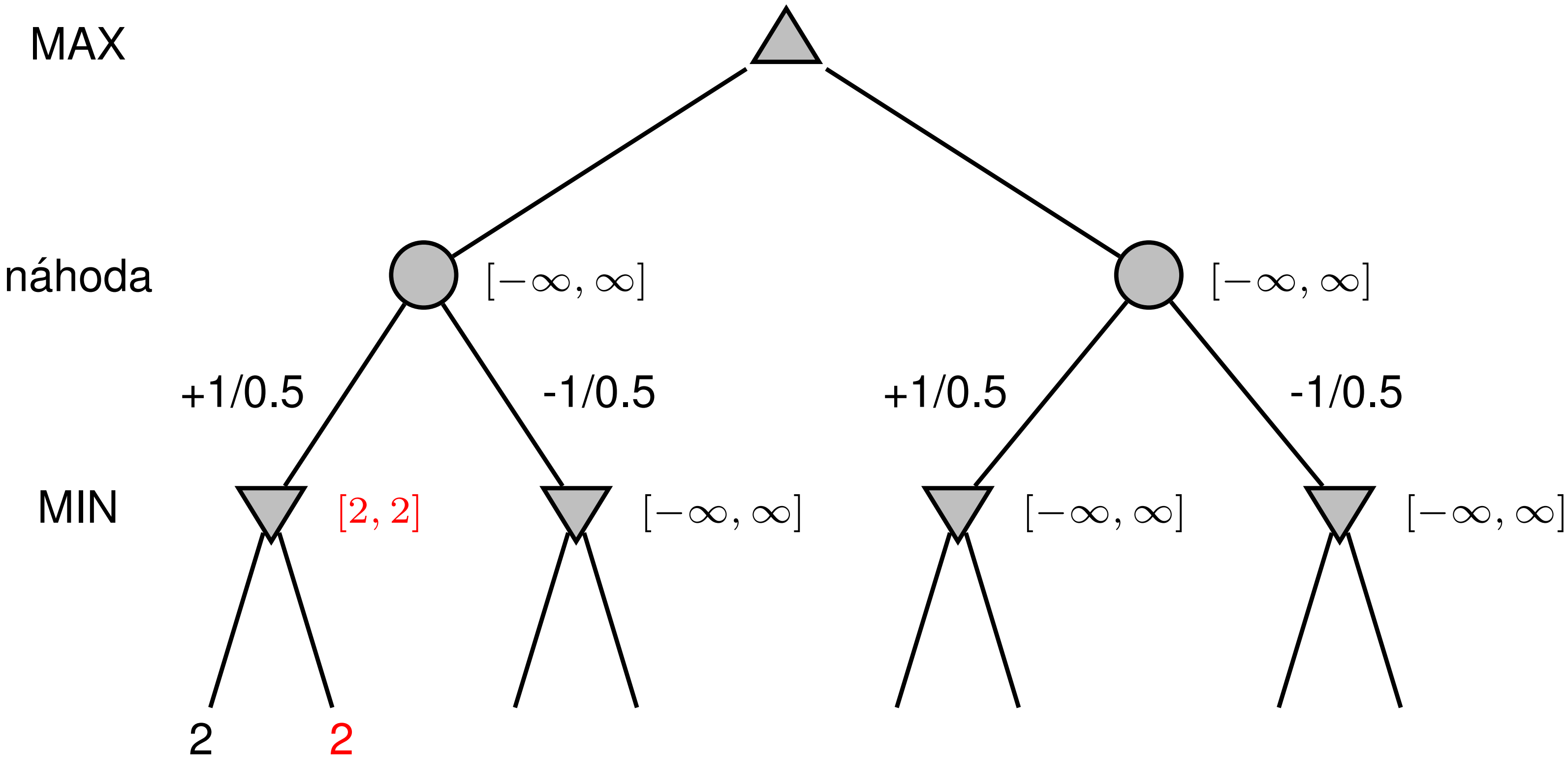
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



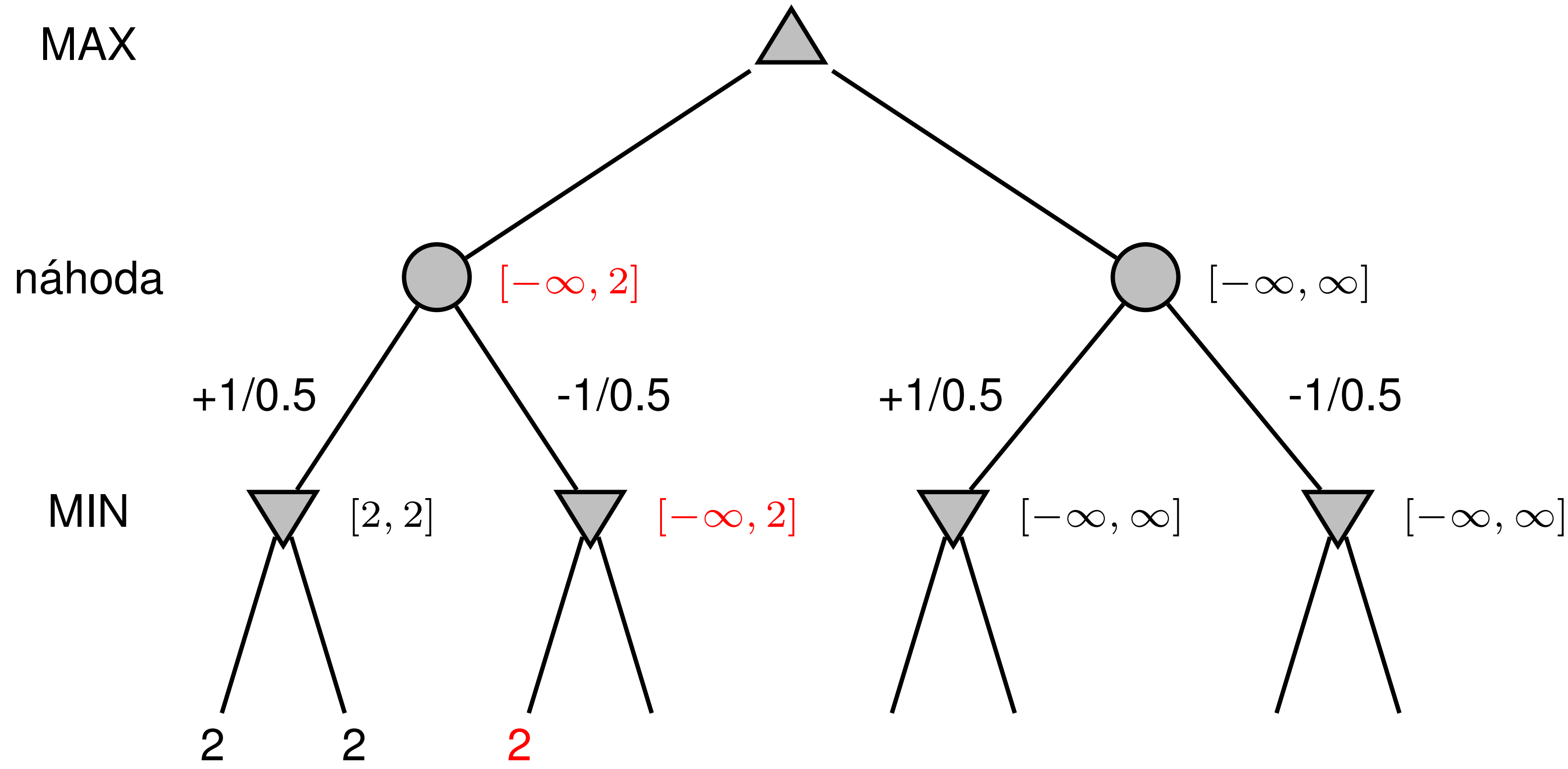
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



