

DD2380

Artificial Intelligence

Lecture 6 Games

Patric Jensfelt and Jana Tumova

Introduction to Games

- Some problems are multi-agent
 - Cooperative vs competitive interaction
- Search problems in these environments are called games

Why study games?

- Precise rules
- Limited number of actions
- They are hard problems!
- Example: Chess
 - Average branch factor of 35
 - Typically 50 moves for each player
 - $\rightarrow 35^{100} \approx 10^{154}$ nodes in search tree

Why study games?

- Precise rules
- Limited number of actions
- They are hard problems!
- Example: Chess
 - Average branch factor of 35
 - Typically 50 moves for each player
 - → $35^{100} \approx 10^{154}$ nodes in search tree
- (10^{80} atoms in the observable universe)

Types of games

	Deterministic	Chance
Perfect Information	Chess, Checkers go, othello	Backgammon monopoly
Imperfect Information	battleships	bridge, poker scrabble,

Perfect information = fully observable

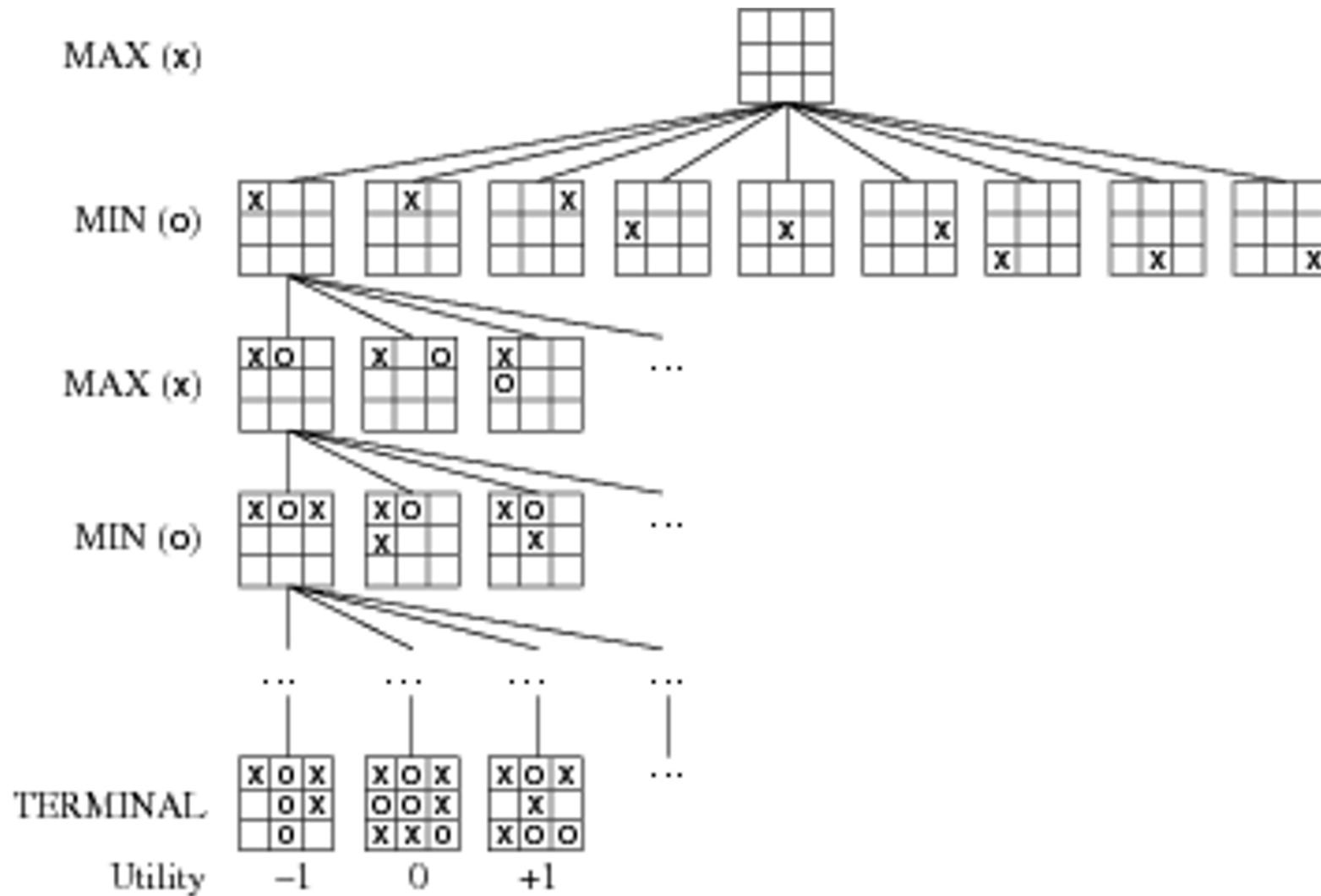
Today

- Turn-taking, two-player, **zero-sum** games
 - Gain of winner same as loss of loser

Formalizing the problem: MAX and MIN

- Two players MAX and MIN
- An initial state (the start situation)
- A successor function (move, state)
- A terminal state - when is the game over?
- A utility function gives value for terminal state typically (+1,-1,0)
- (aka objective or payoff function)
- The progression can be modeled as a (game-) tree

Game tree (2-player, deterministic, turn-based)

