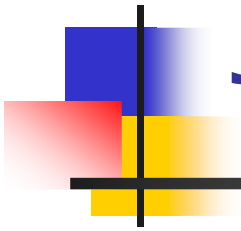
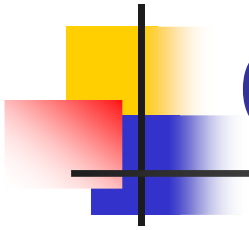


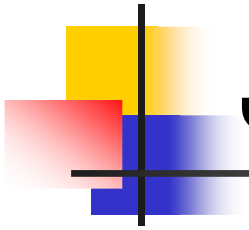
Jogos

Prof. Carlos Lopes



Os Jogos em IA

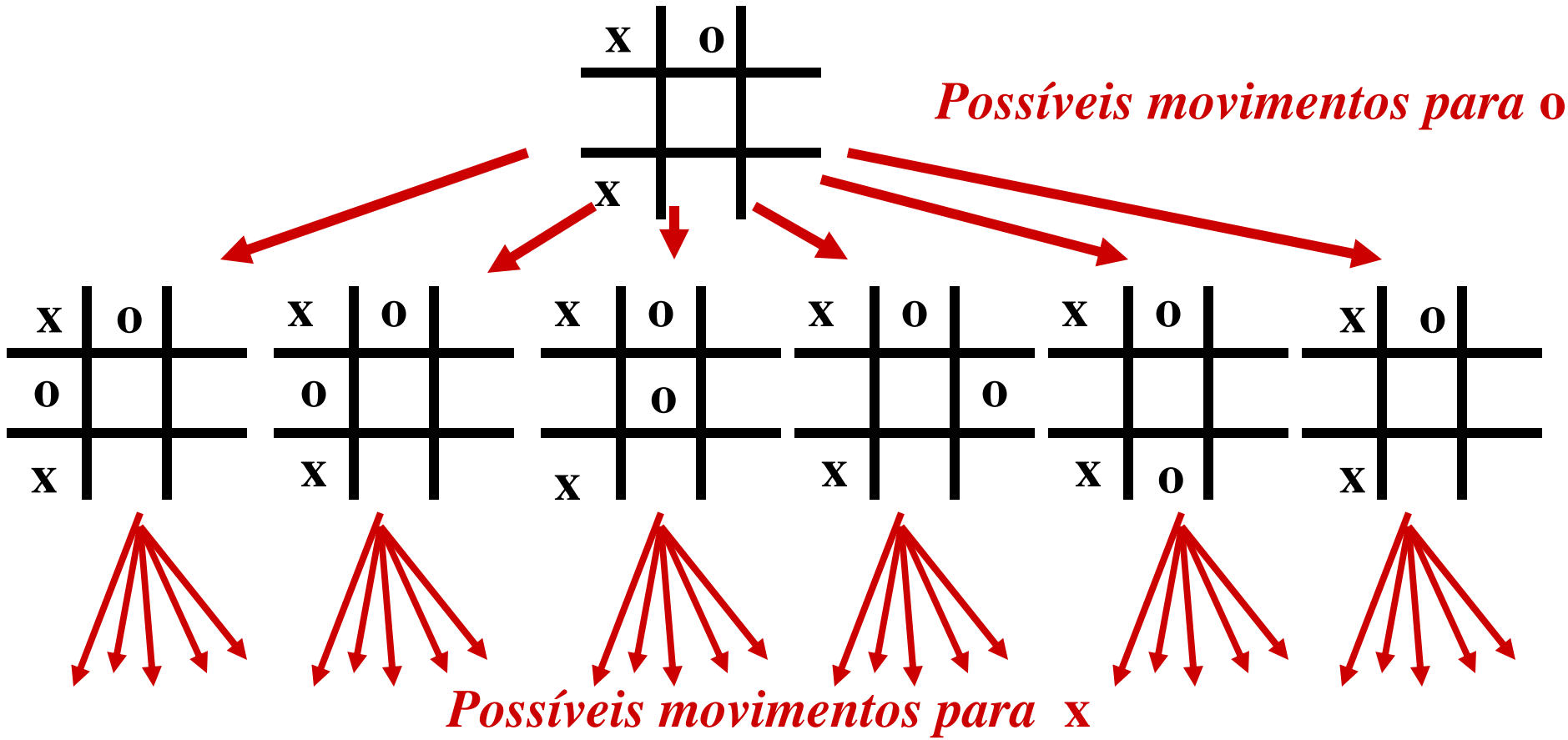
- A forma de jogos mais estudada em IA é determinística, com dois jogadores e alternância entre eles e com perfeita informação. Isto significa que:
  - o ambiente é determinístico
  - completamente observável
  - multiagente: existem dois agentes que se alternam na seqüência de movimentos e os valores de utilidade no fim do jogo são sempre iguais e opostos.