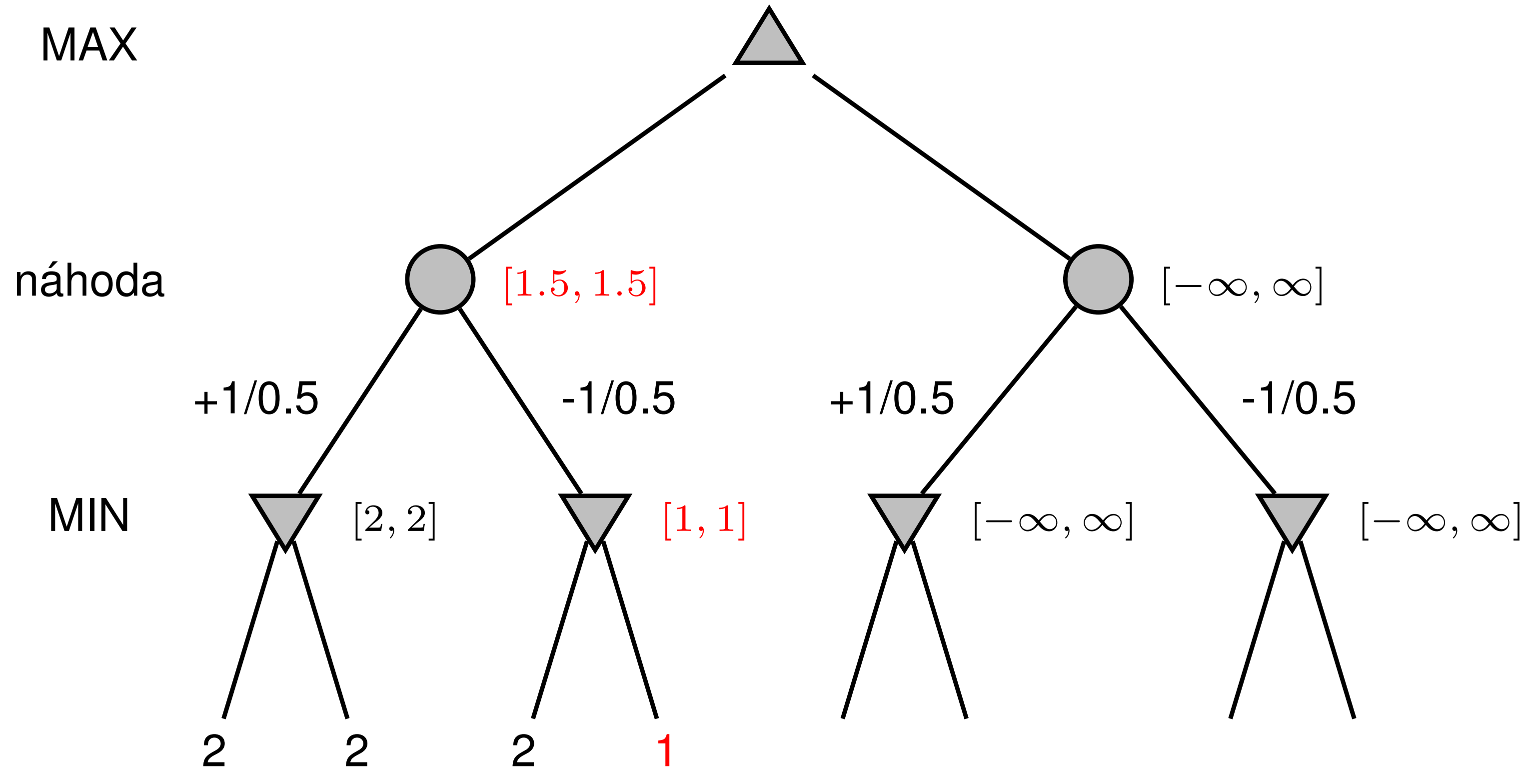
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



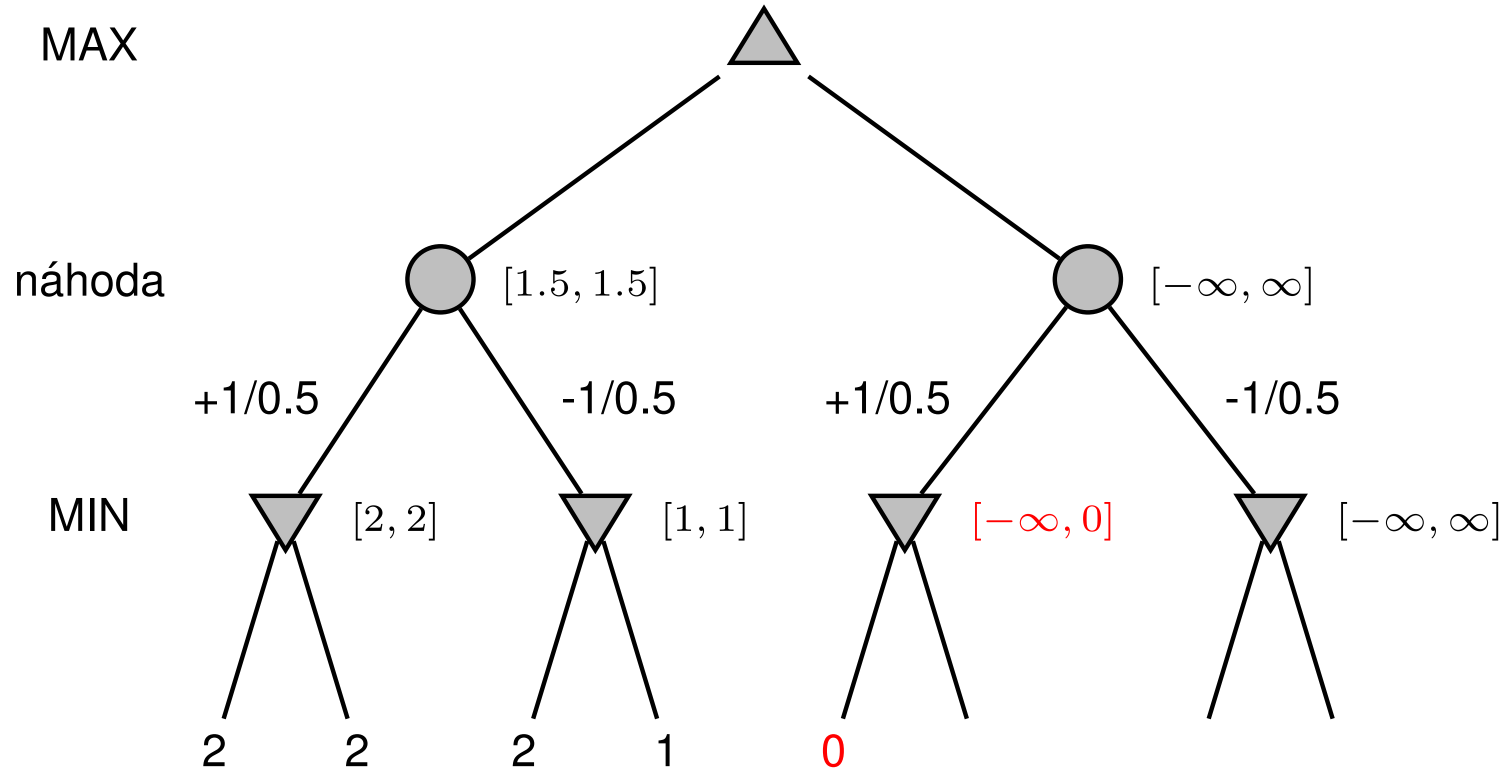
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



