Proiectarea Algoritmilor

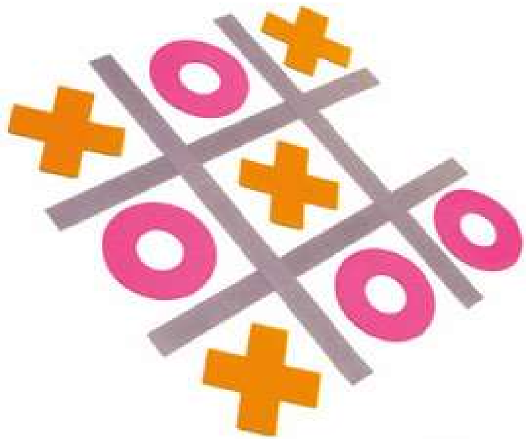
Curs 12 – Algoritmi pentru jocuri

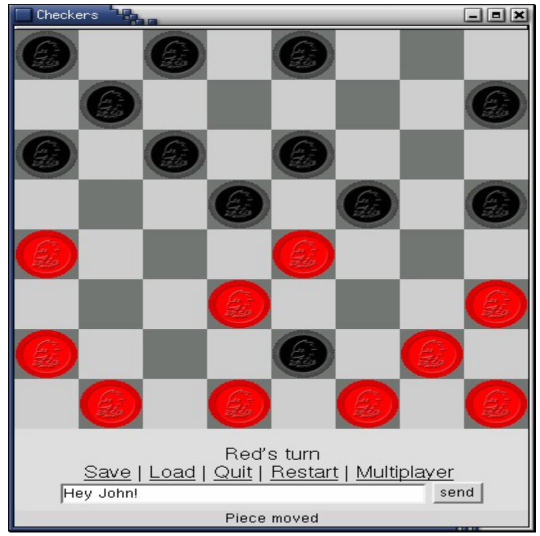
Minimax, α-β

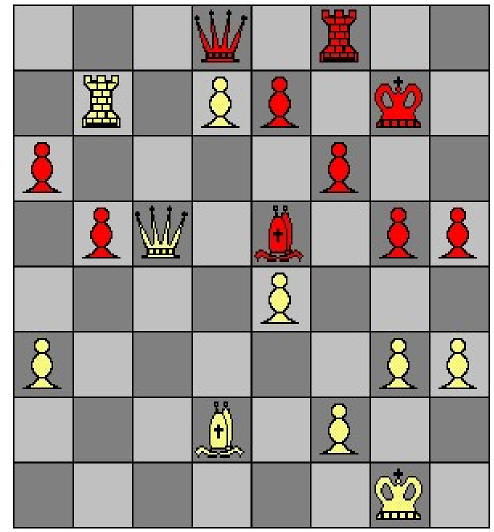
Bibliografie

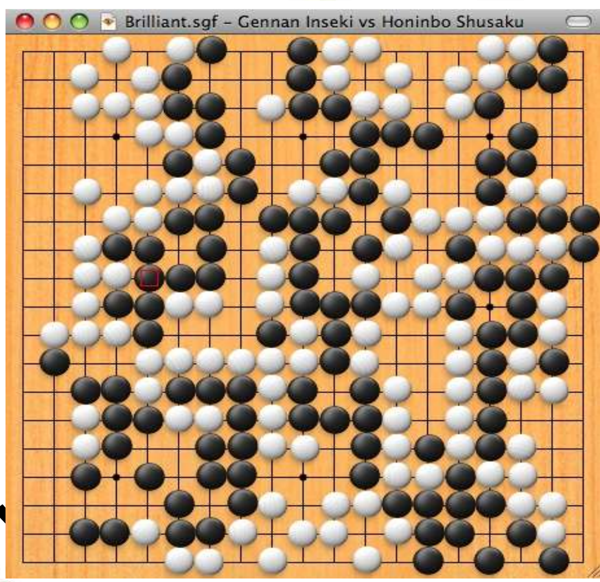
- Giumale – Introducere in Analiza Algoritmilor cap 7.6
- http://www.dwheeler.com/chess-openings/#Sicilian%20Defense
- http://mouserunner.com/MozllaTicTacToe/Mozilla_Tic_Tac_Toe.htm
- http://www.emunix.emich.edu/~evett/AI/AlphaBeta_movie/index_movie.htm

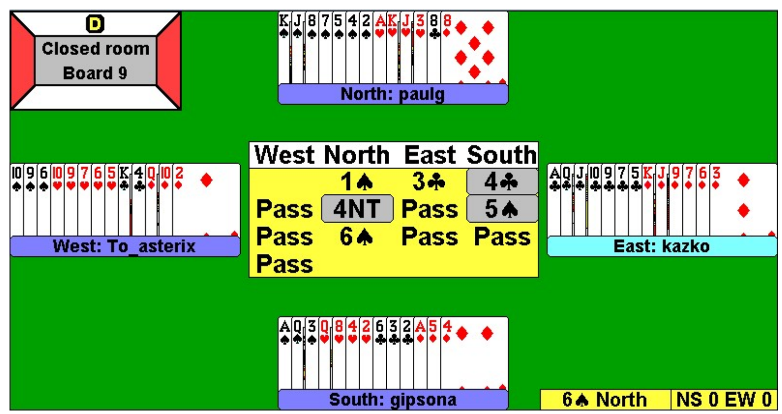
Problema



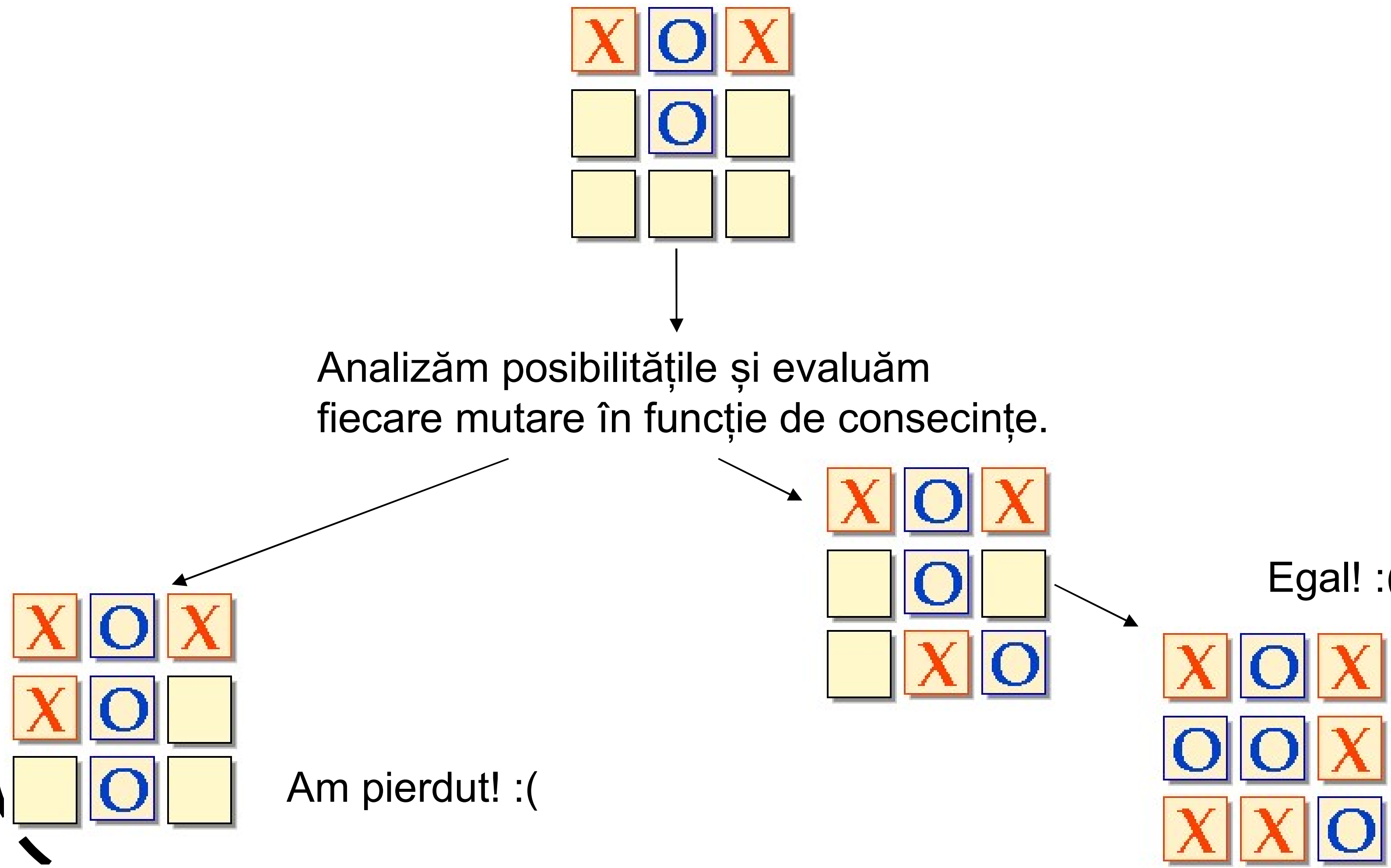

Checkers
Red's turn
Save | Load | Quit | Restart | Multiplayer
Hey John!
send
Piece moved



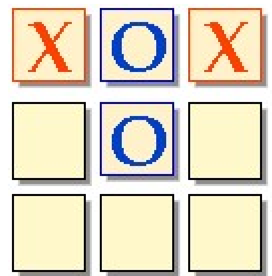

Brilliant.sgf – Gennan Inseki vs Honinbo Shusaku


D
Closed room
Board 9
North: paulg
West North East South
1♠ 3♣ 4♣
Pass 4NT Pass 5♠
Pass 6♠ Pass Pass
Pass
West: To_asterix
East: kazko
South: gipsona
6♠ North
NS 0 EW 0

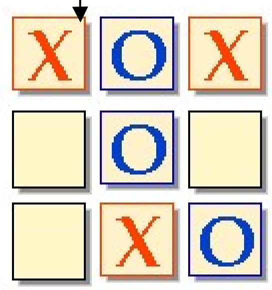
Cum gândim noi?

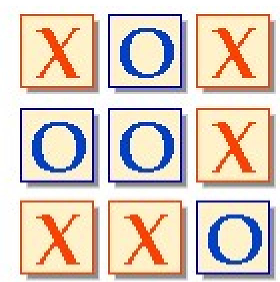


Cum gândim noi?



Ne dăm seama ”instinctiv” că avem o singură opțiune pentru a nu pierde partida și mutăm în consecință!