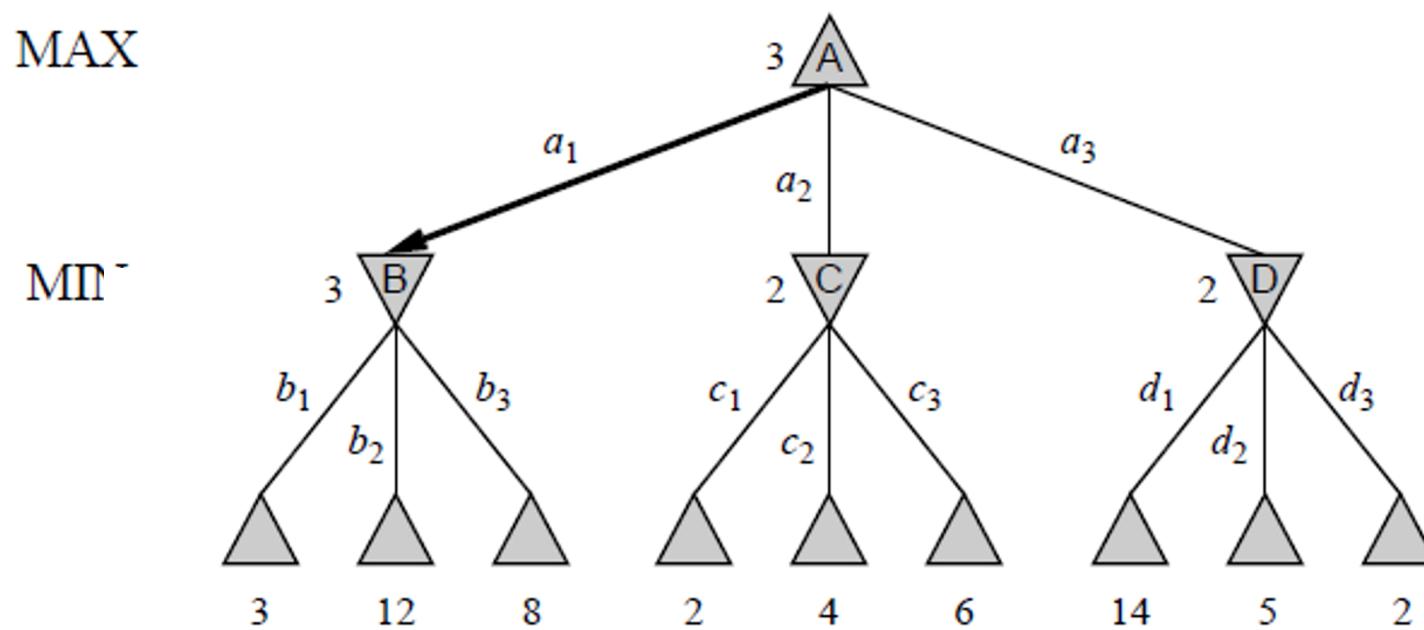


Optimal strategies

- In “normal search”: reach the goal configuration
- In a game (adversarial search): not enough, MAX needs to reach the goal configuration regardless of what MIN does
- Optimal strategy: at least as good as any other strategy when playing against an infallible MIN

Q

- Consider a 2-ply (half-move) game tree
 - MAX has actions a_1, a_2, a_3 and MIN has actions b_1, b_2, b_3



- What is the best strategy for MAX?

The minimax algorithm

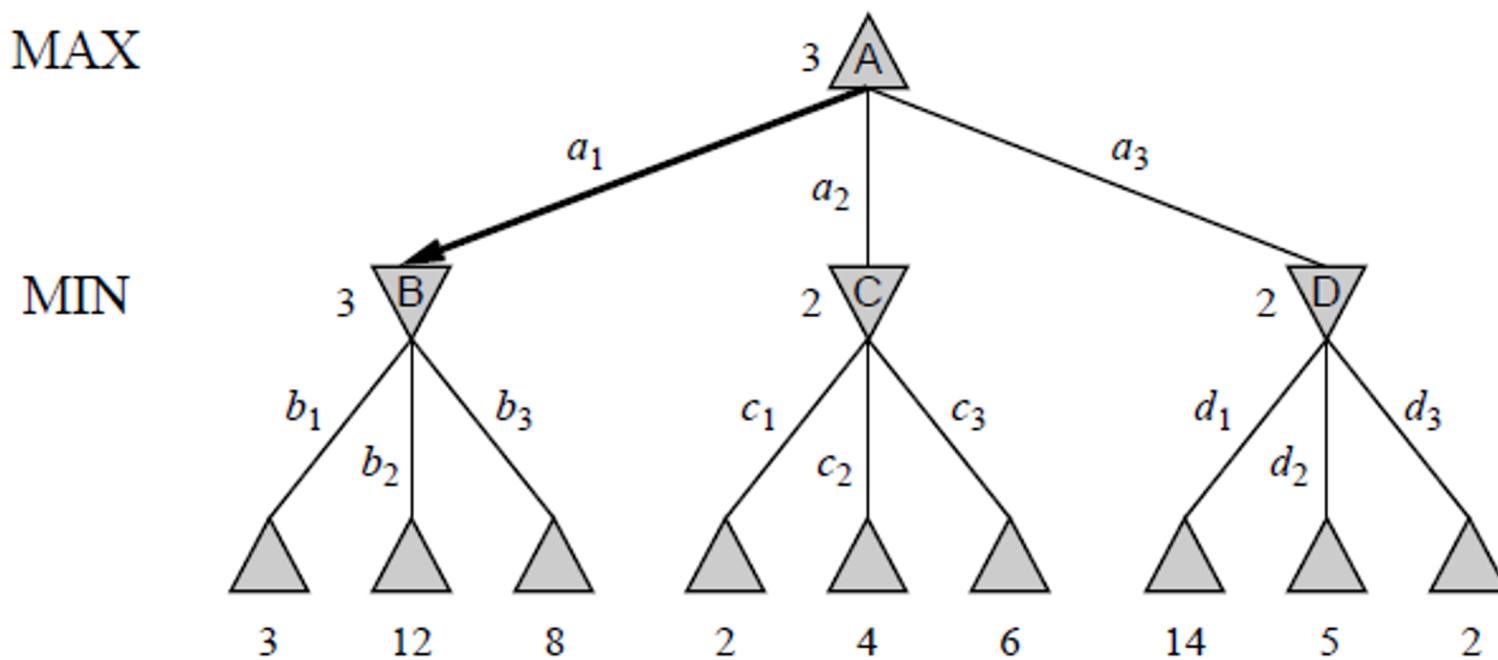
- Assumes that the agents are rational, self-interested
- The strategy of best achievable payoff against best play by opponent
-