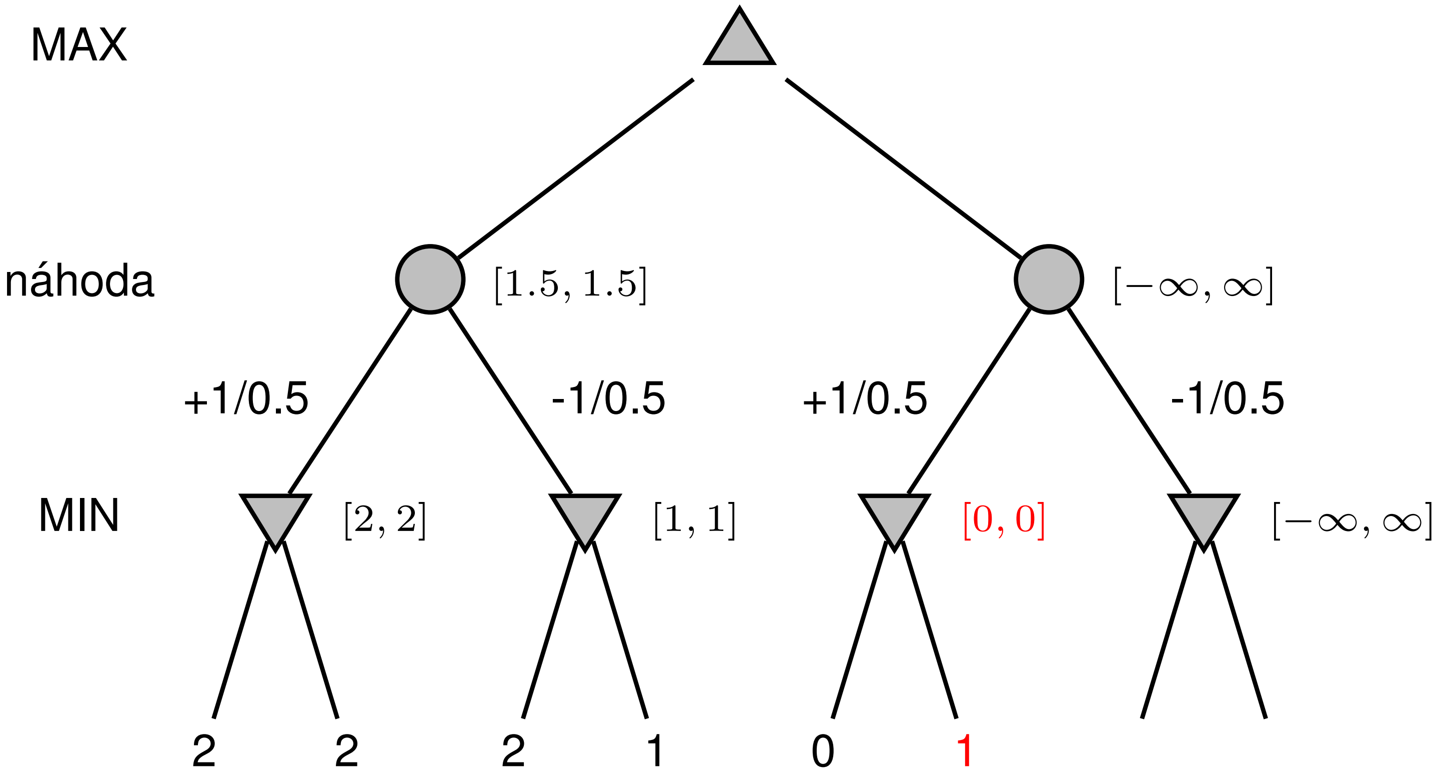
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



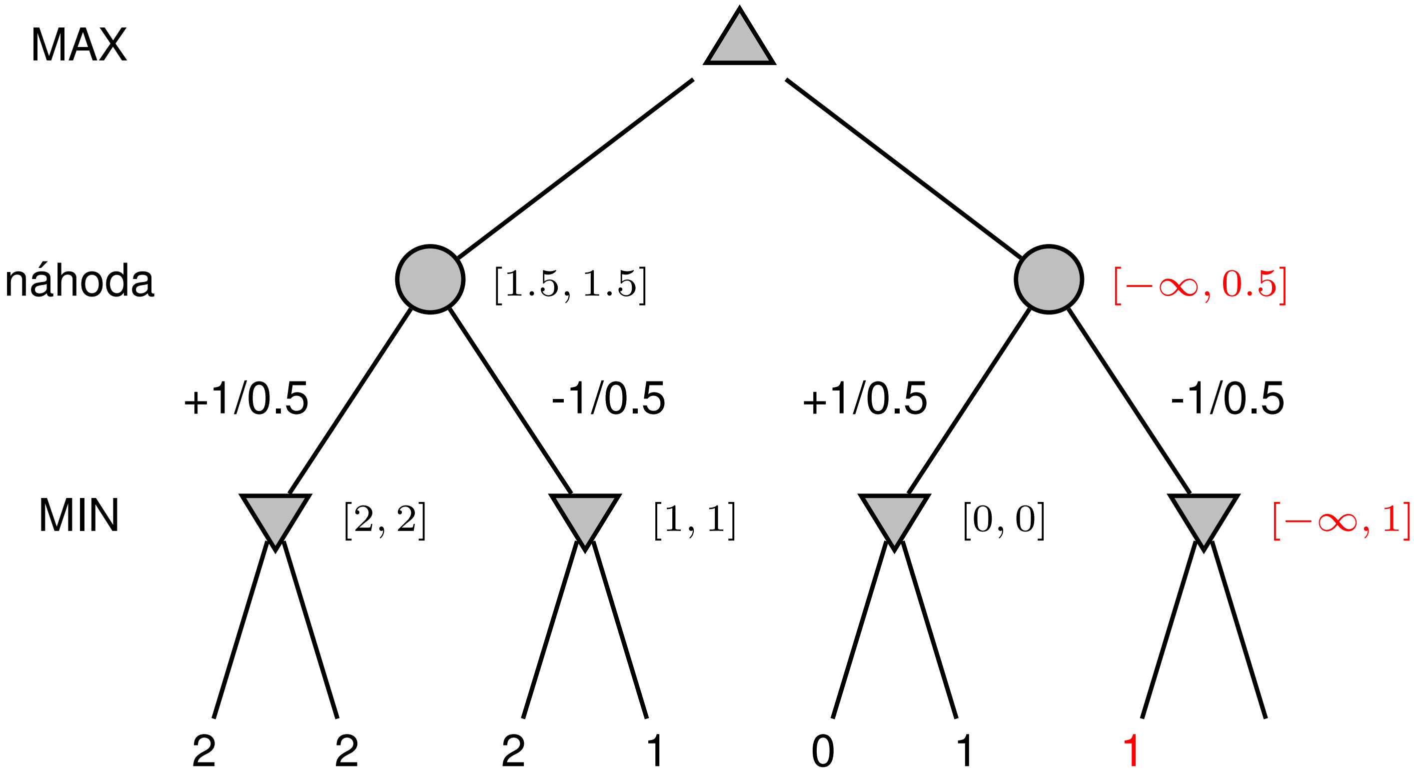
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