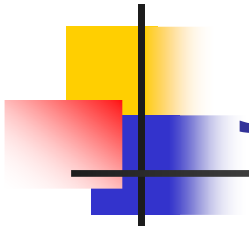


Jogos e Busca Competitiva

Árvore de jogo: árvore semântica na qual os nós representam configurações do tabuleiro (ou estados) e os movimentos (operadores) representam transições entre elas. Desta forma pode ser possível utilizar uma estratégica de busca que gere uma seqüência de movimentos que permita ganhar a partida.

Exemplo de uma Árvore de Jogo

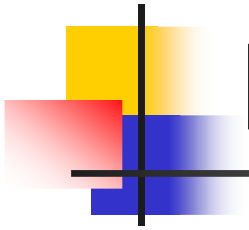




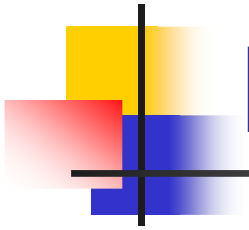
Jogos e Busca Competitiva

- A maioria dos jogos com 2 jogadores requer jogadas sucessivas de cada jogador.
  - Cada transição é um movimento
- A maioria dos jogos não-triviais não admite busca exaustiva: árvores muito grandes.
- Assim, faz-se necessário usar uma técnica de busca associada a uma heurística.

Jogos e Busca Competitiva: Definições

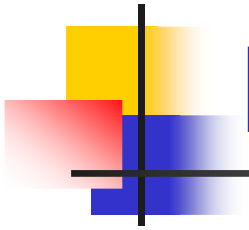


- **Estado inicial:** configuração inicial e indicação de quem deve iniciar o movimento
- **Operadores:** definem os movimentos permitidos
- ***Ply***: número de níveis na árvore, incluindo a raíz
- **Teste terminal:** define quando o jogo termina
- **Função utilidade:** fornece um valor numérico para o resultado do jogo