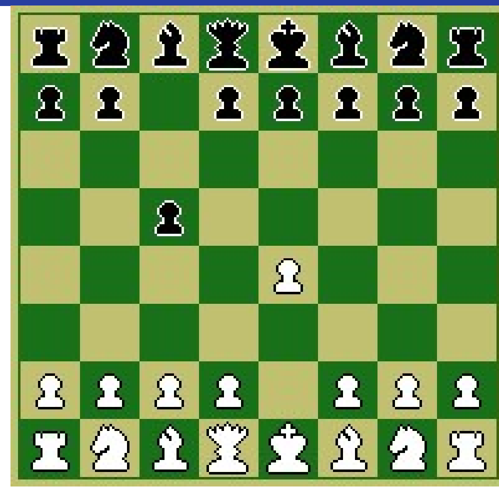




Egal! :(

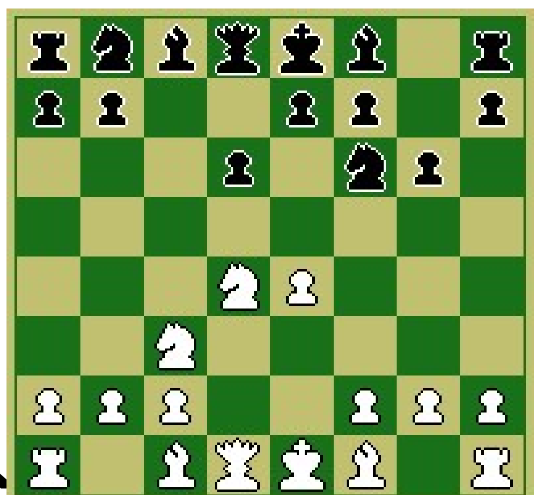
Cum gândim noi?

http://www.dwheeler.com/chess-openings/#Sicilian%20Defense



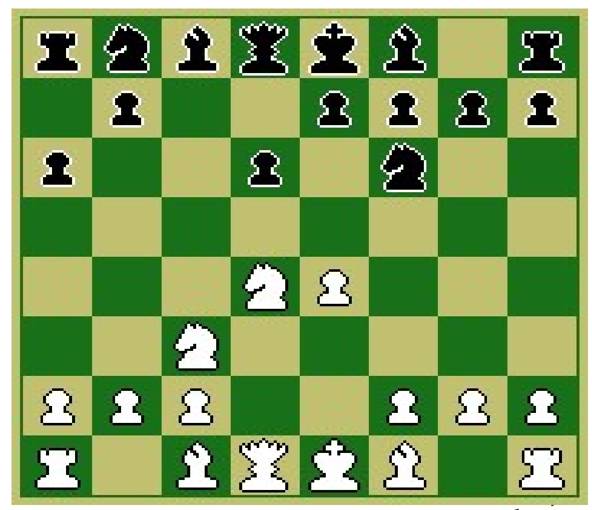
Apărarea siciliană!

Varianta Najdorf



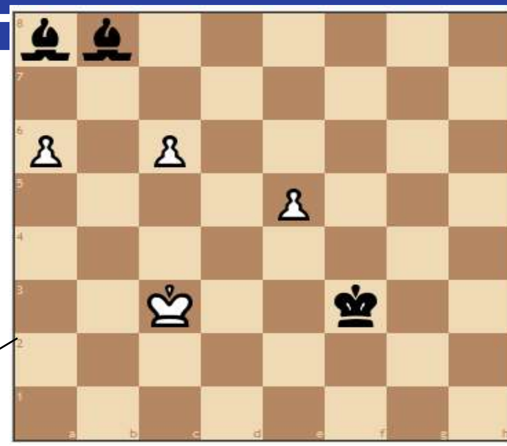
Când avem foarte multe posibilități la dispoziție încercăm să folosim poziții (pattern-uri) cunoscute.

Varianta Dragon

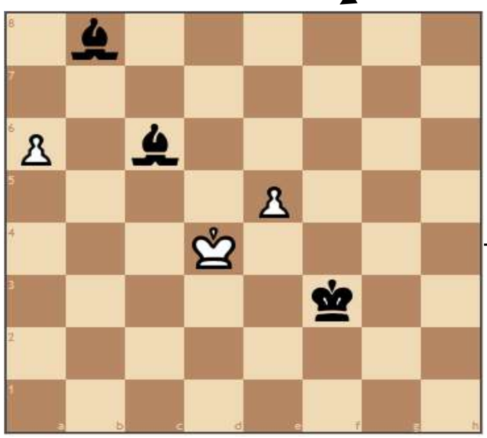


Cum gândim noi?

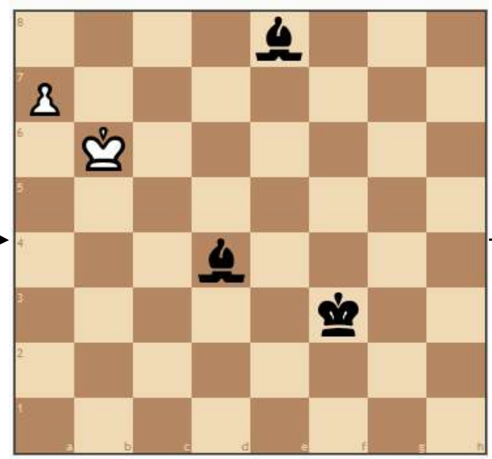
Albul la mutare
-11 posibilități de mutare;
- le putem încerca pe toate să vedem ce se întâmplă.



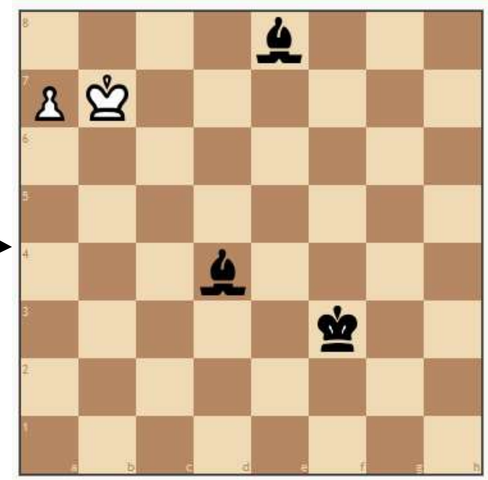
Circa 15.000 de mutări de analizat – ușor pentru calculator. Noi eliminăm mutările ce ni se par fără sens (mai mult de jumătate).



Numărul mutărilor posibile se reduce la 6.



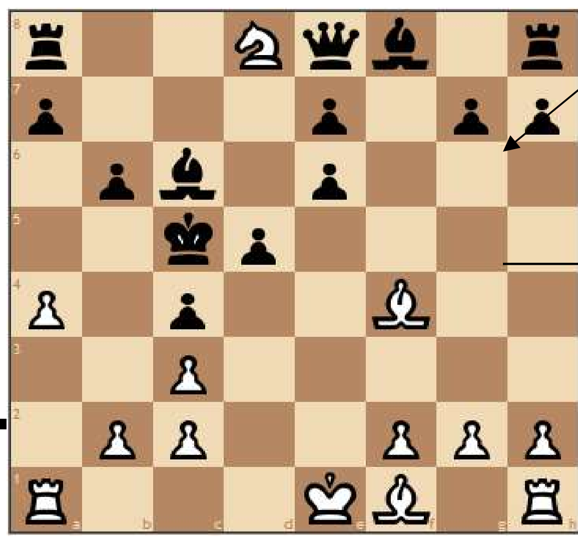
Doar 4 mutări posibile.



Remiză asigurată!

Cum gândim noi?

Cam 35 de mutări posibile



Varianta câștigătoare presupune sacrificarea unei piese valoroase:

Cum gândim noi?

- Evaluăm amenințările:
  - Căutăm mutări care să minimizeze pierderile;
  - Căutăm mutări care să maximizeze câștigul.

- Alegem mutările ce ni se ”par” cele mai bune pe moment:
  - Explorăm în adâncime graful mutărilor;
  - Numărul de niveluri = minim dintre:
    - Terminarea jocului;
    - Obținerea unui avantaj consistent fără pericol aparent de a-l pierde;
    - Nivelul maxim al capacității noastre de calcul.

Abordări posibile pentru calculator

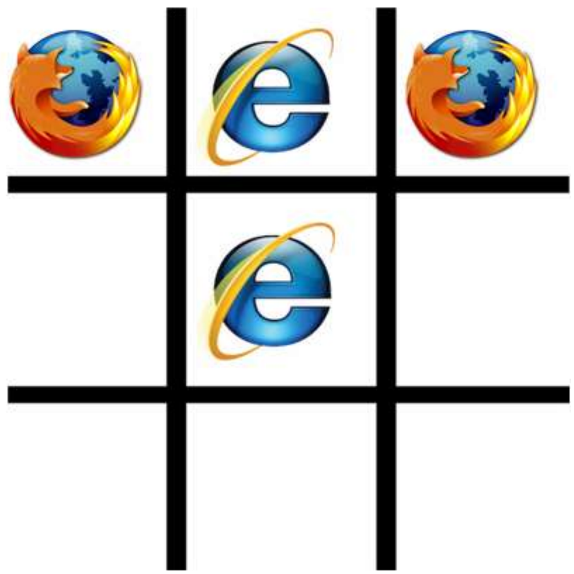
- Șabloane pentru poziții standard.
- Căutare în spațiul de poziții.
- Utilizare euristici pentru evaluarea poziției curente.
- Ne vom concentra asupra căutărilor.

Metoda Minimax

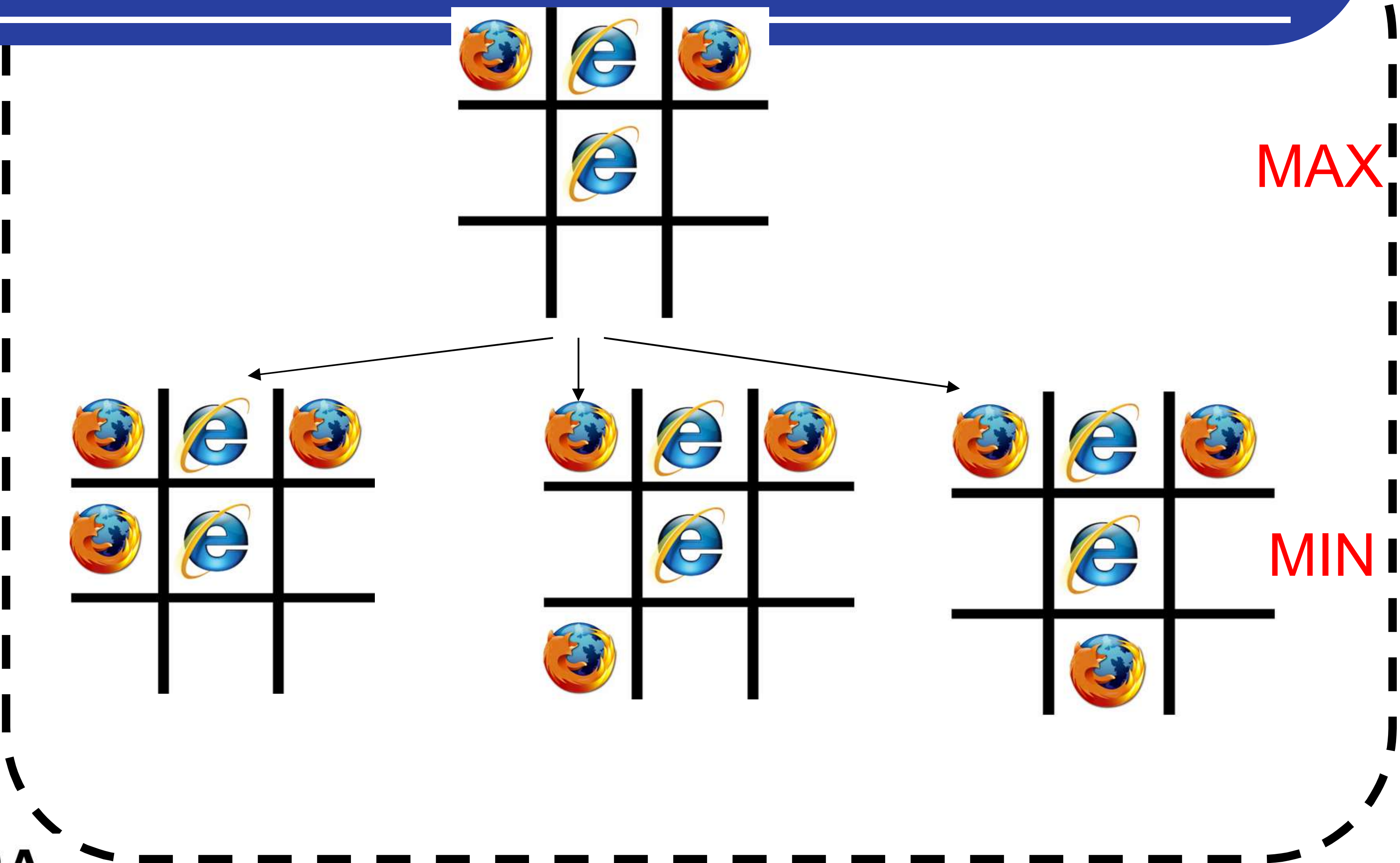
- 2 jucători: Max și Min care mută pe rând (Max mută primul).
- Max urmărește să-și maximizeze câștigul.
- Min urmărește să-și minimizeze pierderea.
- Se construiește un arbore AND-OR:
- Nivelurile impare → mutările jucătorului Max.
- Nivelurile pare → mutările jucătorului Min.
- Frunzele desemnează câștigul/pierderea lui Max.
- Arcele reprezintă mutările propriu-zise.