$\text{MINIMAX}(s) =$

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

The best strategy

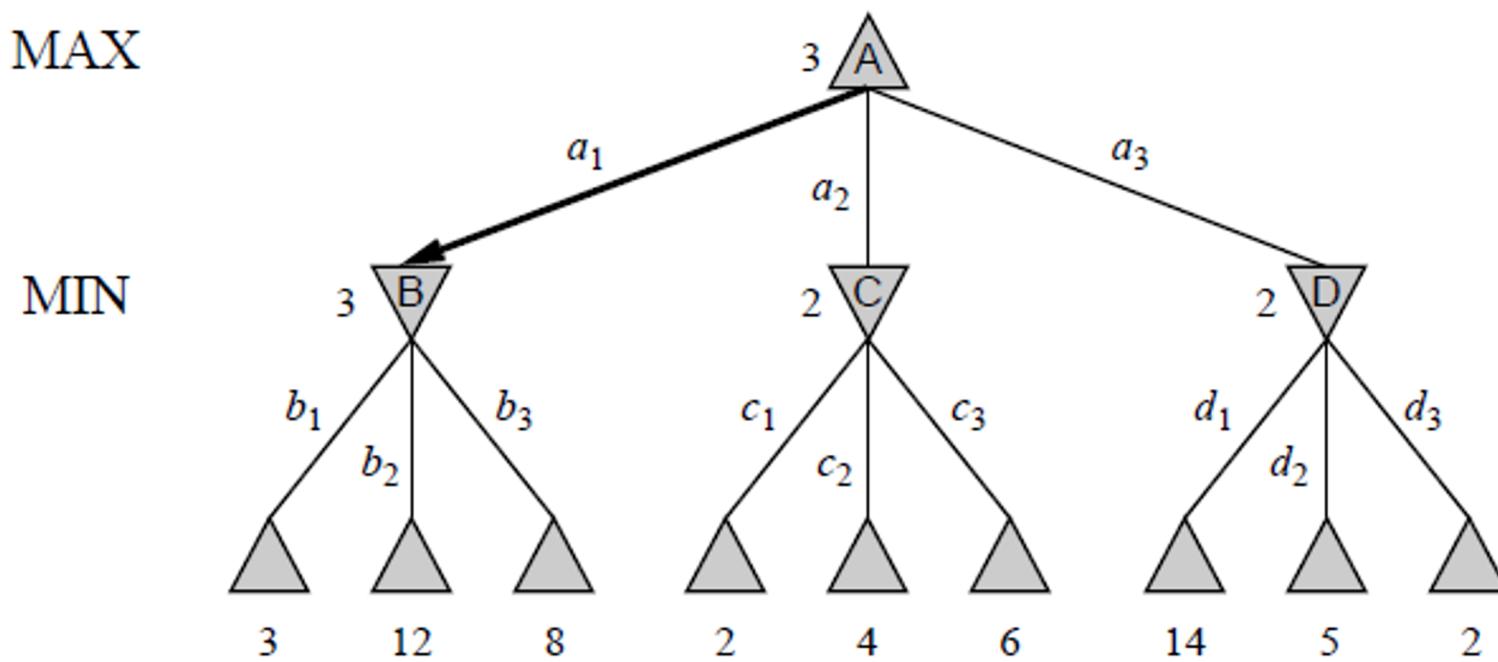
- If MAX picks a_1 MIN will pick $b_1 \rightarrow 3$, if MAX picks a_2 MIN will pick $c_1 \rightarrow 2$, a_3 will give $d_3 \rightarrow 2$



- Best to pick a_1 as it gives utility 3

The best strategy

- Q: What if MIN is not infallible, is it still the best strategy?