Procedimento Minimax

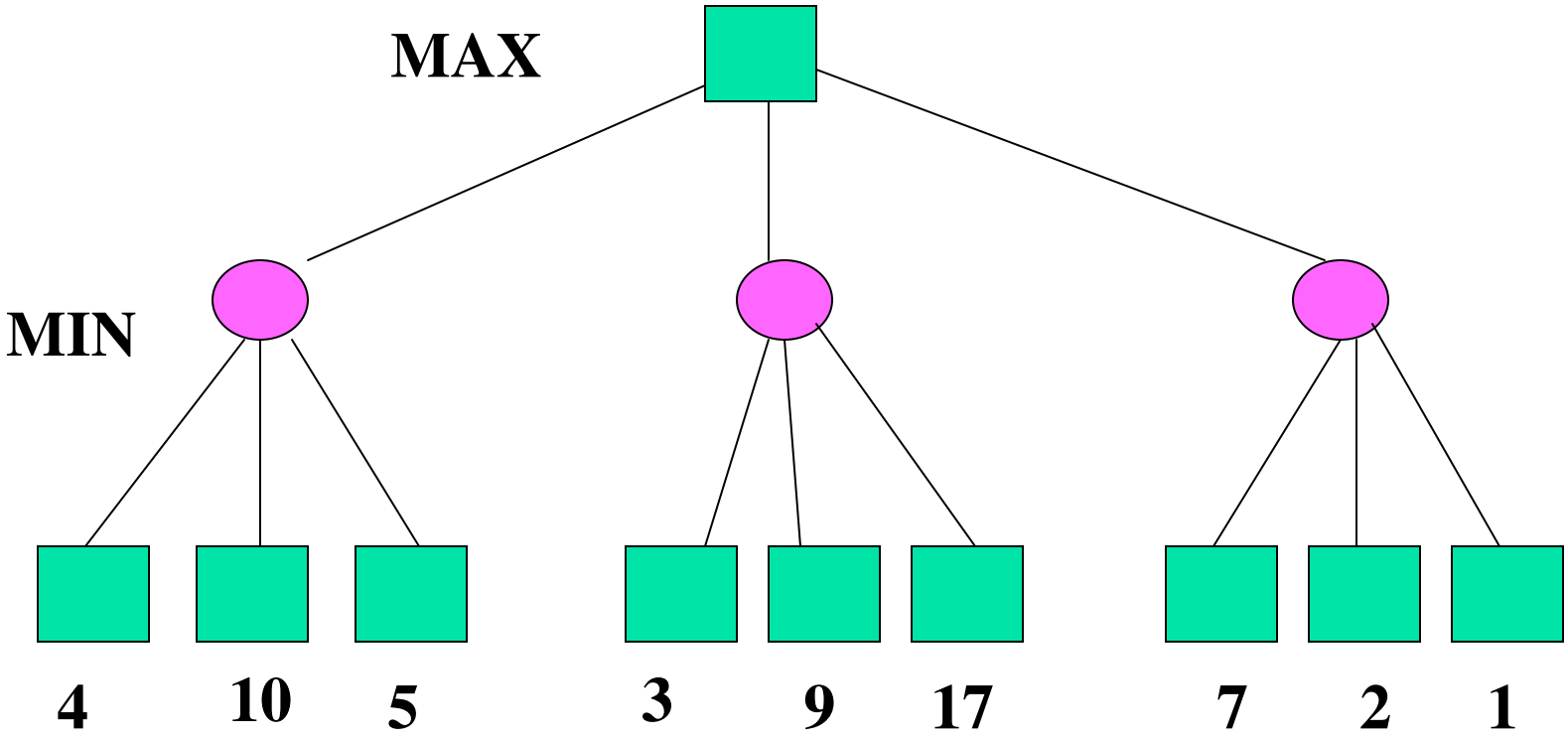
- Jogos envolvem competição:
  - Dois jogadores estão trabalhando para atingir objetivos conflitantes
  - Assim, a árvore de busca difere dos exemplos anteriores já que as jogadas de cada jogador visam objetivos conflitantes: *não existe uma busca para um simples objetivo !*
- **Avaliação Estática**: valor numérico que representa a qualidade da configuração (tabuleiro)
  - Realizada por um avaliador estático
    - Valores positivos: vantagem para um jogador
    - Valores negativos: vantagem para o outro



Procedimento Minimax (cont.)

- **Maximizador:** jogador esperando por avaliações (números) positivas
- **Minimizador:** jogador esperando por avaliações (números) negativas
- A árvore de jogo consiste em camadas sucessivas de maximização e minimização
  - Presume-se que, em cada camada, o jogador deseja a avaliação mais vantajosa para ele

Procedimento Minimax (cont.)



O Algoritmo Minimax

function MINIMAX-DECISION(*state*) returns *an action*
inputs: *state,* current state in game
***v*← MAX-VALUE(*state*)**
return the *action* in SUCCESSORS(*state*) with value *v*