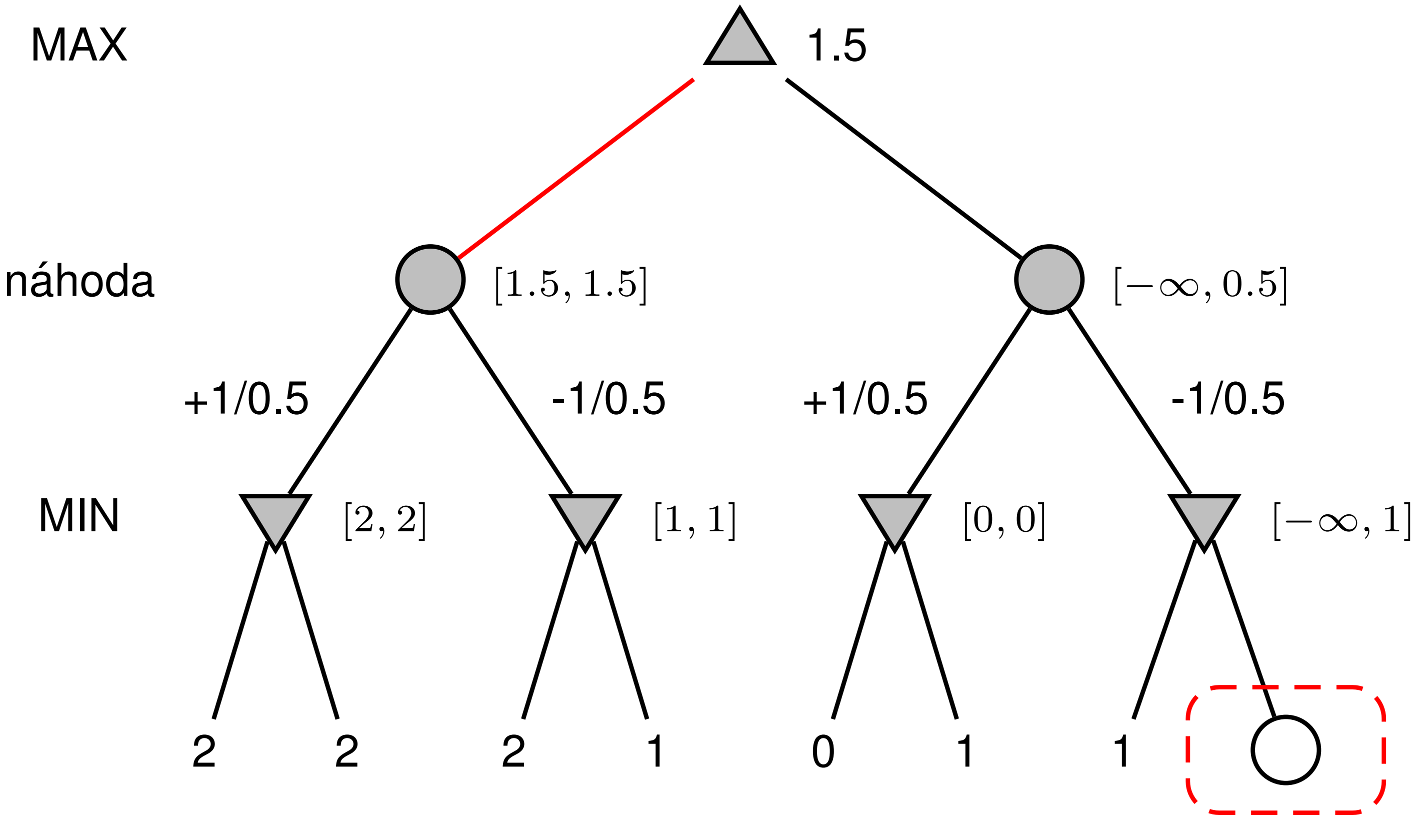
PROŘEZÁVÁNÍ V NEDETERMINISTICKÝCH HRÁCH

je možné použít upravené Alfa-Beta prořezávání



Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání

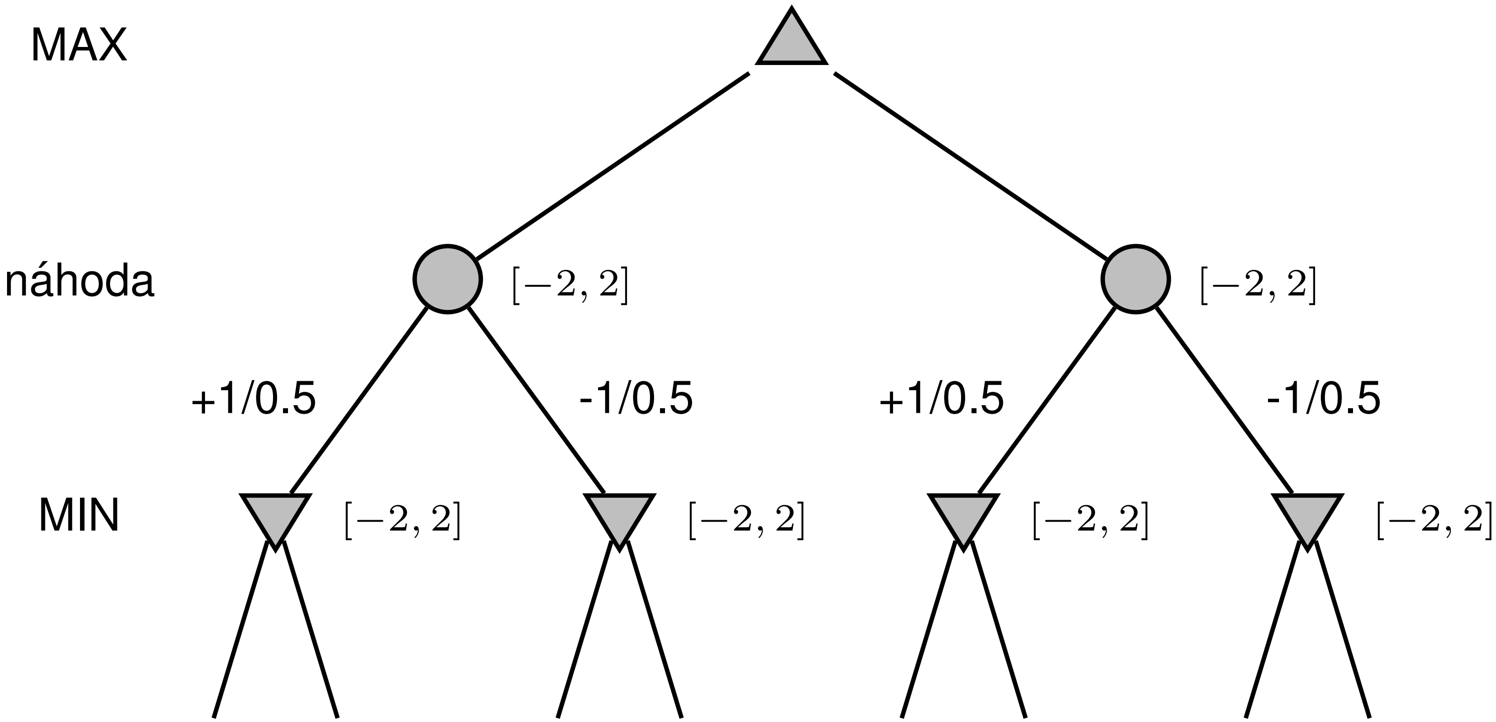


Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší

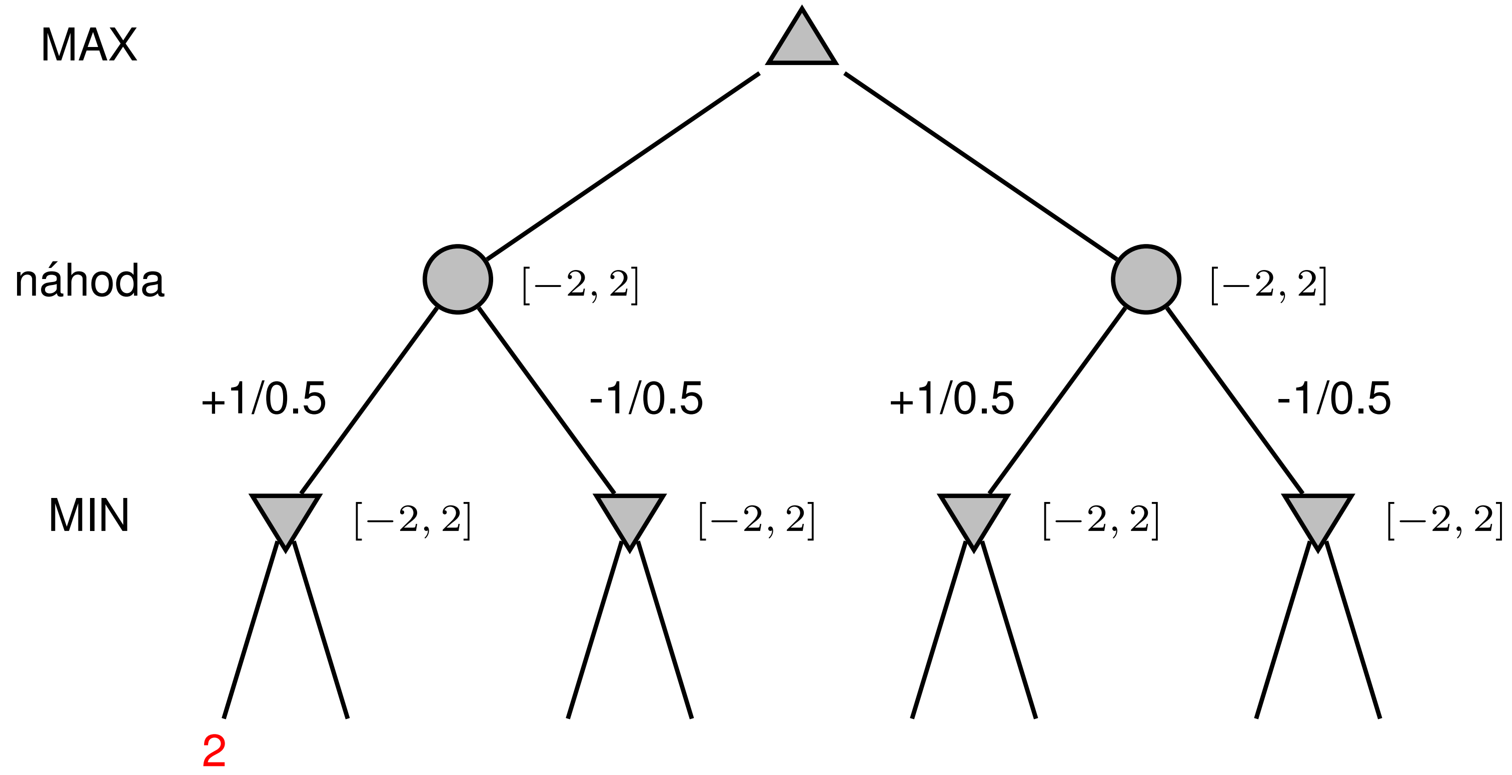
Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



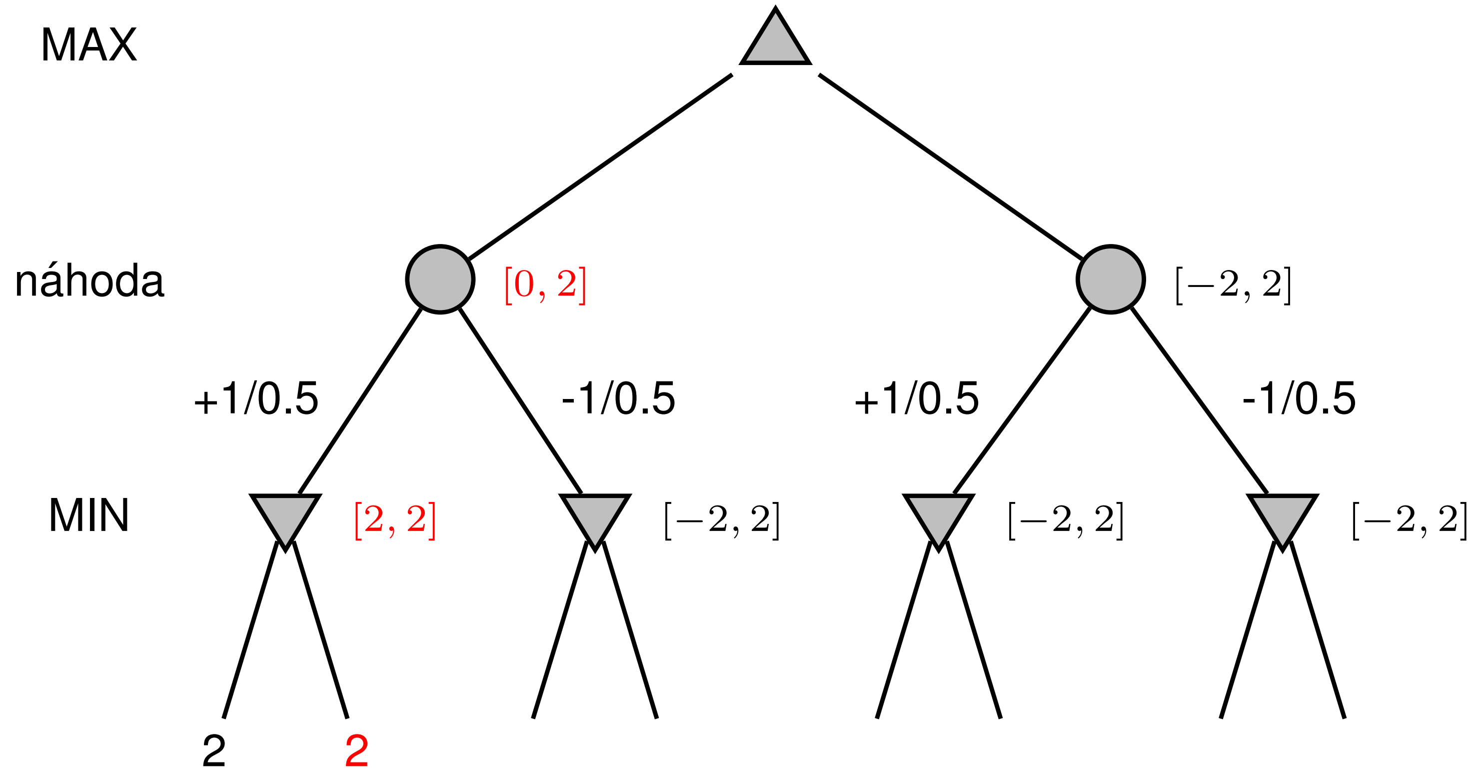
Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