Recursion

- Make sure you know how to work with recursive functions

Minimax algorithm

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

```
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
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    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
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

Properties of minimax

- Complexity $O(b^m)$

Optimal strategies

- In “normal search”: reach the goal configuration
- In a game (adversarial search): not enough, MAX needs to reach the goal configuration regardless of what MIN does
- Optimal strategy: at least as good as any other strategy when playing against an infallible MIN
- **Analysis of the full game tree can be demanding or impossible**

Repeated states

- Transposition causes repeated states (among others)
 - Many different moves result in same configuration
 - Example: chess
- Store evaluation in transposition (hash) table

Handling resource limitations

- Do not use real utility but a heuristic evaluation function that estimates the value of a particular strategy/position
 - Should order the terminal states the same as the utility function
 - Should be fast to compute
 - Should reflect chances to win
- Chess: number of white pawns, black pawns, white queens, black queens, etc.
- Cutoff test instead of terminal test
 - Limited depth search

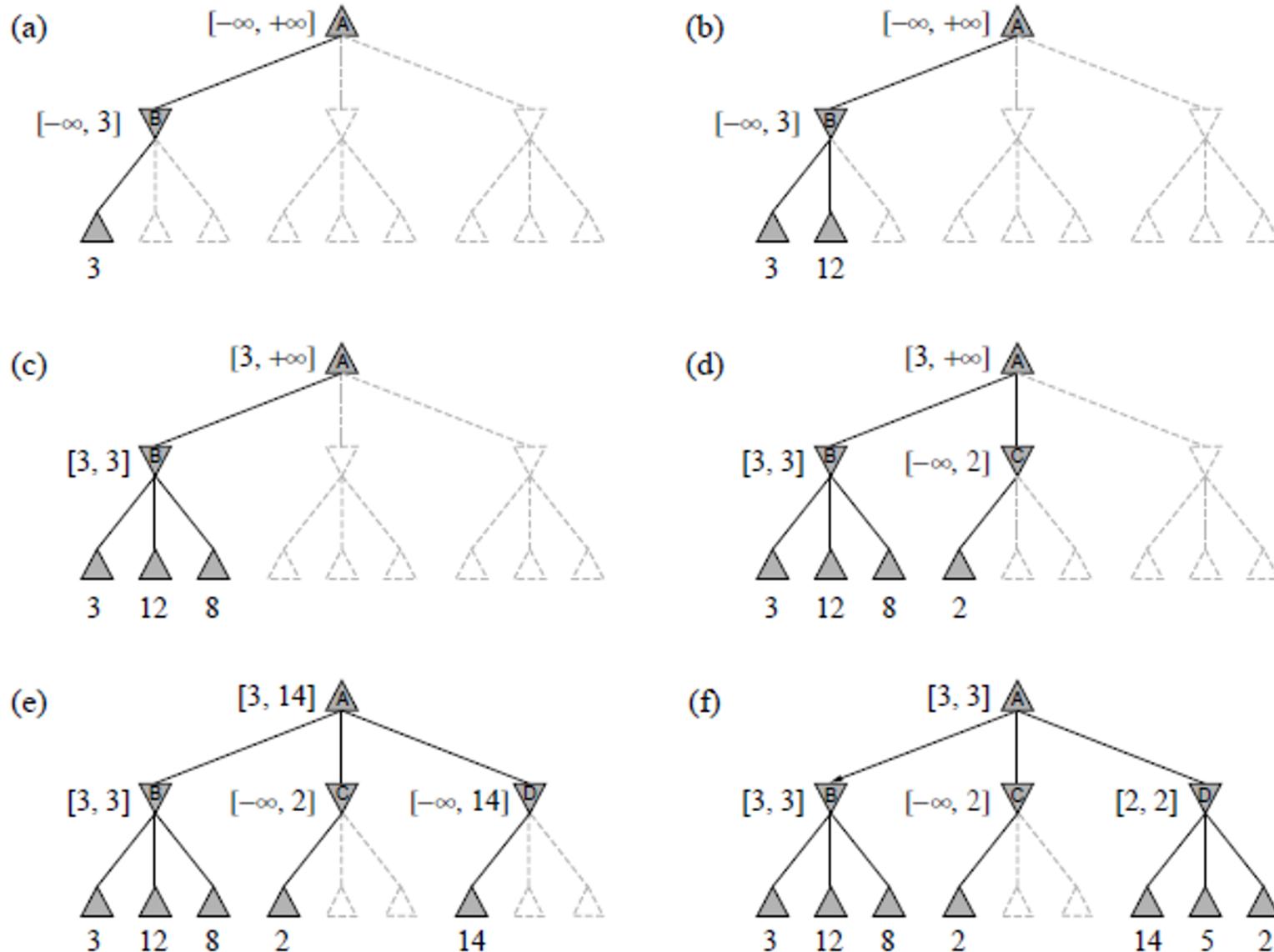
a- β pruning

- Do we really need to analyze all nodes?
- Stop investigate unrealistic plays as early as possible, i.e. prune the tree
- Cuts down computations, but still gives the same Minimax result

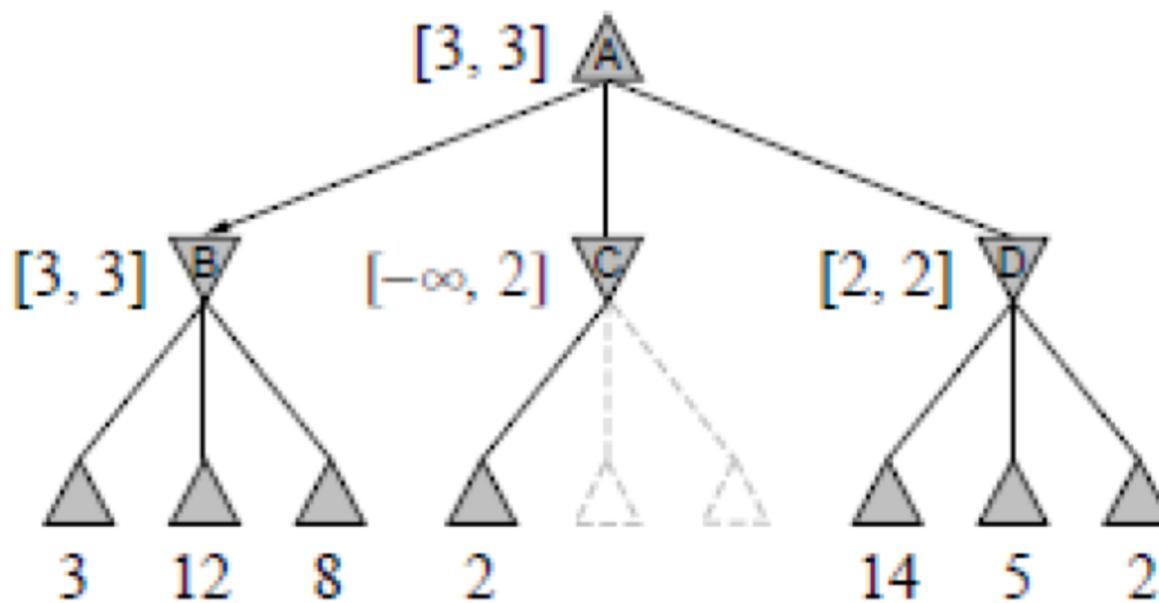
a- β pruning

- α : the value of the best choice for MAX
- β : the value of the best choice for MIN
- found along the path till now

Re-analyze the 2-ply game tree



Re-analyze the 2-ply game tree


$$\begin{aligned}\text{MINIMAX}(\text{root}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \quad \text{where } z = \min(2, x, y) \leq 2 \\ &= 3.\end{aligned}$$

Minimax algorithm

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