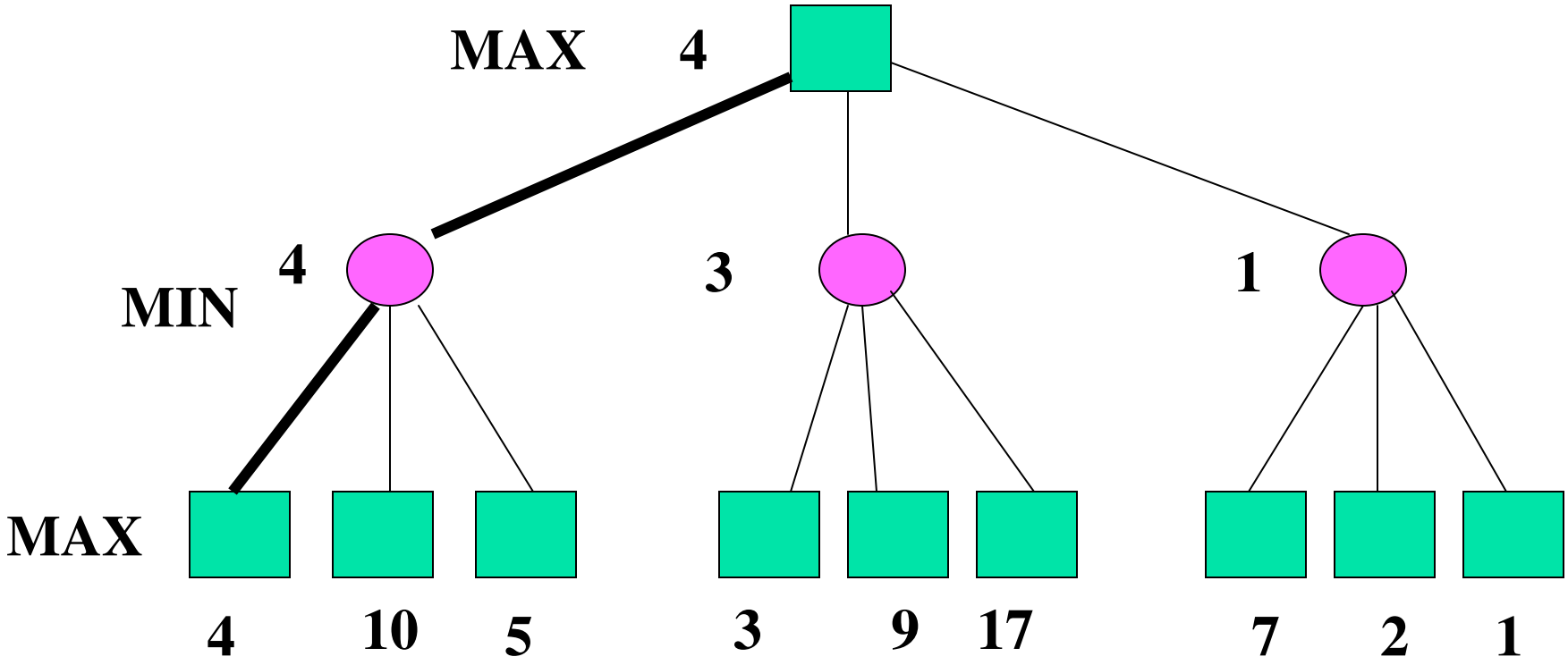
---

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

---

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

O Algoritmo Minimax



O Desempenho de Minimax

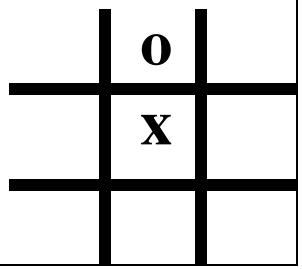
- Assumindo uma busca em profundidade :
  - **Complexidade de tempo =** $O(b^m)$
  - **Complexidade de espaço:** $O(bm)$
- Para jogos reais uma busca exaustiva é inviável. Por exemplo, a árvore de busca para xadrez tem cerca de 35^{100} nós. Então como melhorar o processo? Duas possibilidades:
  - **Podemos avaliar nós que não são terminais usando uma função de avaliação heurística.**
  - **É possível podar partes do espaço de busca que não necessitam ser examinadas.**

Funções de Avaliação

- Com uma função de avaliação estática não é necessário expandir a árvore de busca completamente.
- Uma função de avaliação estática estima o quão boa é a configuração do tabuleiro com respeito a um jogador (tipicamente MAX)
- A qualidade dos movimentos selecionados selecionados por minimax baseada em profundidade é uma função da qualidade do avaliador do tabuleiro estático.
- Uma boa função de avaliação:
  - **corretamente reflete a probabilidade de vitória para um dado nó na árvore de busca.**
  - **deve ser calculada de forma eficiente.**

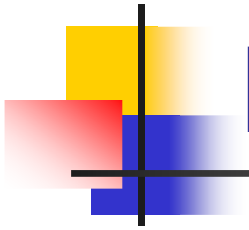
Um Exemplo usando o Jogo da Velha

- Um exemplo de função de avaliação para a posição do tabuleiro *p*:
  - *Eval(p)* = (o número de linhas completas, colunas, ou diagonais que estão ainda abertas para MAX)– (o número de linhas completas, colunas, ou diagonais que estão ainda abertas para MIN).
  - *Eval(p)* = ∞ se MAX ganha
  - *Eval(p)* = - ∞ se MIN ganha.