Exemplu (I)

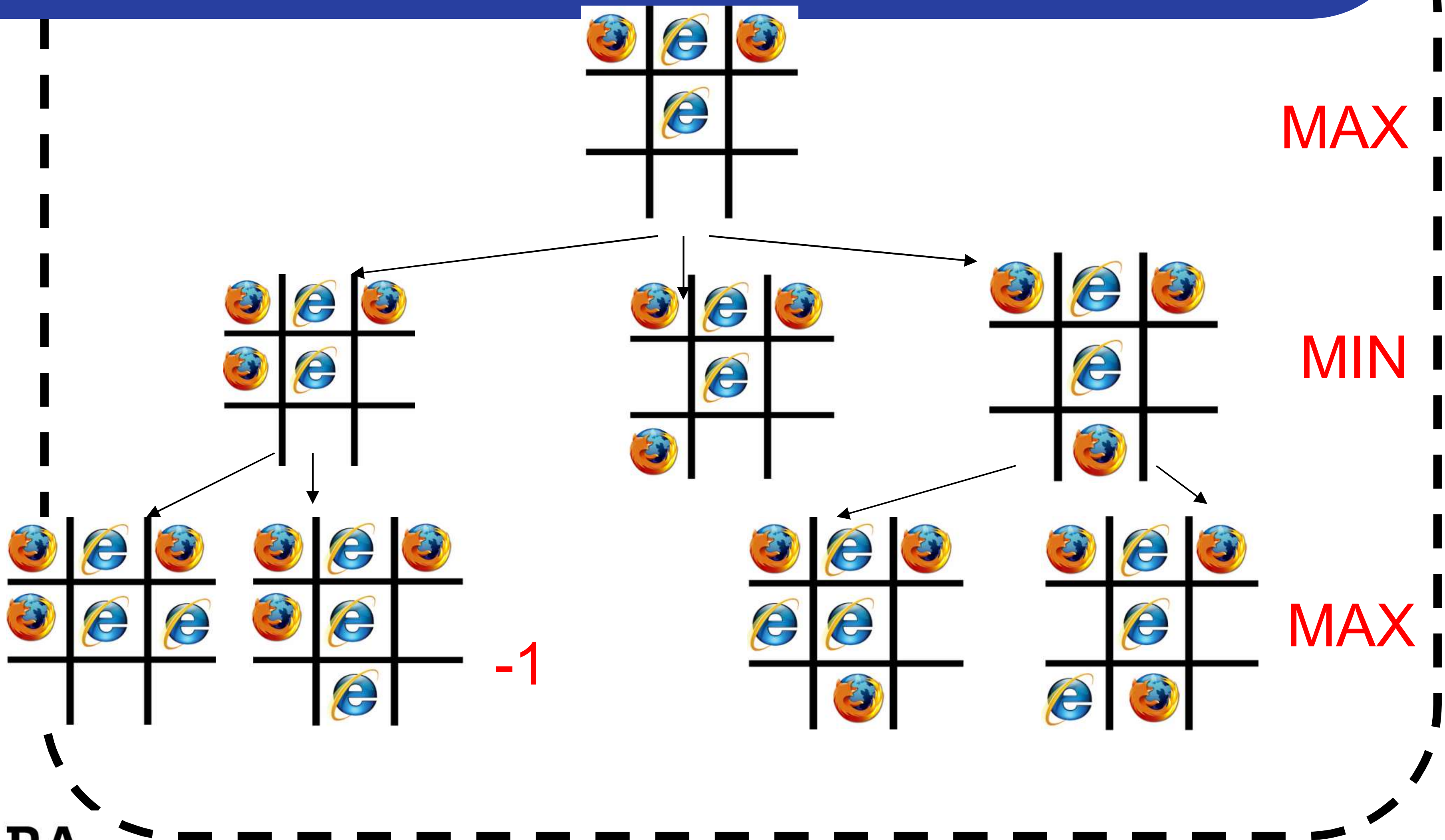
MAX (Firefox) trebuie să mute:



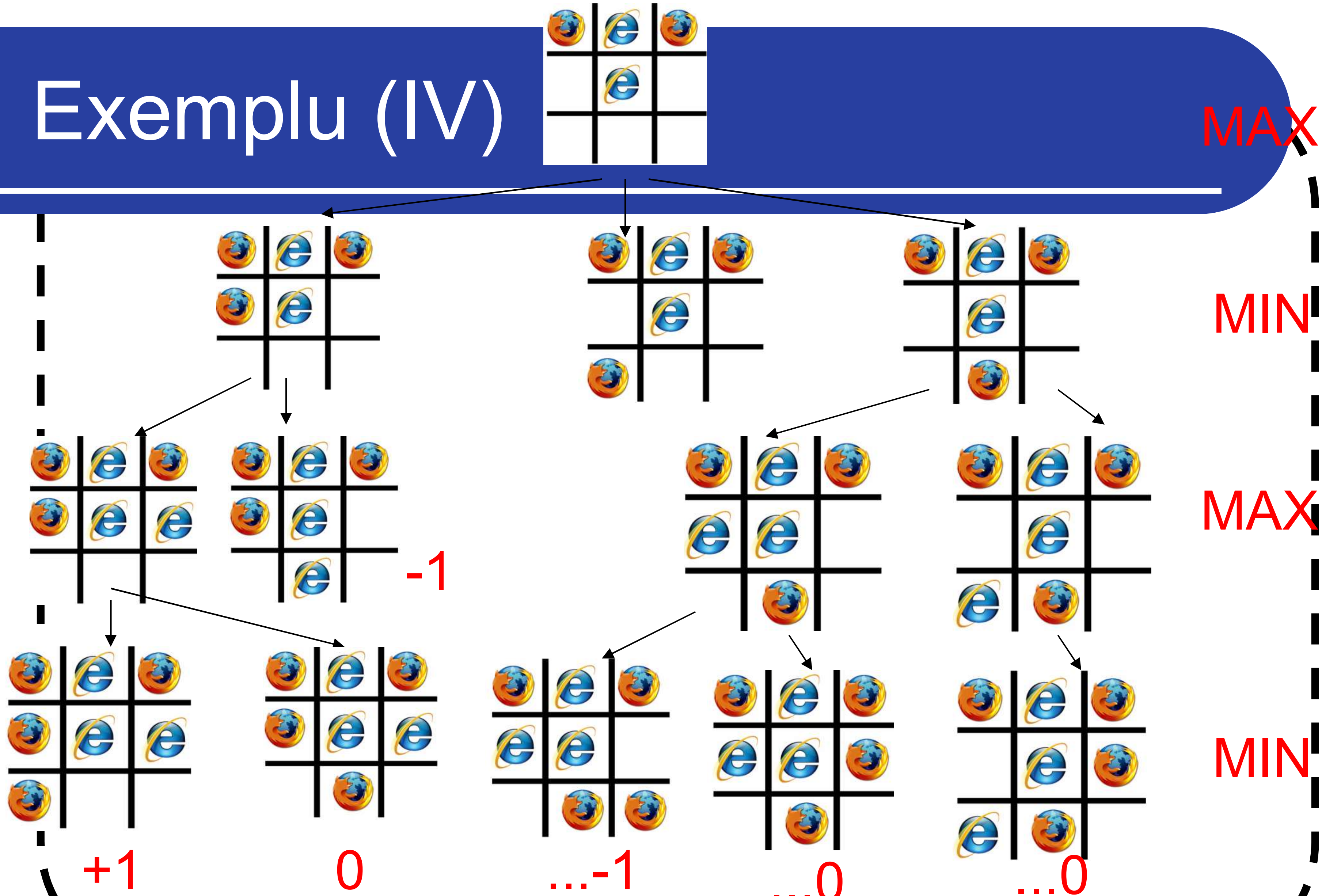
Exemplu (II)



Exemplu (III)



Exemplu (IV)