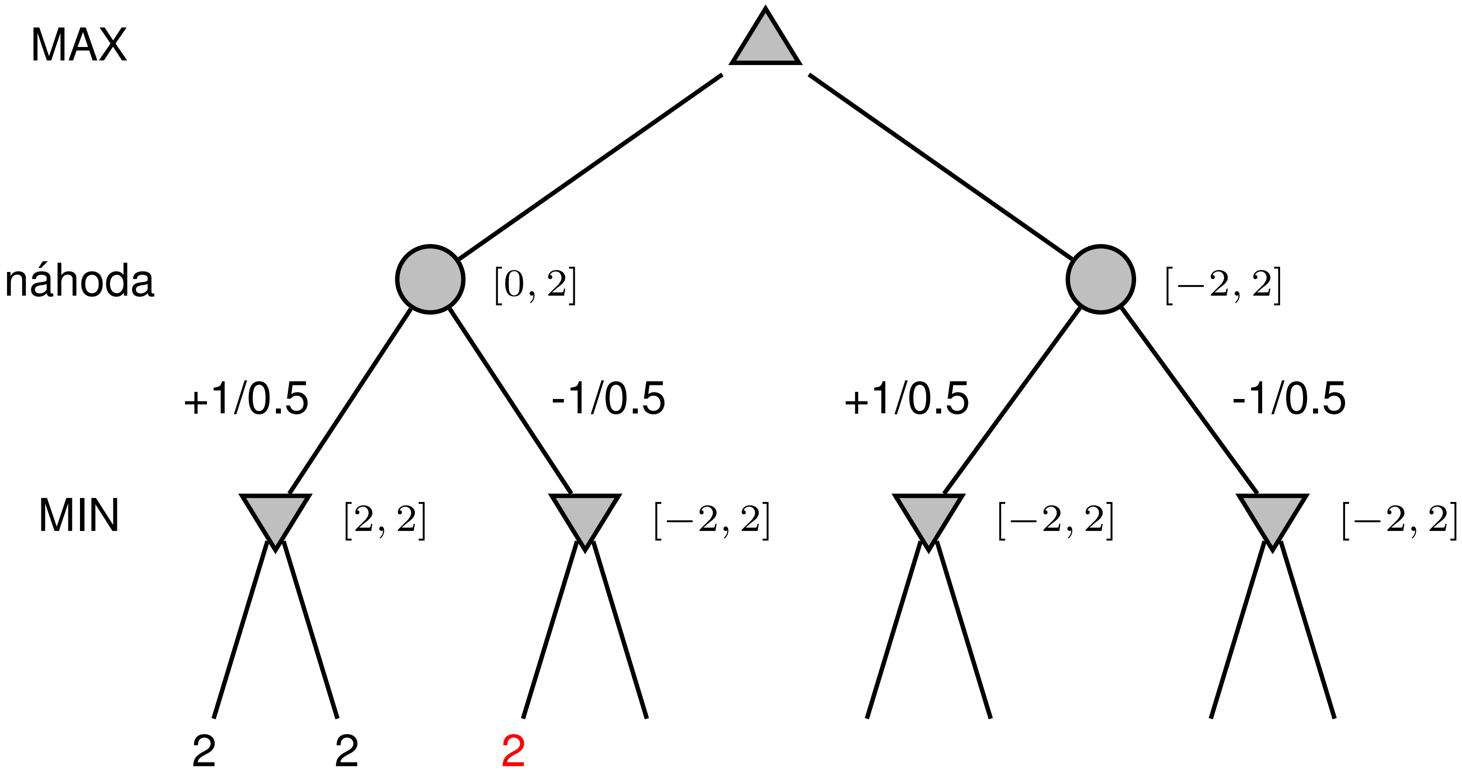
PROŘEZÁVÁNÍ V NEDETERMINISTICKÝCH HRÁCH pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



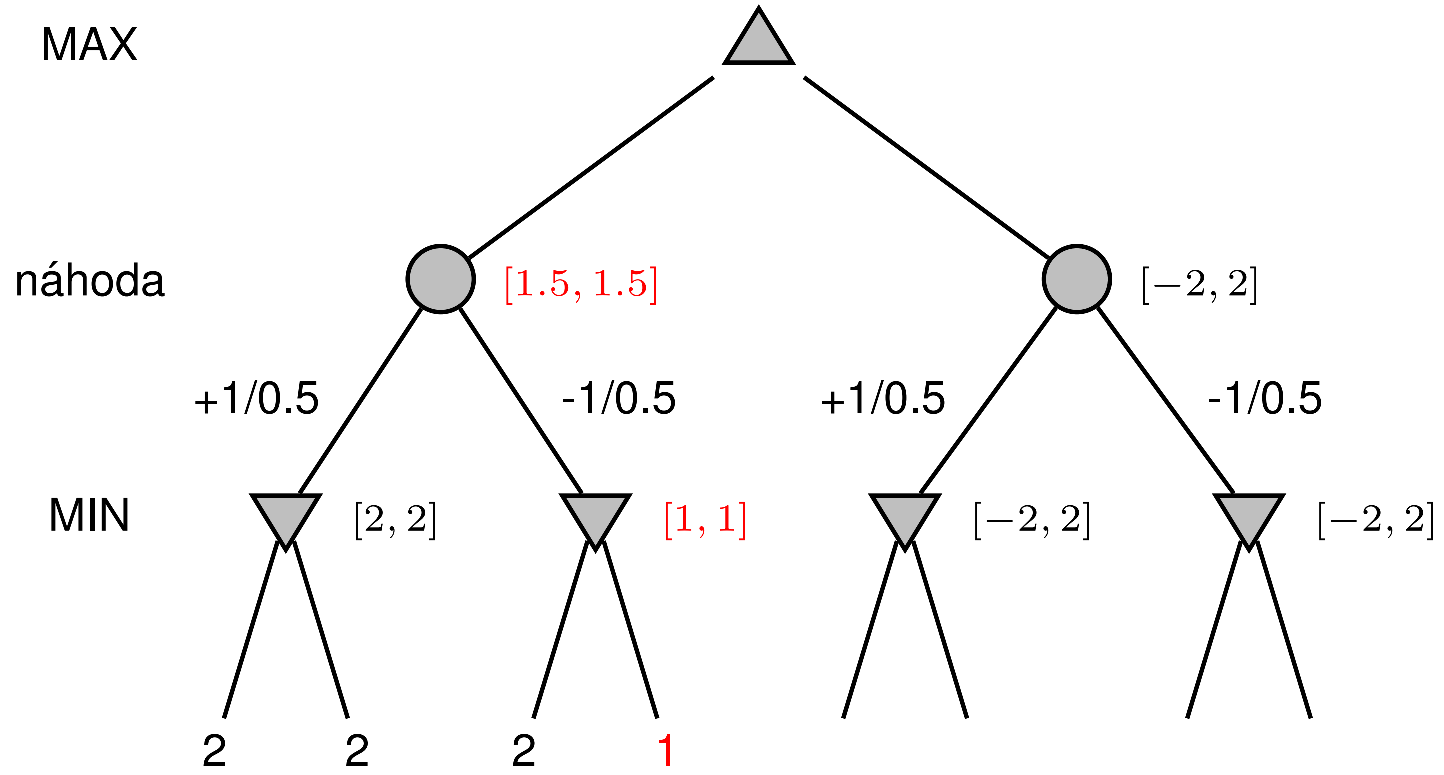
Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