```
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
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    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
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
    return the 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  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
        if v  $\geq \beta$  then return v
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
    return v
```

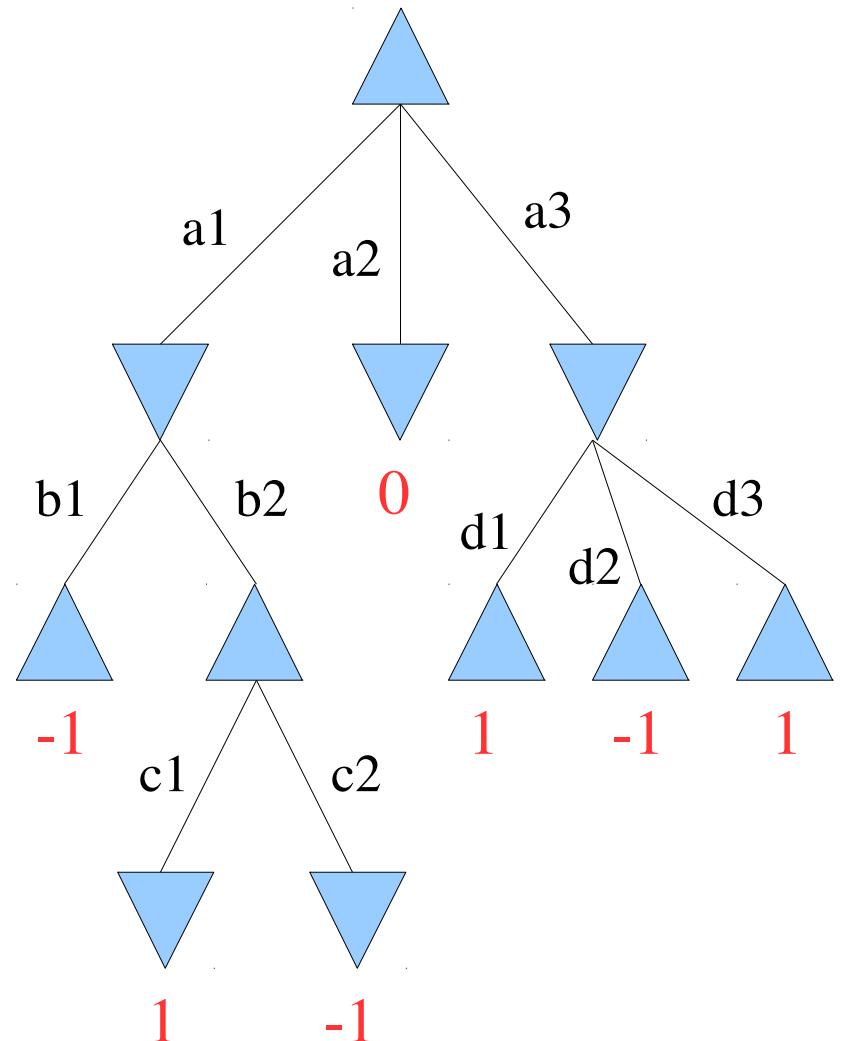
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow +\infty$ 
    for each a in ACTIONS(state) do
        v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
        if v  $\leq \alpha$  then return v
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
    return v
```

a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty, +\infty$ )
  return the 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),  $\alpha, \beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha, \beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