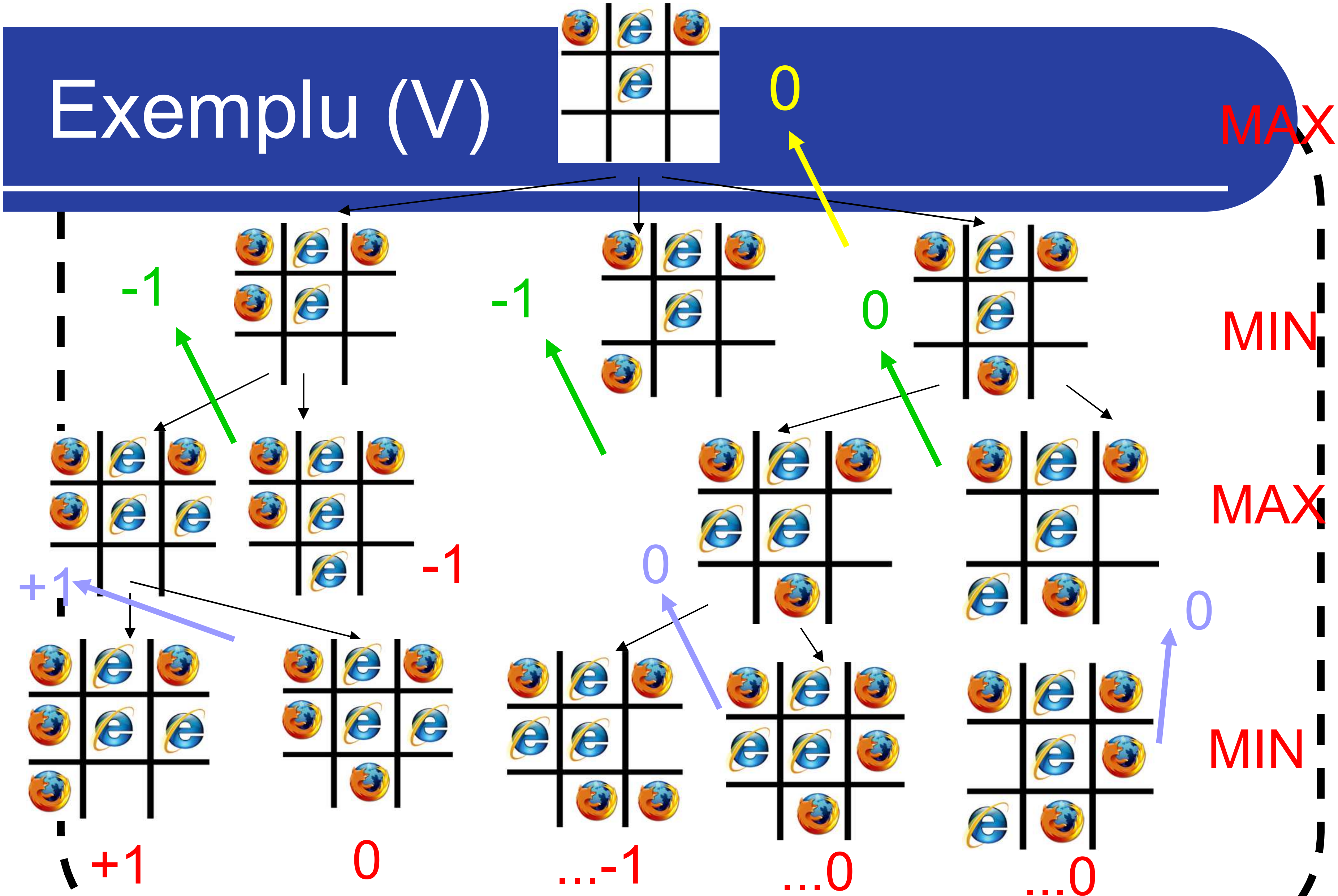


http://mouserunner.com/MozllaTicTacToe/Mozilla_Tic_Tac_Toe.htm

Funcționare Minimax

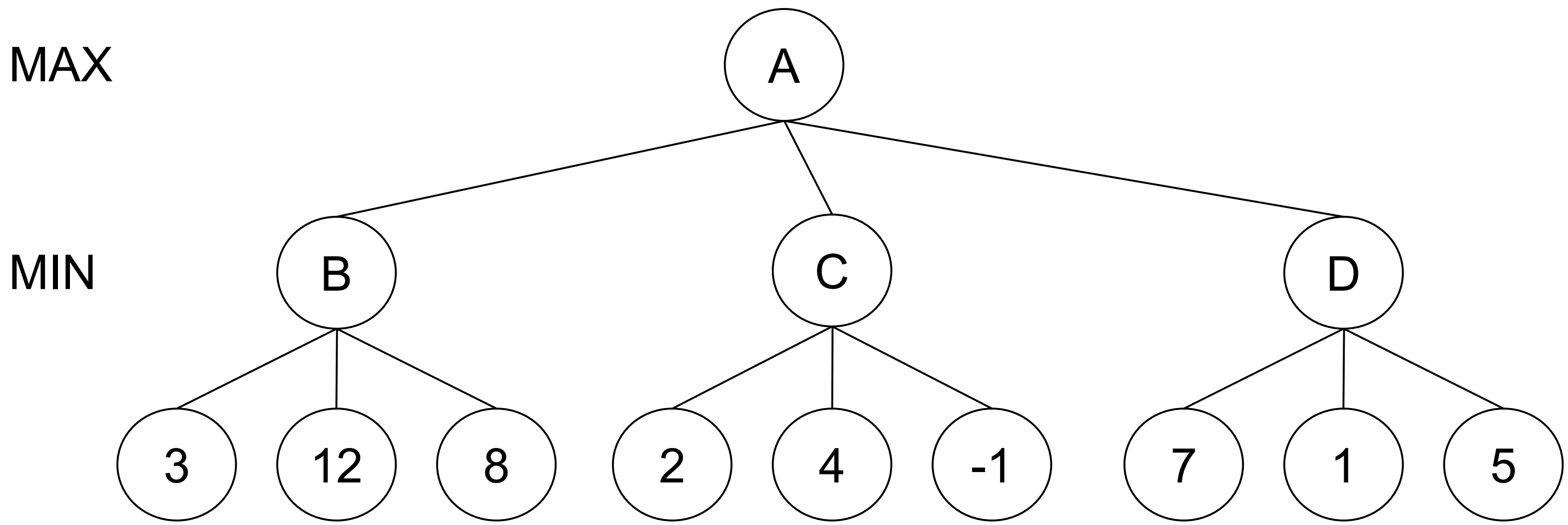
- 1) Se generează întregul arbore;
- 2) Se evaluează frunzele și li se asociază valori;
- 3) Se propagă rezultatele dinspre frunze spre rădăcină astfel:
  - Nivelul MIN alege cea mai mică valoare dintre cele ale copiilor.
  - Nivelul MAX alege cea mai mare valoare dintre cele ale copiilor.

Exemplu (V)



Deci poziția conduce la remiză!

Alt exemplu (I)



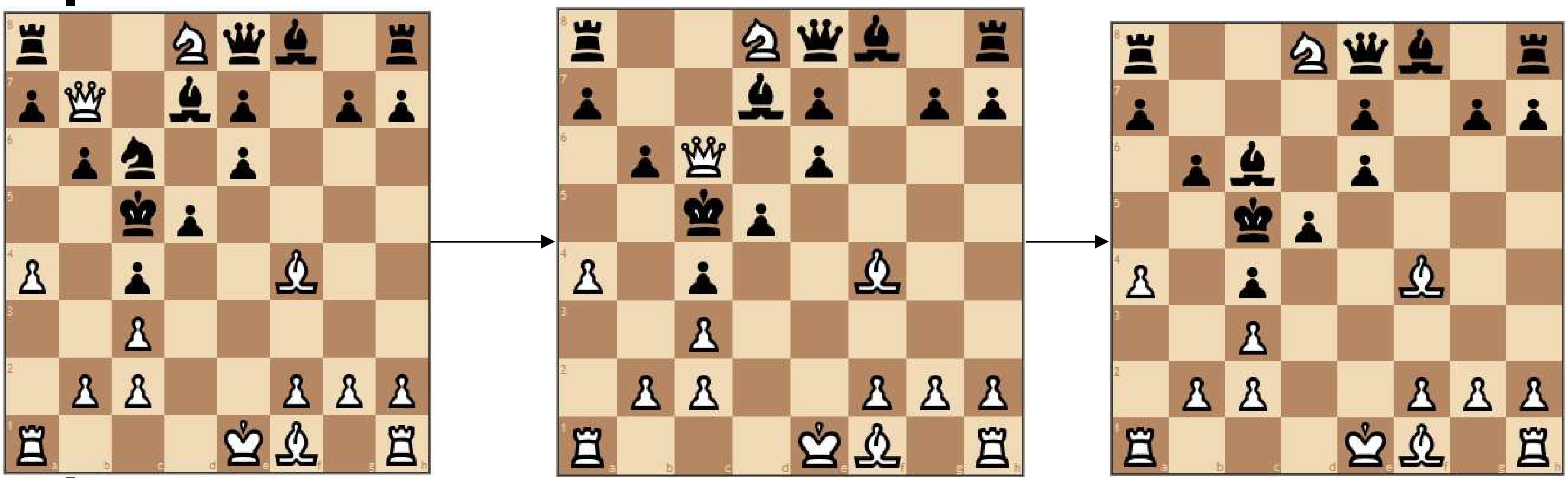
Probleme

- Dimensiunea arborelui pentru “X și 0” e ≤ 9!
- Pentru Șah fiecare nod are în medie 35 copii!
- Pentru Go ramificarea este de cca. 150 – 250!
- Complexitatea arborelui:
  - pentru Șah – 10^{123} noduri;
  - pentru Go – 10^{360} noduri.
- Limitări:→ Nu putem să construim întregul arbore → Nu putem ajunge de fiecare dată la stările finale pentru a le putea evalua.

Optimizări minimax

- Limitarea adâncimii căutării
  - Trebuie să construim o funcție euristică care să estimeze șansele de câștig pentru o poziție dată.
    - Ex. pentru șah:
      - Regină:10p; Turn: 5p; Cal, Nebun: 3p; Pion: 1p;
      - Ex: Funcție de evaluare a poziției = suma pieselor proprii – suma pieselor adversarului.
  - Oprirea căutării:
    - Limitare statică: după un număr maxim de nivele/interval de timp.
    - Limitare dinamică: când profitul obținut din continuarea căutării devine foarte mic (scade sub o valoare fixată).
  - Se estimează valoarea funcției de evaluare la nivelul respectiv.
  - Apoi propagăm valorile conform principiului enunțat anterior.

Exemplu și contraexemplu



Eval: 36-37=-1	Eval: 36-34=2	Eval: 26-34=-8
Funcția nu ține cont de poziție – albul are o poziție net superioară dar funcția de evaluare o ignoră	Dacă căutarea se oprește la acest nivel atunci aparent albul iese în câștig material ignorându-se faptul că la mutarea următoare se pierde dama	În cazul în care căutarea se oprește la acest nivel aparent albul iese in dezavantaj deoarece a pierdut dama

Minimax – funcții de evaluare

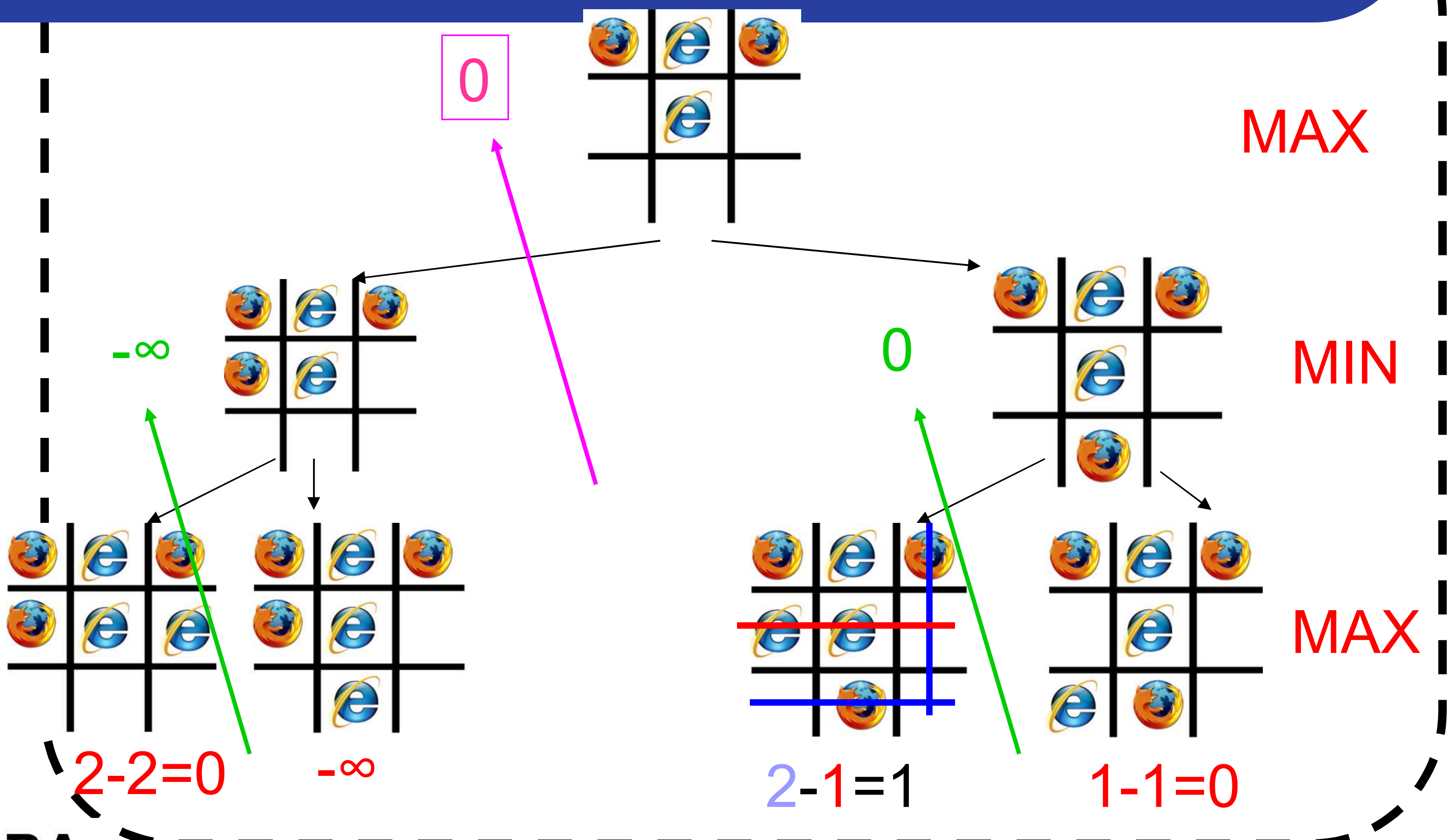
- Funcția euristică trebuie să cuantifice ”poziția”.
  - Chiar în dauna avantajului material.
- Trebuie să ia în calcul potențialele amenințări!