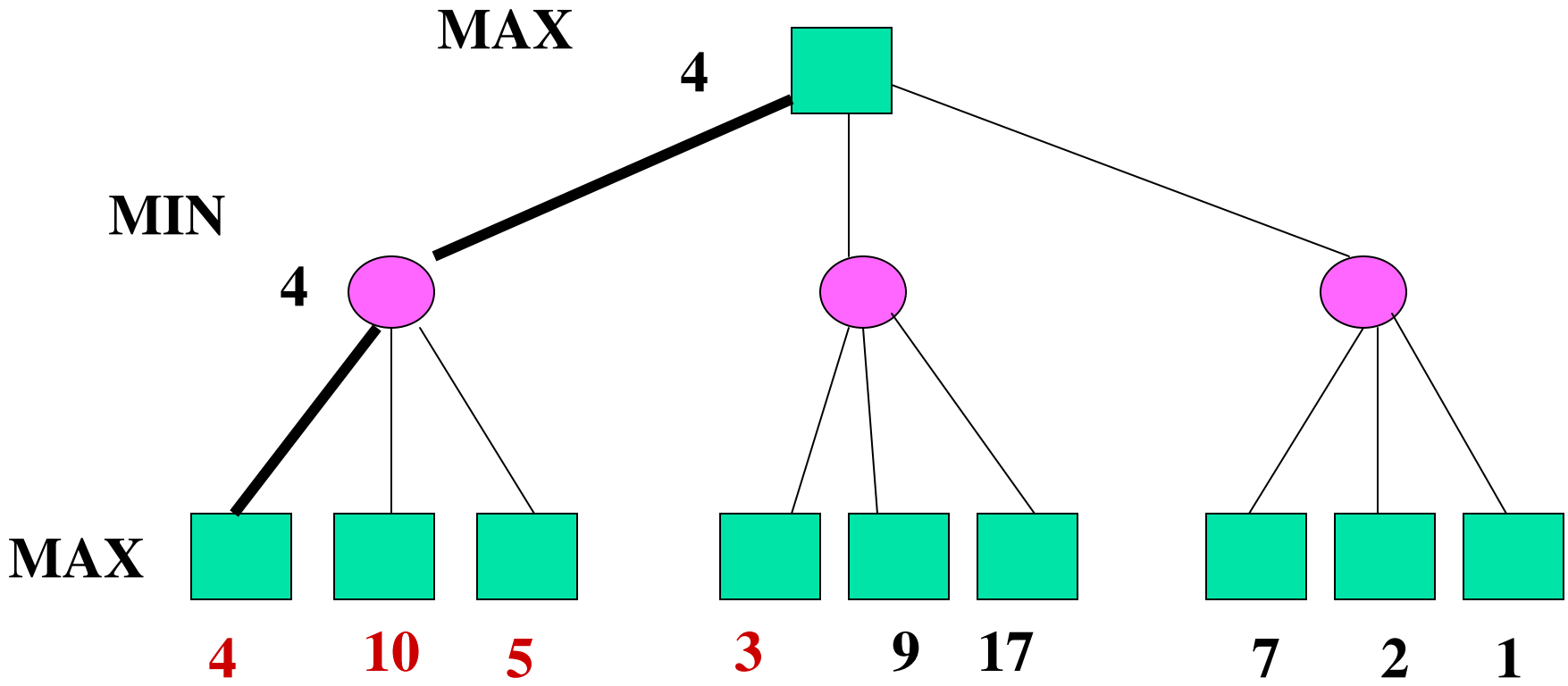
$$Eval(p) = 6 - 4 = 2$$

Reduzindo o Espaço de Busca



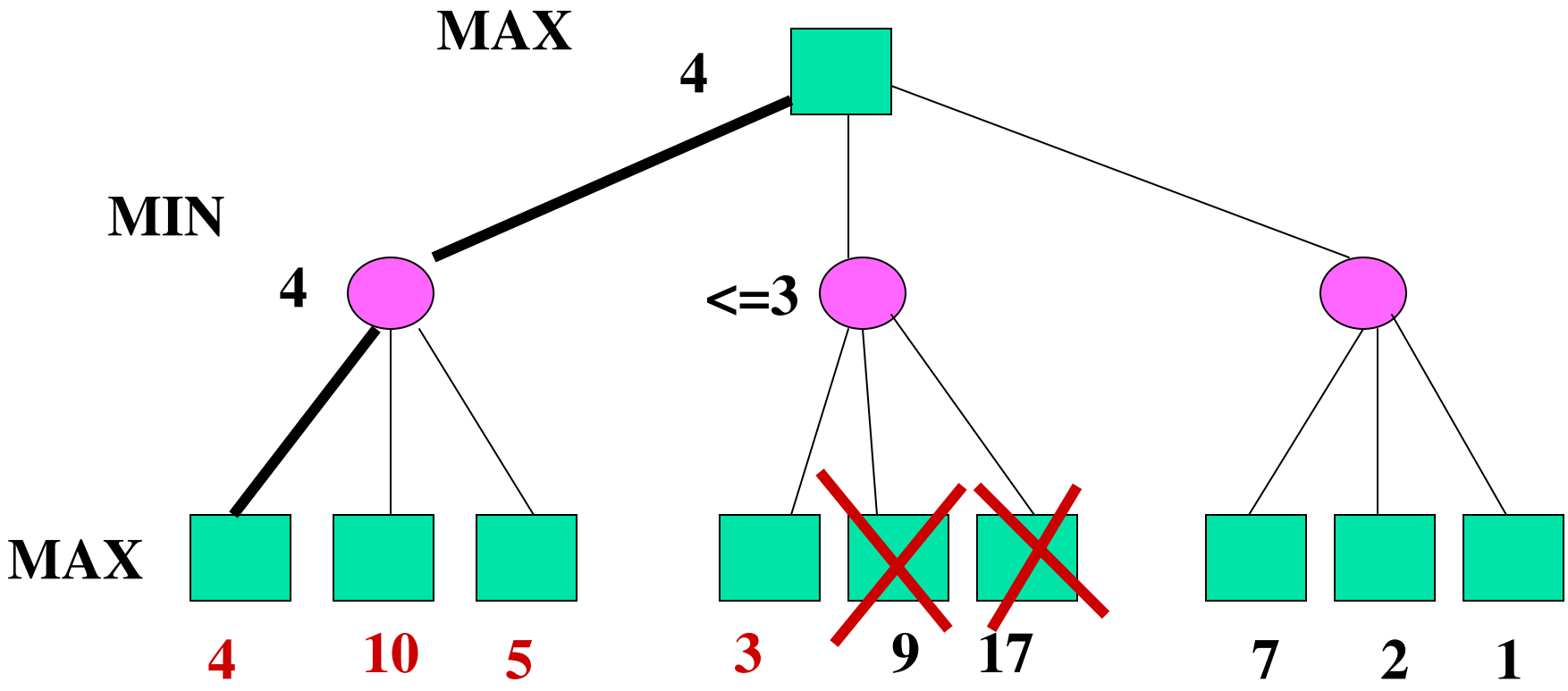
- ***Poda alfa-beta***: o valor de um nó é relevante somente se existe uma possibilidade de nos depararmos com ele durante o processo de busca. Se pudermos provar que jogadores racionais jamais atingirão tal nó, independente do seu valor, então não existe a possibilidade de examina-lo ou mesmo gera-lo.