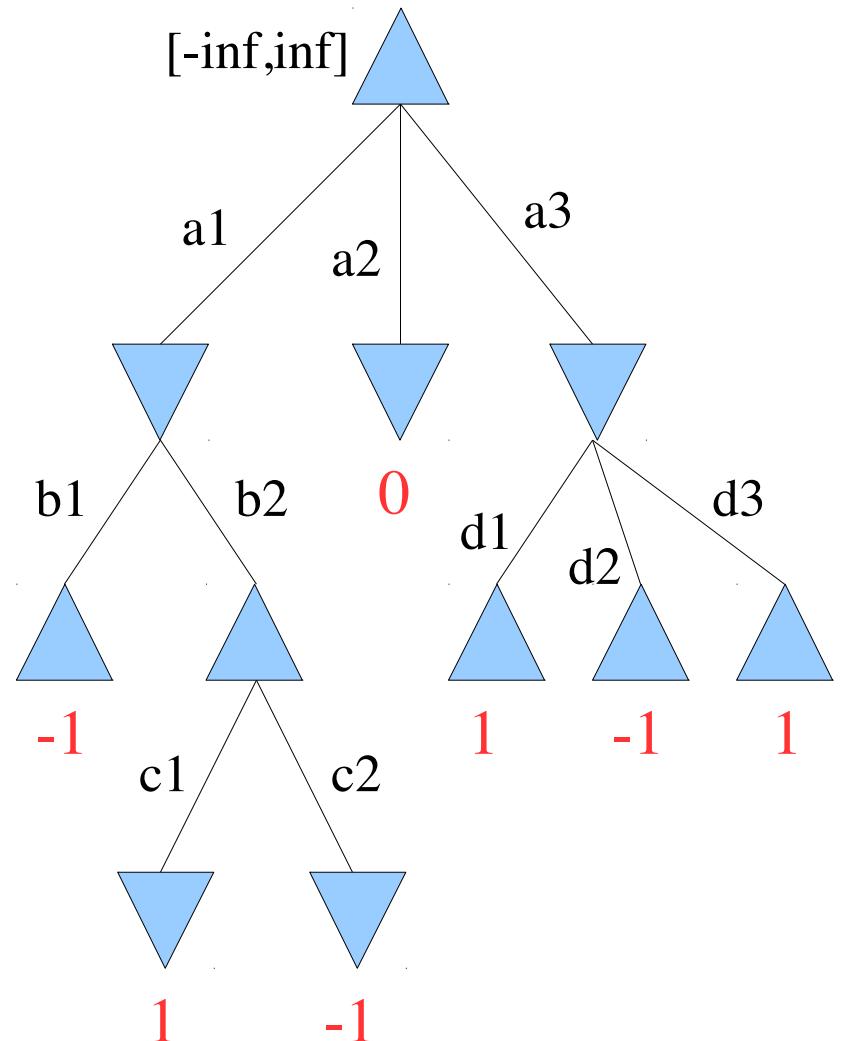


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

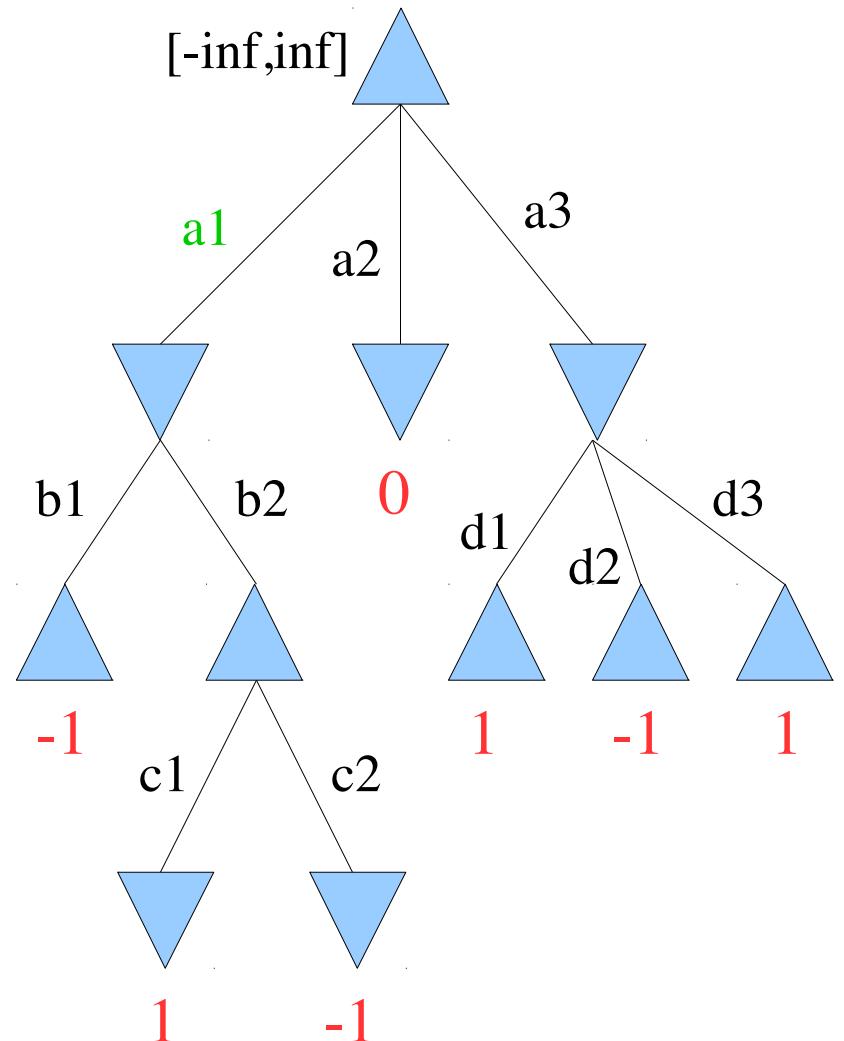


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

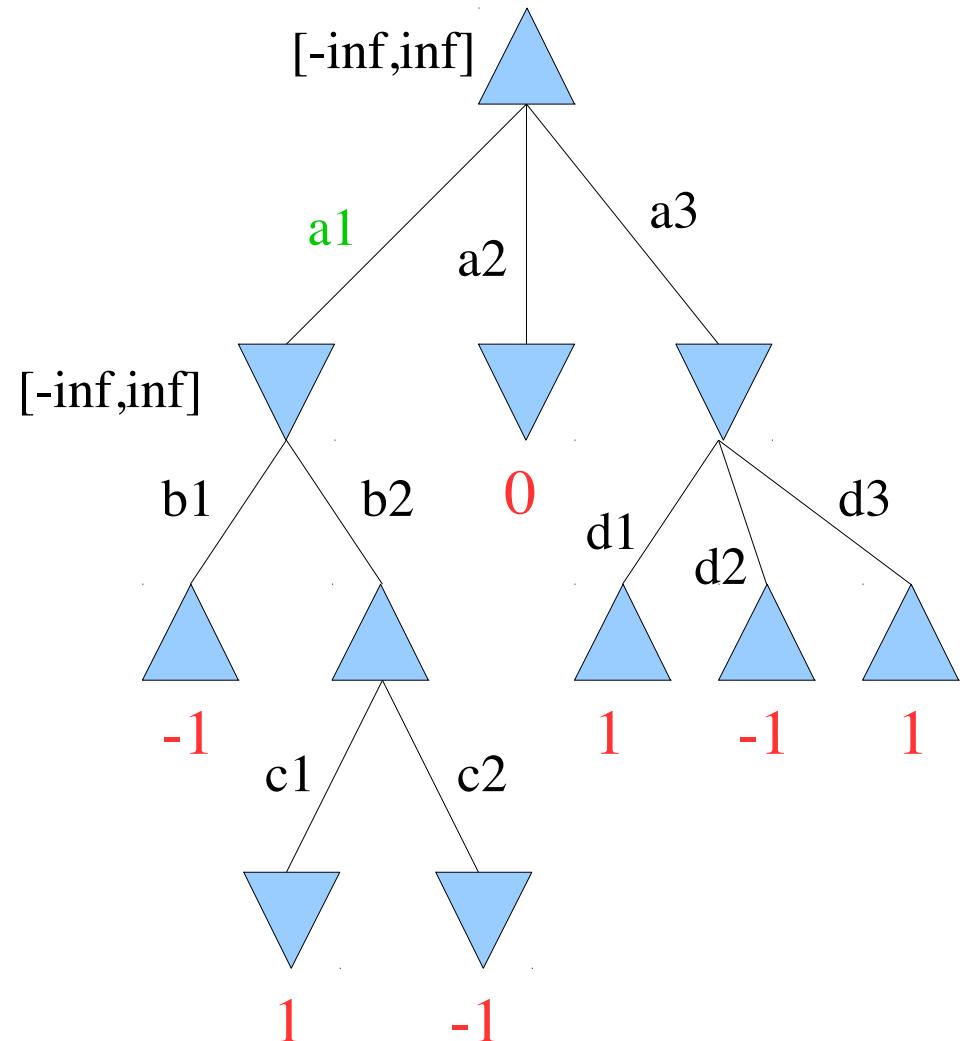


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