Exemplu funcție euristică X și 0

- F = numărul de linii/coloane/diagonale posibil câștigătoare pentru MAX – numărul de linii/coloane/diagonale posibil câștigătoare pentru MIN.

- Dacă MAX poate să mute și să câștige atunci F = +∞; dacă MIN poate să mute și să câștige F = -∞.

Exemplu funcție euristică X și 0



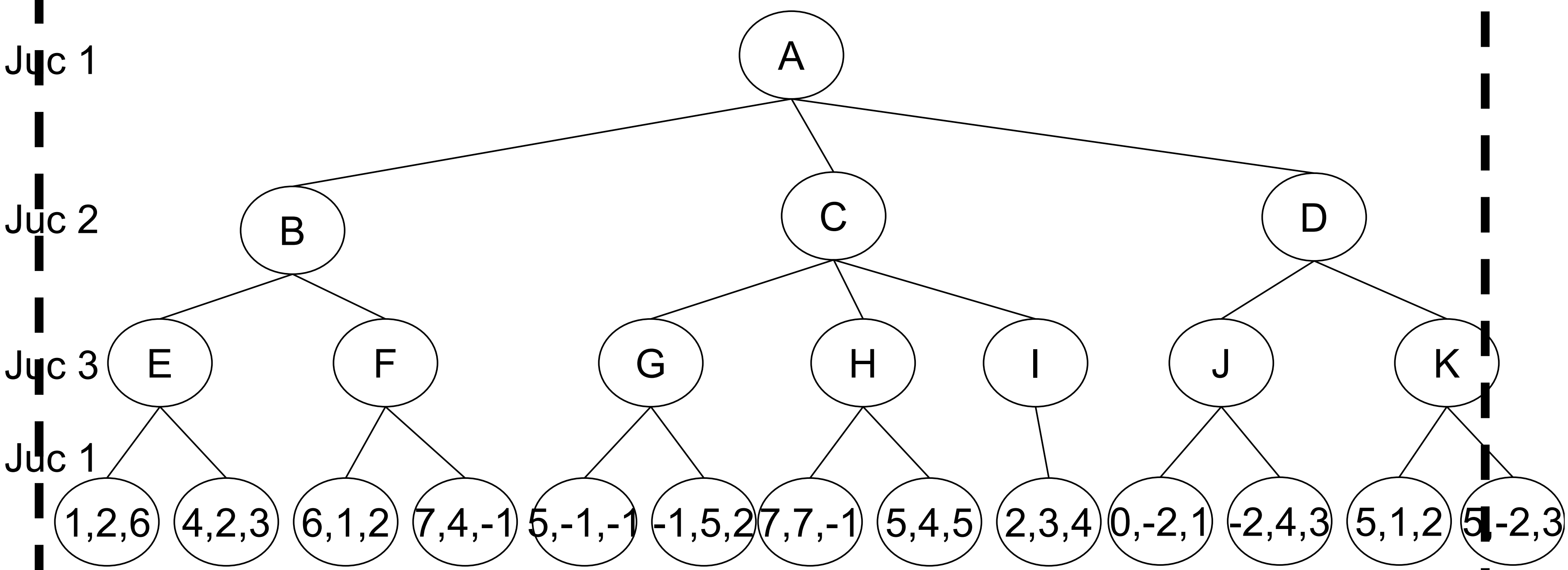
Algoritm MINIMAX

- MINIMAX_limitat (n, nivel_limită)
  - **Pentru fiecare** n’ ∈ succs(n) // pentru toate mutările
    - Fie m = mutarea corespunzătoare arcului (n, n’)
    - VAL(m) = w(n’, nivel_limită, 1) // determin valoarea mutării
  - **Întoarce** m a.î. VAL(m) = max {VAL(x)| x ∈ mutări(n)}

- W(n, limită, nivel)
  - **Dacă** n este frunză **Întoarce** cost(n)
  - **Dacă** nivel ≥ limită **Întoarce** euristică(n)
  - **Dacă** jucătorul MAX este la mutare **Întoarce**
    - max {w(n’, limită, nivel + 1) | n’ ∈ succs(n)}
  - **Dacă** jucătorul MIN este la mutare **Întoarce**
    - min {w(n’, limită, nivel + 1) | n’ ∈ succs(n)}

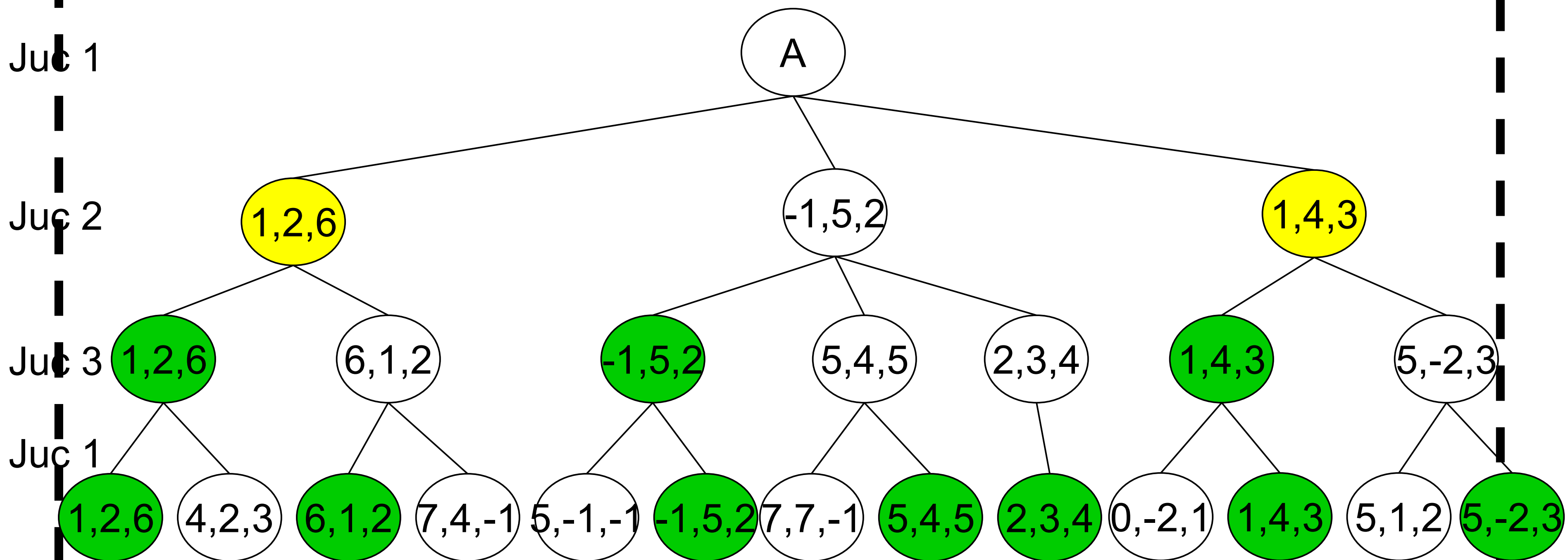
Caz special - Minimax 3 jucatori (1)

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu



Caz special - Minimax 3 jucatori (2)

Jucătorii vor alege pe rând valoarea care le maximizează câștigul propriu

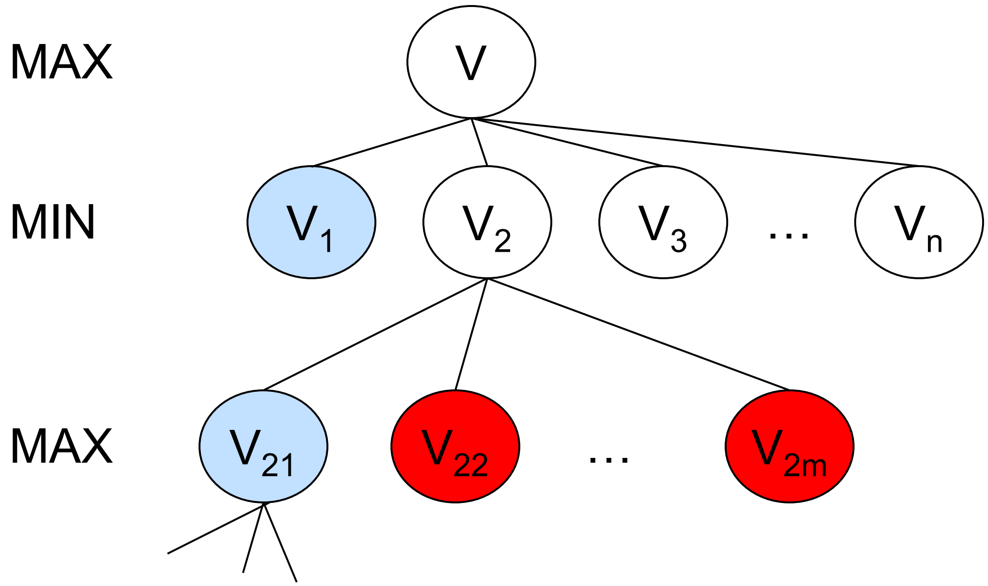


Caz special (2) – Minimax Probabilistic

- La unele jocuri, mutările sunt guvernate de șansă.
- Ex: Jocul de Table – mulțimea mutărilor este limitată de:
  - starea curentă a jocului;
  - combinația zarurilor în starea curentă.
- Arborele MINIMAX este completat cu noduri suplimentare (noduri șansă) plasate între nodurile MIN/MAX (MIN – șansă – MAX și MAX – șansă – MIN).
- Valorile se calculează ca sumă ponderată între probabilitatea nodului și evaluarea acestuia (prin cost sau euristică).