Alfa-Beta: Motivação

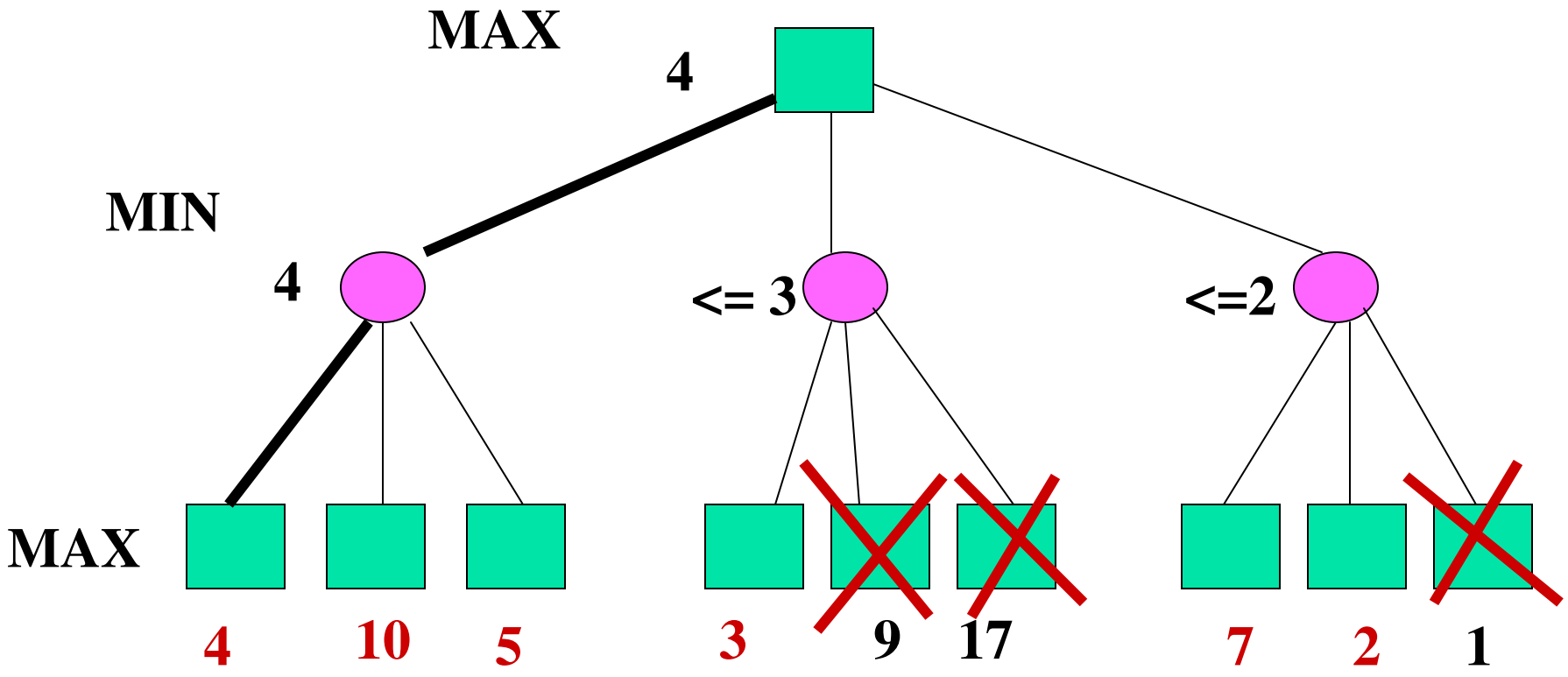


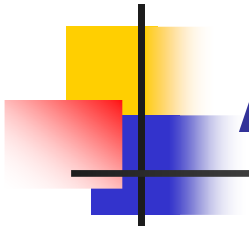
Após considerar a subárvore esquerda, sabemos que MAX tem um movimento com valor 4. Após expandir o primeiro nó da subárvore central O que podemos concluir?

Alfa-Beta (cont.)



Alfa-Beta (cont.)