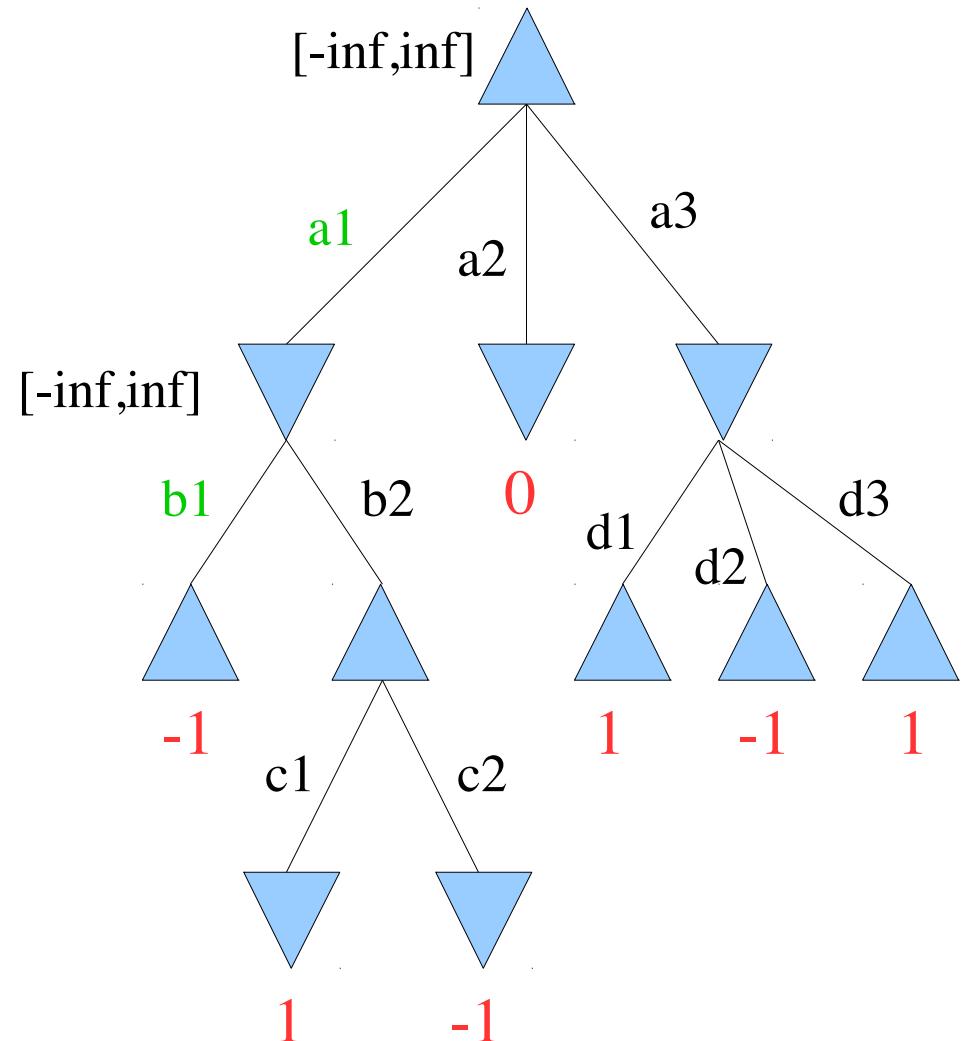


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty, +\infty$ )
  return the 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),  $\alpha, \beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha, \beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