Tăiere α-β

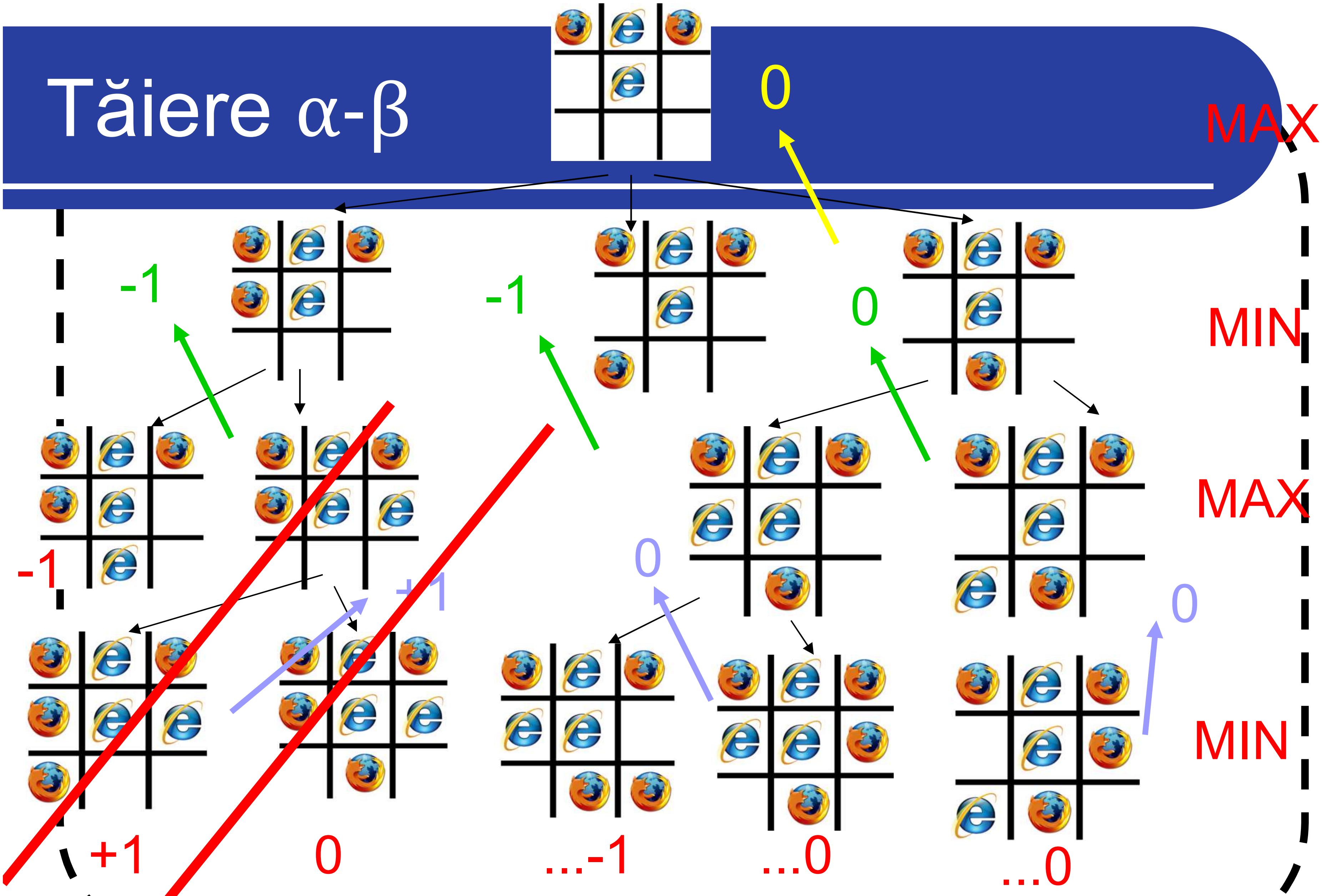
- Încercăm să limităm spațiul de căutare prin eliminarea variantelor ce nu au cum să fie alese.
- Idee:
  - Dacă $V_{21}<V_1$ toată ramura V_2 poate fi Ignorată.



Tăiere α-β

- α = max dintre valorile găsite pentru un nod MAX.
- β = min dintre valorile găsite pentru un nod MIN.
- Tăiem o ramură dacă:
  - am găsit un nod pe nivelul MAX cu valoare β <= oricare din valorile α calculate anterior;
  - am găsit un nod pe nivelul MIN cu valoare α >= oricare din valorile β calculate anterior.
- Teorema α-β. Fie J un nod din arborele MINIMAX explorat. Daca $\alpha(J) \geq \beta(J)$, atunci explorarea nodului J nu este necesară.

Tăiere α-β



Min va alege întotdeauna -1 Indiferent de ce alte opțiuni mai sunt!

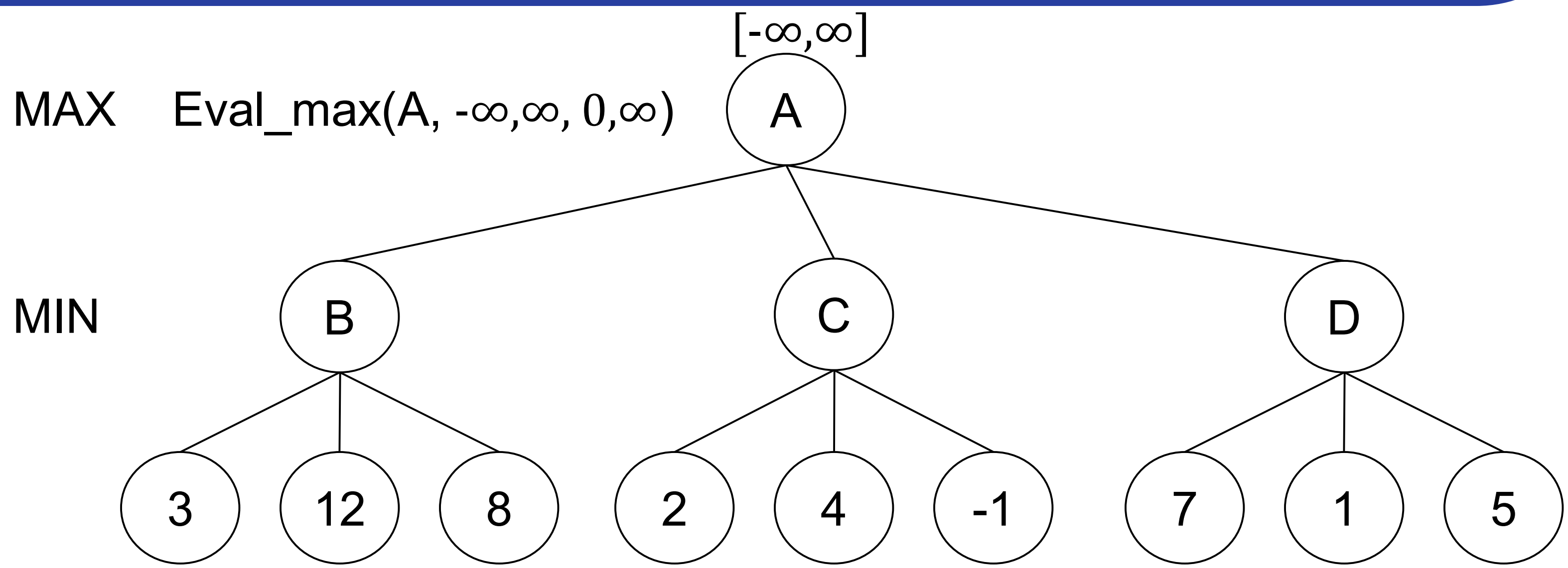
Algoritm α-β

- α-β(n, limită)
  - w = eval_max(n, -∞, ∞, 0, limită)
  - **Întoarce** m ∈ mutări(n) a.i. VAL(m) = w

- eval_max(n, α, β, nivel, limită)
  - **Dacă** n este frunză **Întoarce** cost(n)
  - **Dacă** (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat
  - a = -∞ // valoarea curenta a nodului de tip Max
  - **Pentru fiecare** (n' ∈ succs(n))
    - a = max(a, eval_min(n', max(α,a), β, nivel+1, limită)); // propag
    - **Dacă** (a ≥ β) **oprire**;
  - **Întoarce** a

- similar eval_min

Alt exemplu (II)



eval_max(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

a = -∞ // valoarea curenta a nodului de tip max

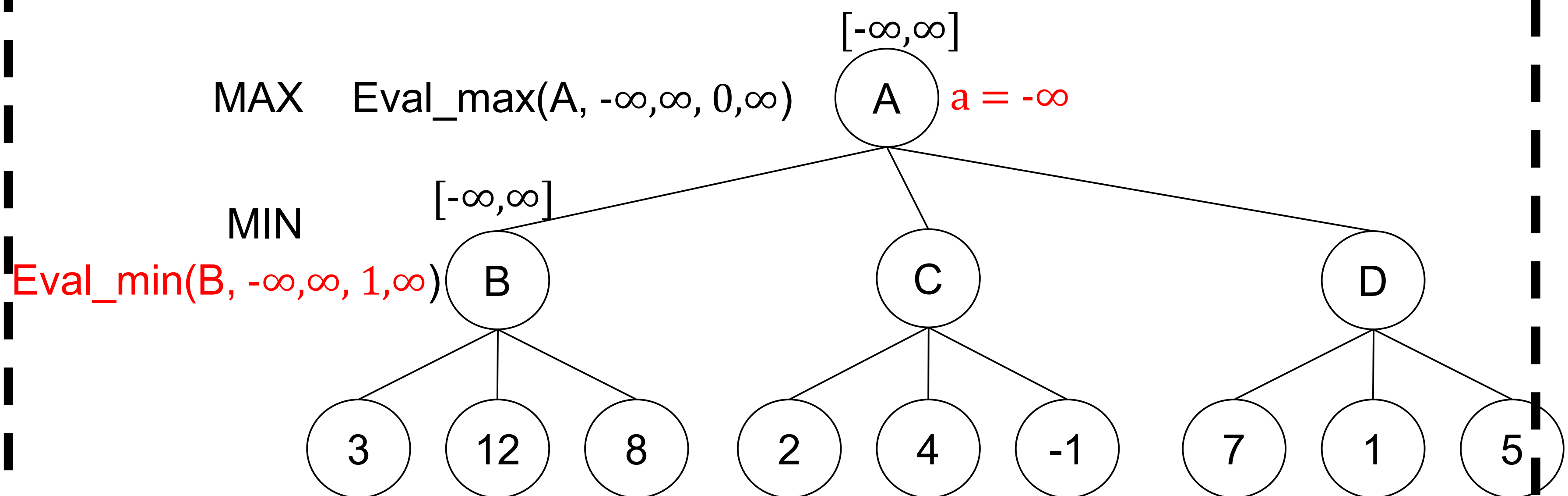
Pentru fiecare (n' ∈ succs(n)) {

a = max(a, eval_min(n', max(α,a), β, nivel+1, limită)); // propag

Dacă (a ≥ β) **oprire**; }

Întoarce a

Alt exemplu (III)