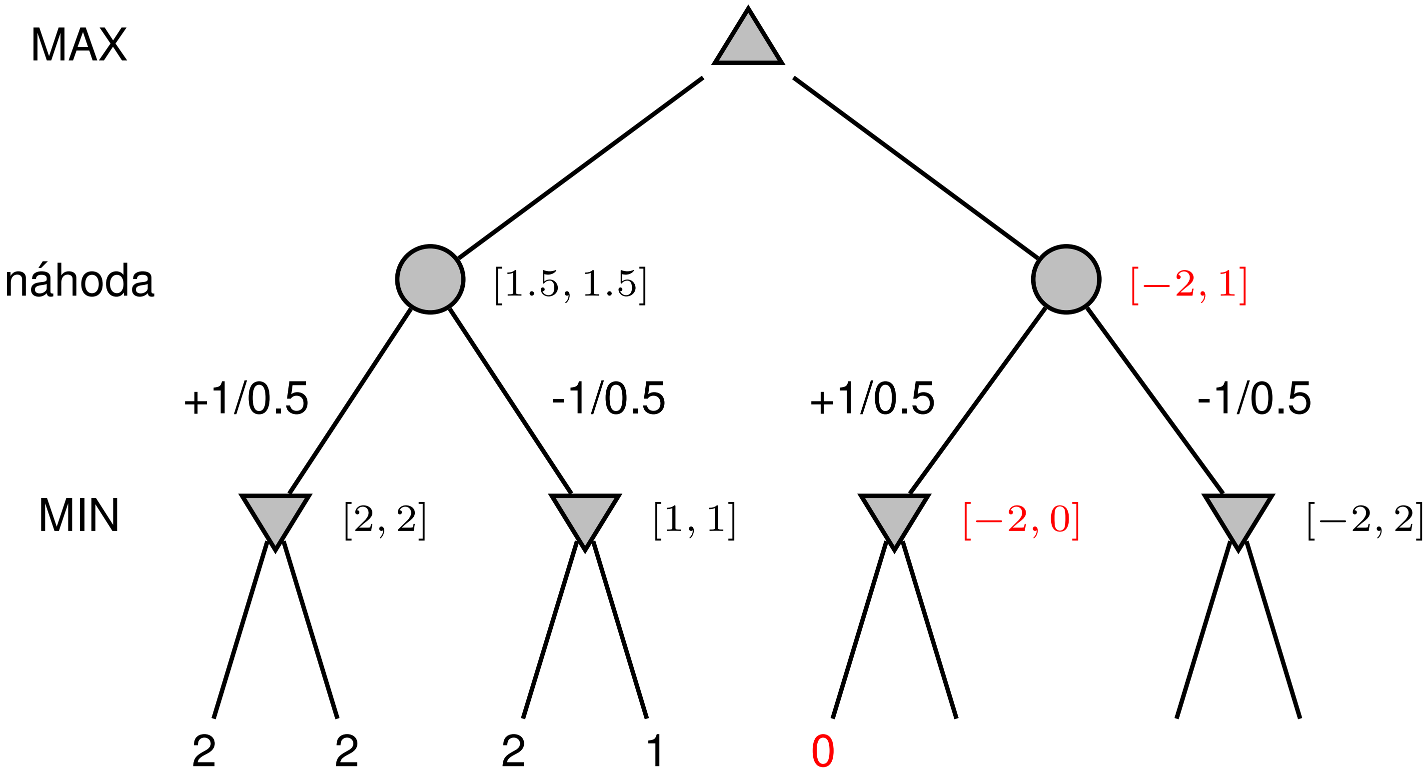
Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



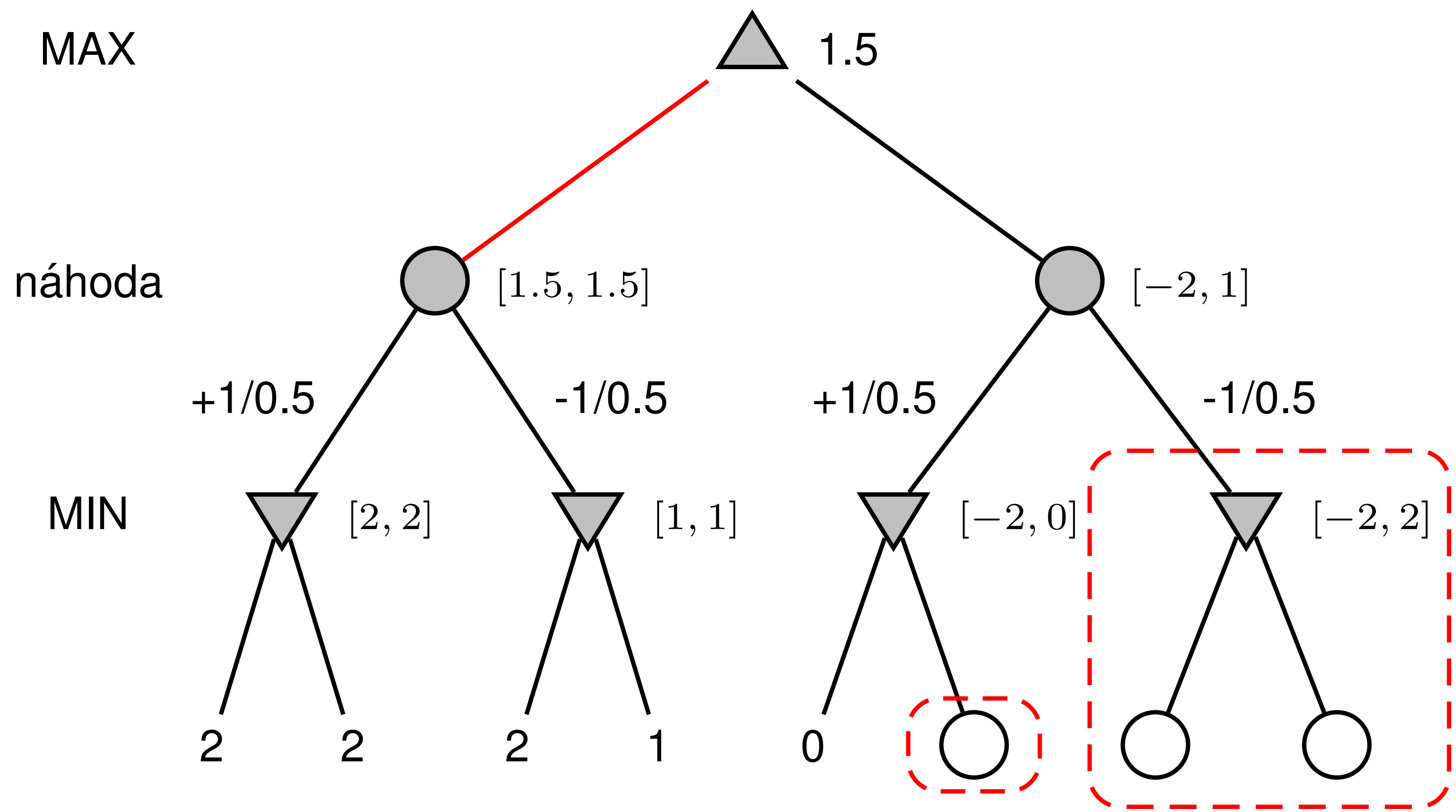
Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



Prořezávání v nedeterministických hrách pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\longrightarrow$ ořezávání je větší