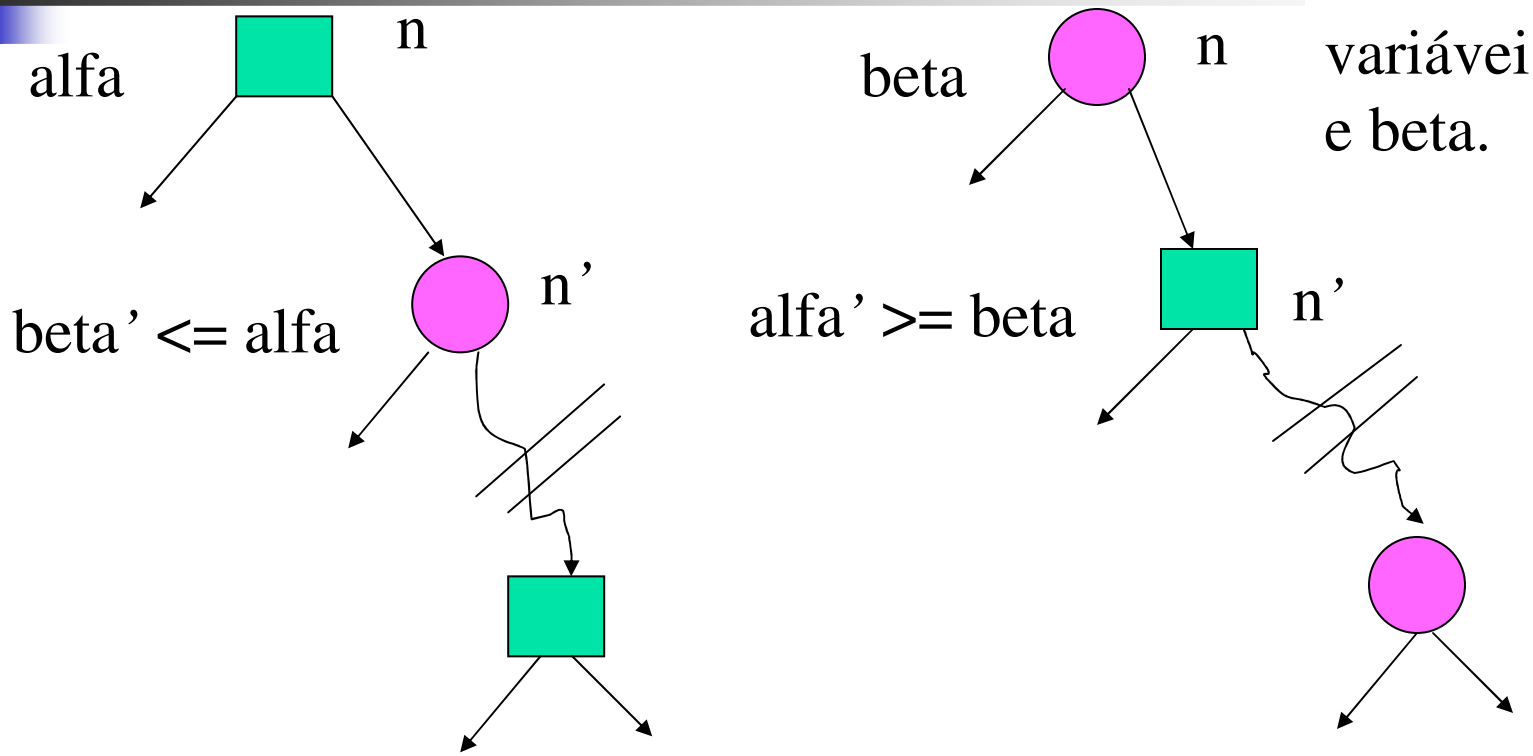


Alfa e Beta

- ***Alfa***, ou α, é o valor da melhor escolha encontrada *até o momento* para qualquer ponto de escolha ao longo do caminho para MAX.
- ***Beta***, ou β, é valor da melhor escolha encontrada *até o momento* para qualquer ponto de escolha ao longo do caminho para MIN.
- Os valores de alfa e beta são modificados durante a busca.
- Estes dois valores são comparados para fazer a redução.

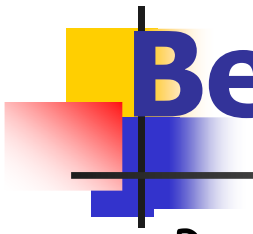
O Princípio da Poda

Todo nó tem duas variáveis: alfa e beta.



A busca pode ser descontinuada abaixo de um nó MIN tendo beta’ <= o alfa de seu pai MAX. Ela pode ser descontinuada abaixo de um nó MAX tendo um alfa’ >= o beta de seu pai MIN.

O Algoritmo de Busca Alfa-Beta



Parecido com MINIMAX com alfa e beta adicionados

function ALPHA-BETA-SEARCH(state) **returns** *an action*

inputs: *state*, current state in game

$v \leftarrow$ MAX-VALUE(*state*, $-\infty,+\infty$)

return the *action* in SUCCESSORS(*state*) with value v

Algoritmo Alfa-Beta (cont.)

function MAX-VALUE(*state*, α, β) **returns** *a utility value v*

inputs: ***state***, current state in game

α, the best alternative for MAX along the path to ***state***

β, the best alternative for MIN along the path to ***state***

if TERMINAL-TEST(state) **then return** UTILITY(state)

$v \leftarrow -\infty$

for each *s* in SUCCESSORS(state) **do**

$v \leftarrow$ MAX(v, MIN-VALUE(s, α, β)

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

$\alpha \leftarrow$ MAX(α, v)

return v

Para busca em profundidade limitada Representa a Avaliação estática

Algoritmo Alfa-Beta (cont.)

function MIN-VALUE(*state*, α, β) **returns** *a utility value v*

inputs: ***state***, current state in game

α, the best alternative for MAX along the path to ***state***

β, the best alternative for MIN along the path to ***state***

if TERMINAL-TEST(state) **then return** UTILITY(state)

$v \leftarrow +\infty$

for each *s* in SUCCESSORS(state) **do**

$v \leftarrow$ MIN(v, MAX-VALUE(s, α, β)

if $v \leq \alpha$ **then return** v

$\beta \leftarrow$ MIN(β, v)

return v

Para busca em profundidade limitada Representa a Avaliação estática