eval_max(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

a = -∞ // valoarea curenta a nodului de tip max

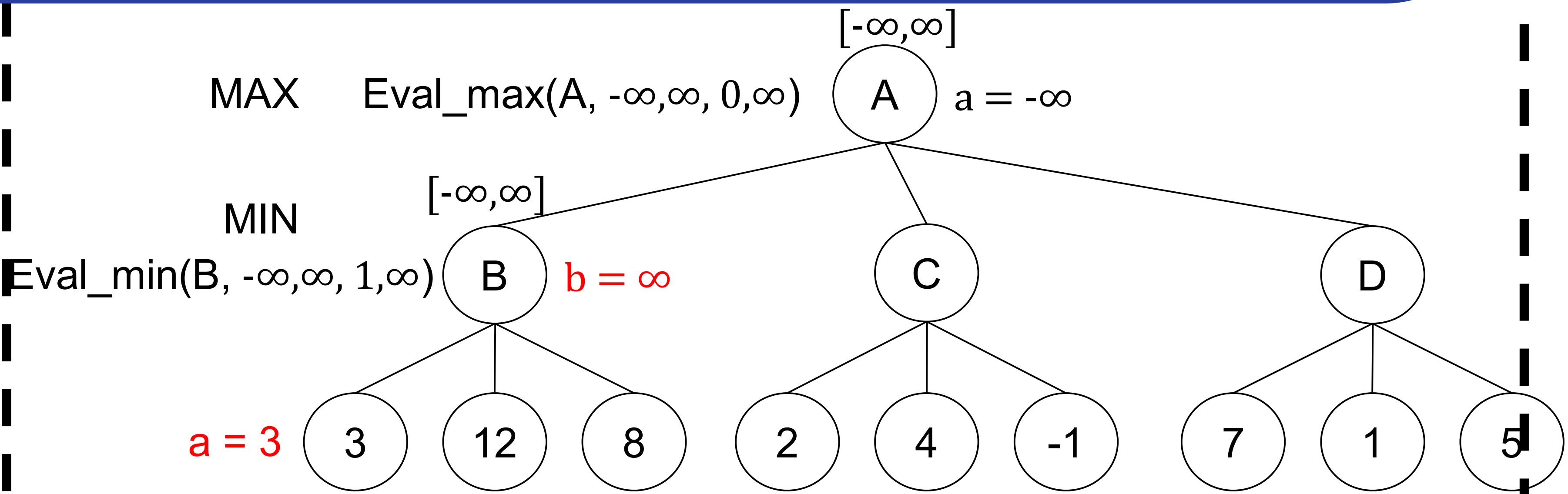
Pentru fiecare (n' $\in$ succs(n)) {

a = max(a, eval_min(n', max(α,a), β, nivel+1, limită)); // propag

Dacă ($a \geq \beta$) **oprire**; }

Întoarce a

Alt exemplu (IV)



eval_min(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

b = ∞ // valoarea curenta a nodului de tip min

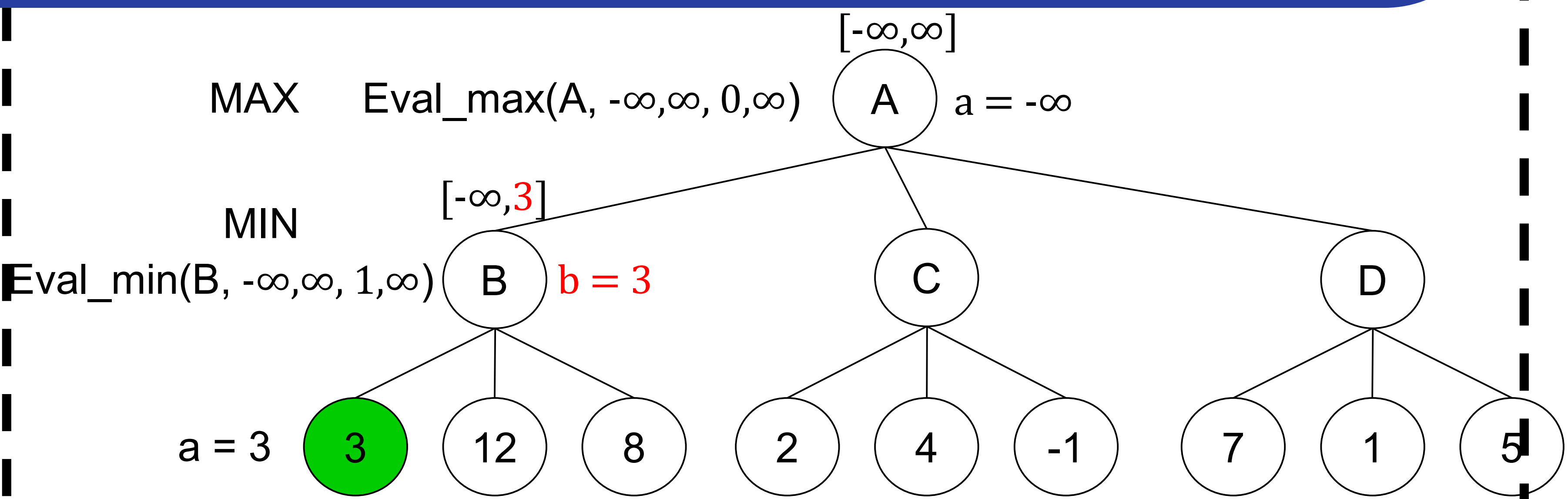
Pentru fiecare (n' ∈ succs(n)) {

b = min(b, eval_max(n', α, min(β,b), nivel+1, limită)); // propag

Dacă (b ≤ α) **oprire**; }

Întoarce b

Alt exemplu (V)



eval_min(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

b = ∞ // valoarea curenta a nodului de tip min

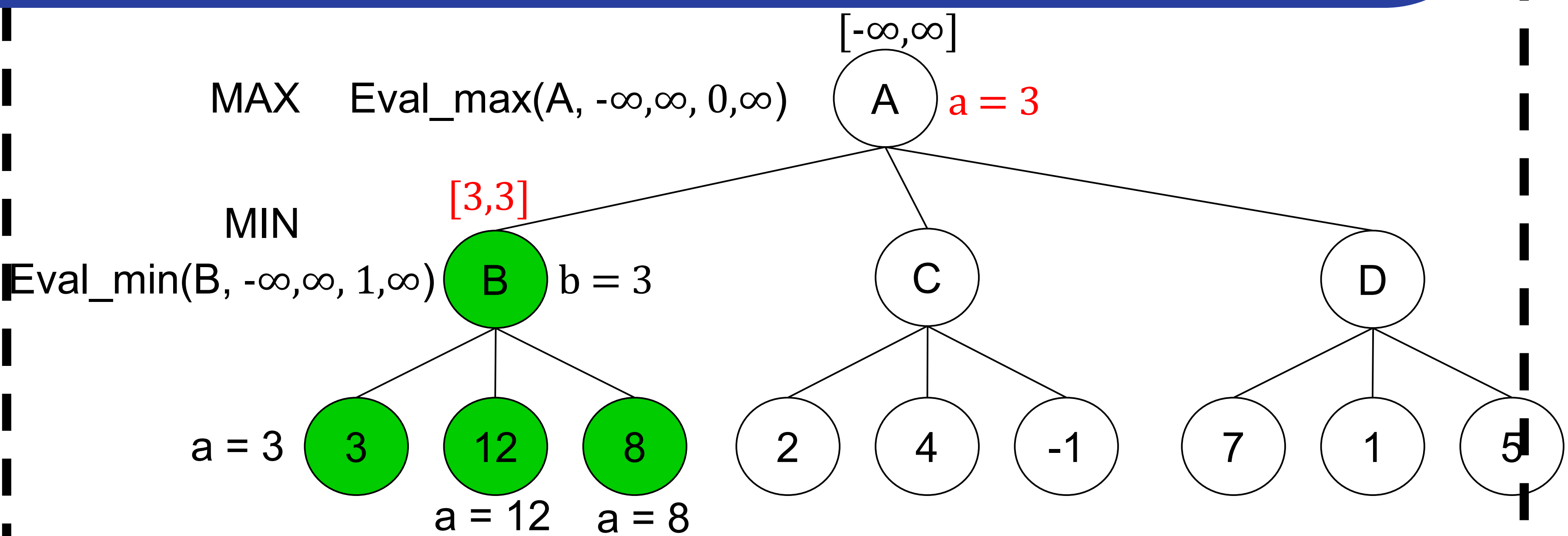
Pentru fiecare (n' ∈ succs(n)) {

b = min(b, eval_max(n', α, min(β,b), nivel+1, limită)); // propag

Dacă (b ≤ α) **oprire**; }

Întoarce b

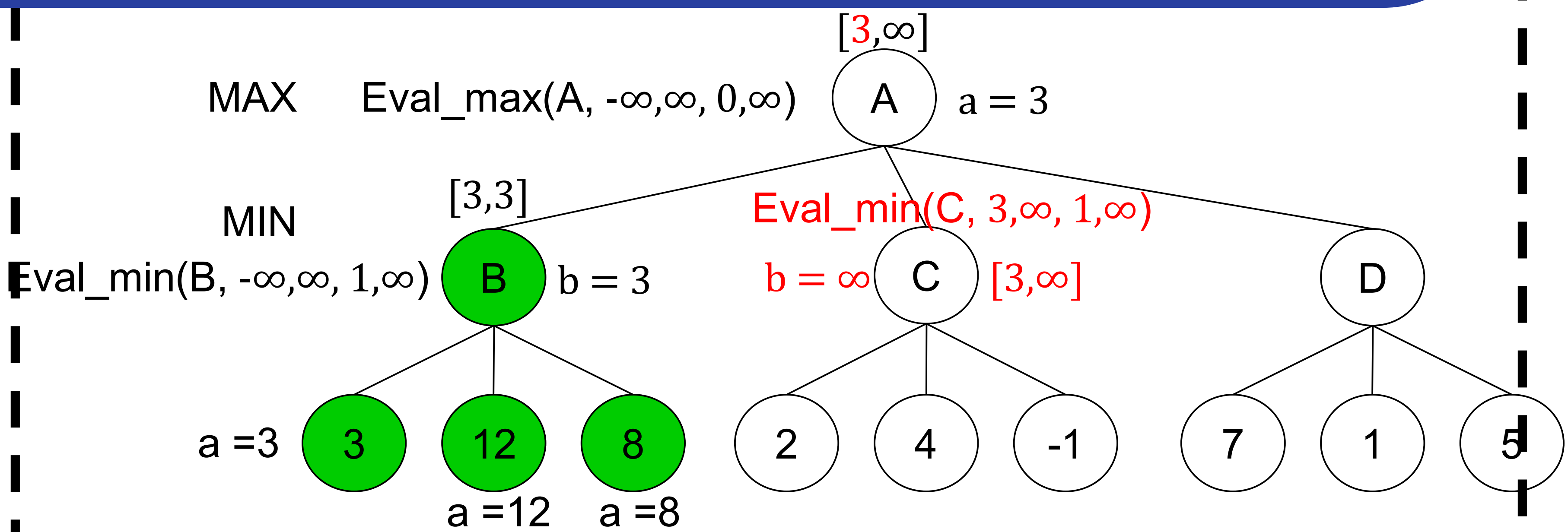
Alt exemplu (VI)



eval_max(n, α, β, nivel, limită)