Nedeterministické hry v praxi

- hody kostkou zvyšují $b \longrightarrow$ se dvěma kostkami 21 možných výsledků
- backgammon – 20 legálních tahů:

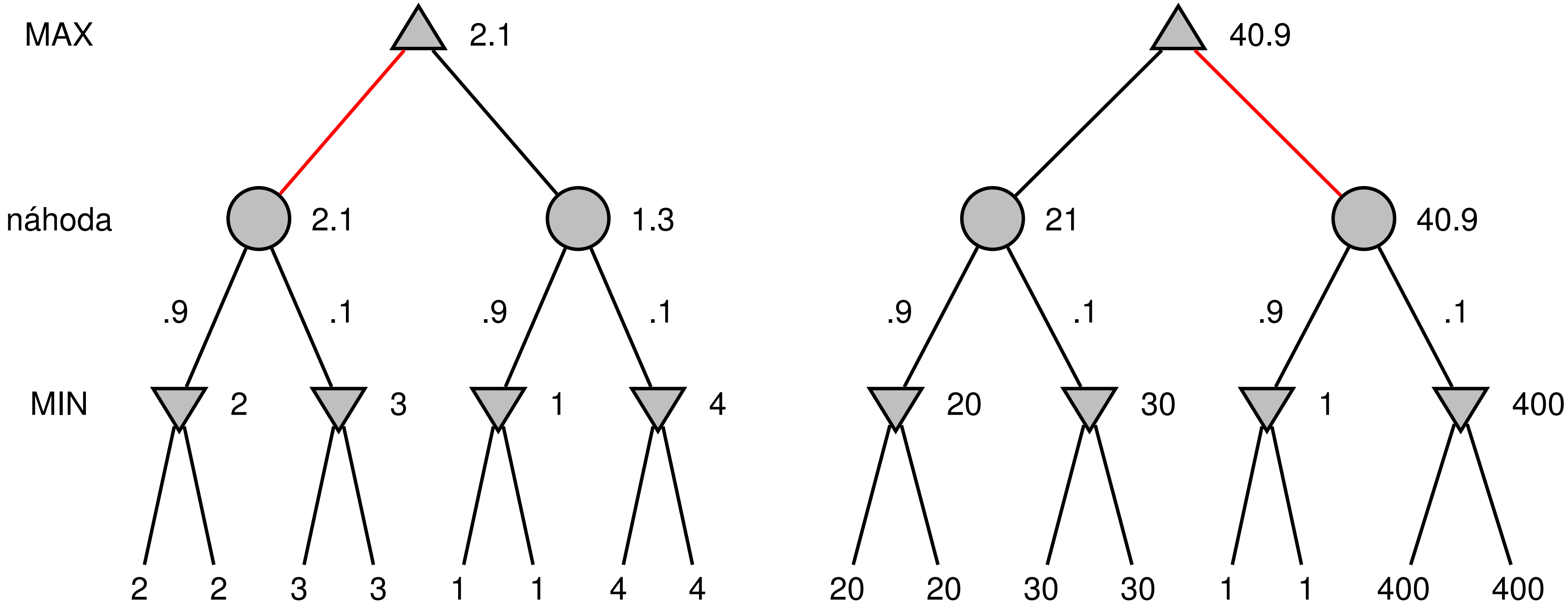
$$\text{hloubka } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

- jak se zvyšuje hloubka $\longrightarrow$ pravděpodobnost dosažení zvoleného uzlu klesá

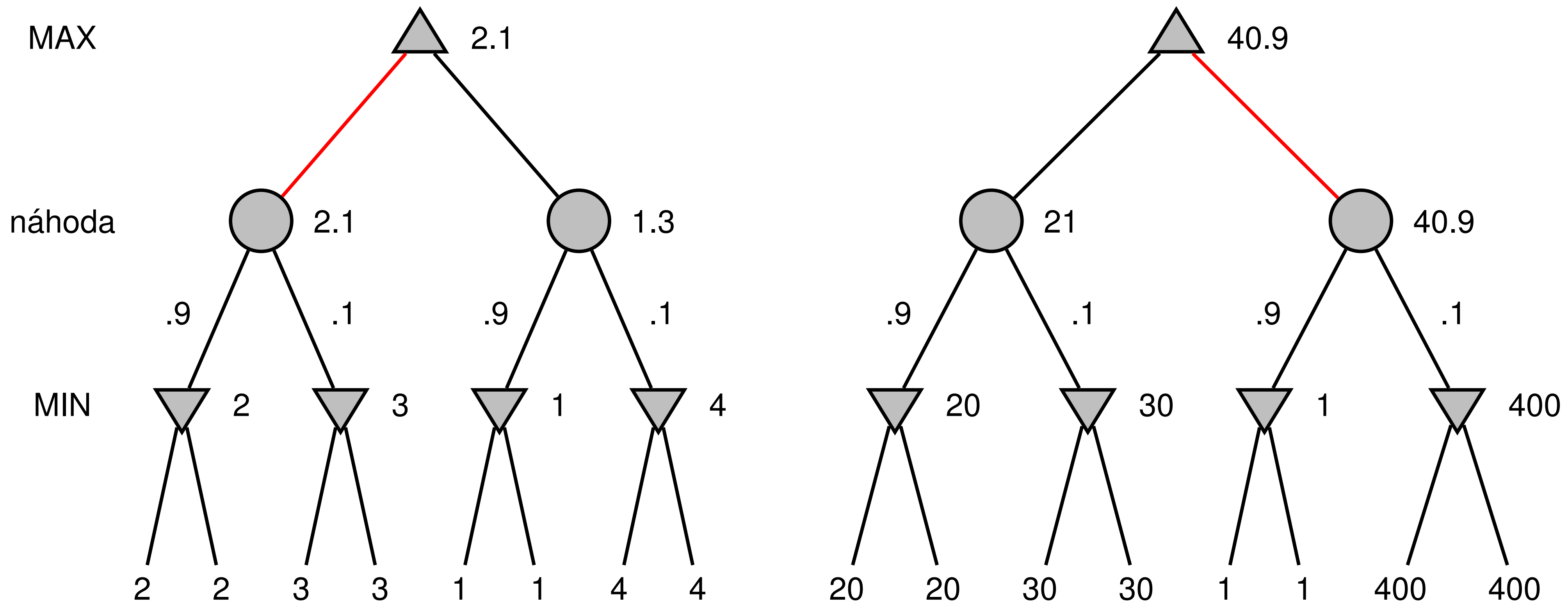
  $\Rightarrow$ význam prohledávání se snižuje
- alfa-beta prořezávání je mnohem méně efektivní
- program *TDGammon* používá prohledávání do hloubky 2 + velice dobrou $Eval$ funkci

  $\approx$ dosahuje úrovně světového šampionátu

Odchylka v ohodnocení nedeterministických her



Odchylka v ohodnocení nedeterministických her



chování je zachováno pouze pro pozitivní lineární transformaci fuknce $Eval$

$Eval$ u nedeterministických her by tedy měla proporcionálně odpovídat očekávanému výnosu

Hry s nepřesnými znalostmi

- např. karetní hry $\longrightarrow$ neznáme počáteční namíchání karet oponenta
- obvykle můžeme spočítat pravděpodobnost každého možného rozdání
- zjednodušeně – jako jeden velký hod kostkou na začátku
- prohledáváme ovšem ne *reálný stavový prostor*, ale domnělý stavový prostor
- program *GIB* vyhrál šampionát v roce 2000:
  1. generuje 100 rozdání karet konzistentních s daným podáním
  2. vybírá akci, která je v průměru nejlepší