- **Dacă** n este frunză **Întoarce** cost(n)
- **Dacă** (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat
- a = -∞ // valoarea curenta a nodului de tip max
- **Pentru fiecare** (n' ∈ succs(n)) {
  - a = max(a, eval_min(n', max(α,a), β, nivel+1, limită)); // propag
  - **Dacă** (a ≥ β) **oprire**; }
- **Întoarce** a

Alt exemplu (VII)



eval_min(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

b = ∞ // valoarea curenta a nodului de tip min

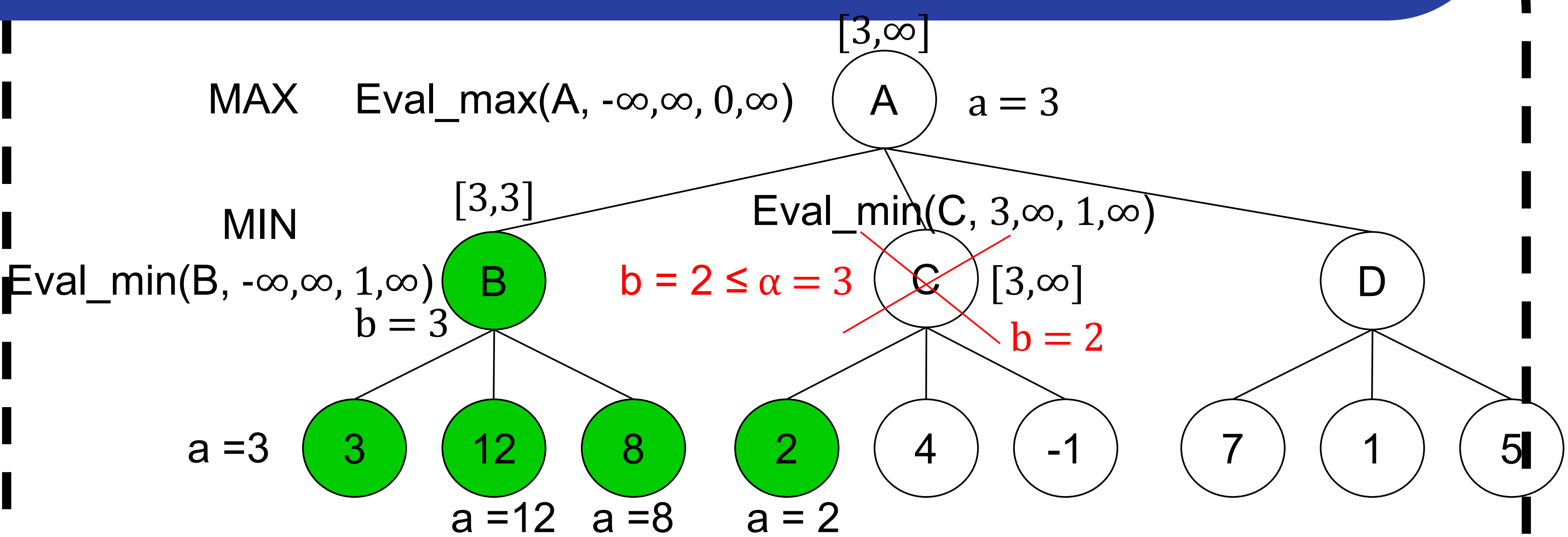
Pentru fiecare (n' ∈ succs(n)) {

b = min(b, eval_max(n', α, min(β,b), nivel+1, limită)); // propag

Dacă (b ≤ α) **oprire**; }

Întoarce b

Alt exemplu (VIII)



eval_min(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

b = ∞ // valoarea curenta a nodului de tip min

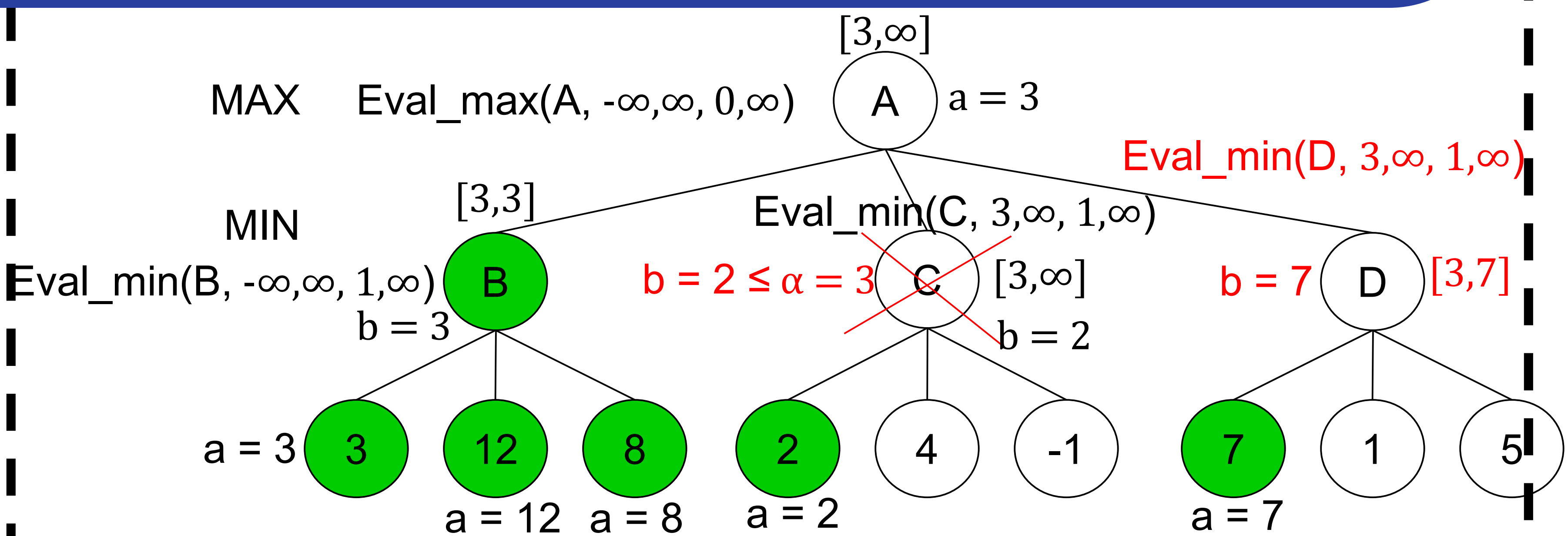
Pentru fiecare (n' ∈ succs(n)) {

b = min(b, eval_max(n', α, min(β,b), nivel+1, limită)); // propag

Dacă (b ≤ α) **oprire**; }

Întoarce b

Alt exemplu (IX)



eval_min(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

b = ∞ // valoarea curenta a nodului de tip min

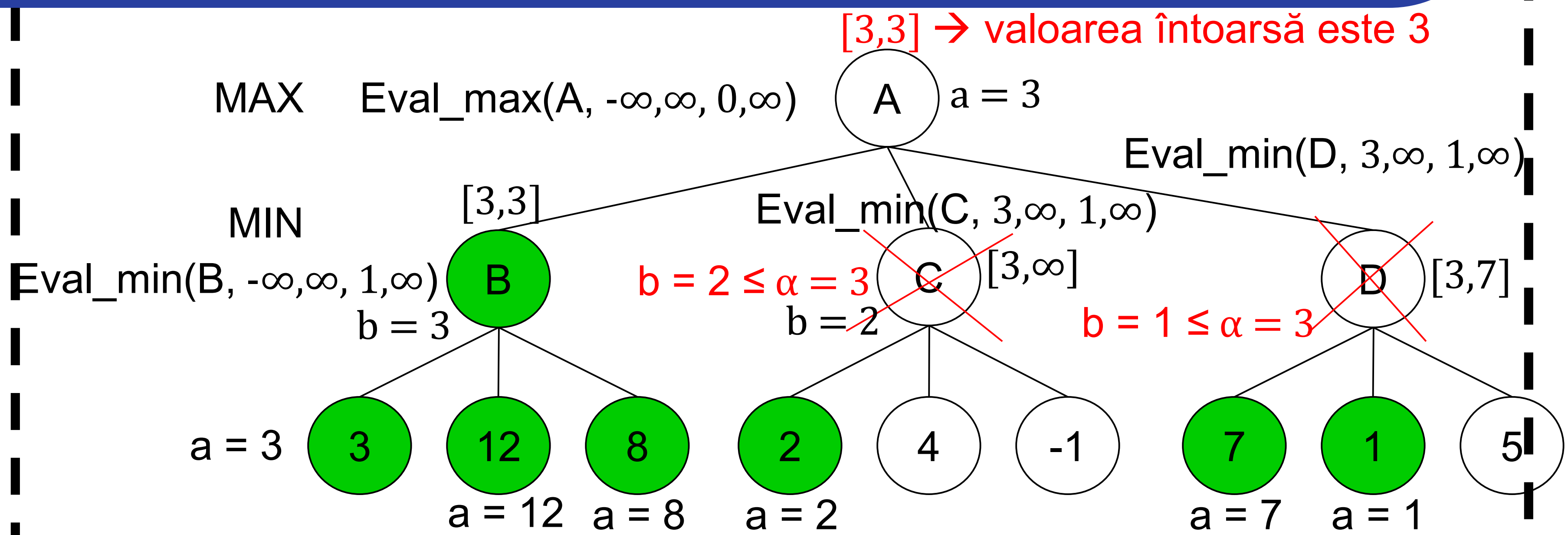
Pentru fiecare (n' ∈ succs(n)) {

  b = min(b, eval_max(n', α, min(β,b), nivel+1, limită)); // propag

  Dacă (b ≤ α) **oprire**; }

Întoarce b

Alt exemplu (X)



eval_max(n, α, β, nivel, limită)

Dacă n este frunză **Întoarce** cost(n)

Dacă (nivel ≥ limita) **Întoarce** euristică(n) // sunt limitat

a = -∞ // valoarea curenta a nodului de tip max

Pentru fiecare (n' ∈ succs(n)) {

a = max(a, eval_min(n', max(α,a), β, nivel+1, limită)); // propag

Dacă (a ≥ β) **oprire**; }

Întoarce a

Observații α-β

- Reduce complexitatea minimax în cazul ideal de la
  - Număr_ramificări$^{\text{număr_nivele}}$ la Număr_ramificări$^{\text{număr_nivele/2}}$
- Contează foarte mult ordinea în care analizăm mutările!
  - Sortarea mutărilor după un criteriu dat nu este costisitoare comparativ cu costul exponențial al algoritmului.
- Se folosesc euristici pentru a alege mutările examinate mai întâi:
  - ex: la șah se aleg întâi mutările în care se iau piese;
  - sau se aleg mai întâi mutările cu scor bun în parcurgeri precedente;
  - sau se aleg mutările care au mai generat tăieri.

Observații MINIMAX și α-β

- Algoritmi de căutare în adâncime.
- Pot cauza probleme când avem un timp limită.
- → Soluție posibilă IDDFS (căutare în adâncime mărind iterativ adâncimea maximă până la care căutăm).

Concluzii

- Algoritmi cu complexitate foarte mare.
- Soluții euristice pentru limitarea complexității.
- Recomandabil să se combine cu alte strategii – baze de date cu poziții, pattern-matching.

ÎNTREBĂRI?