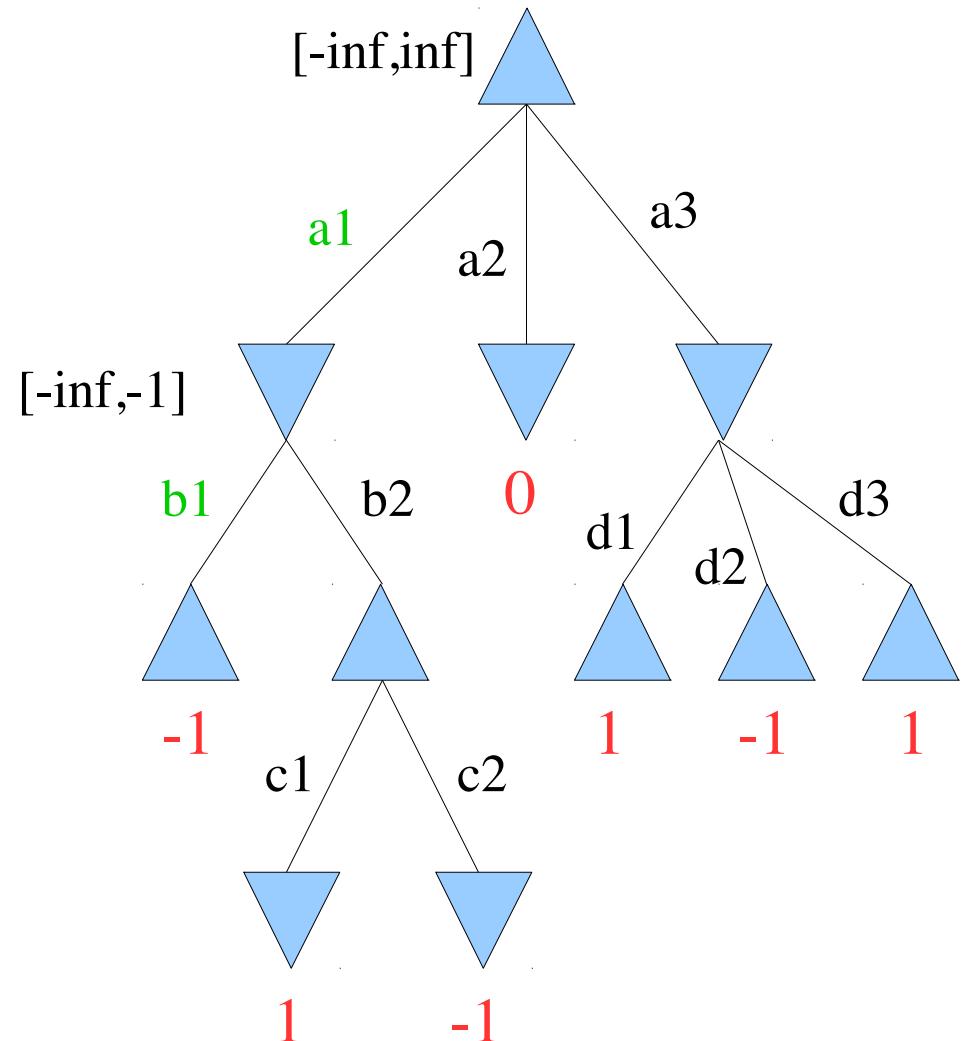


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty, +\infty$ )
  return the 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),  $\alpha, \beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha, \beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

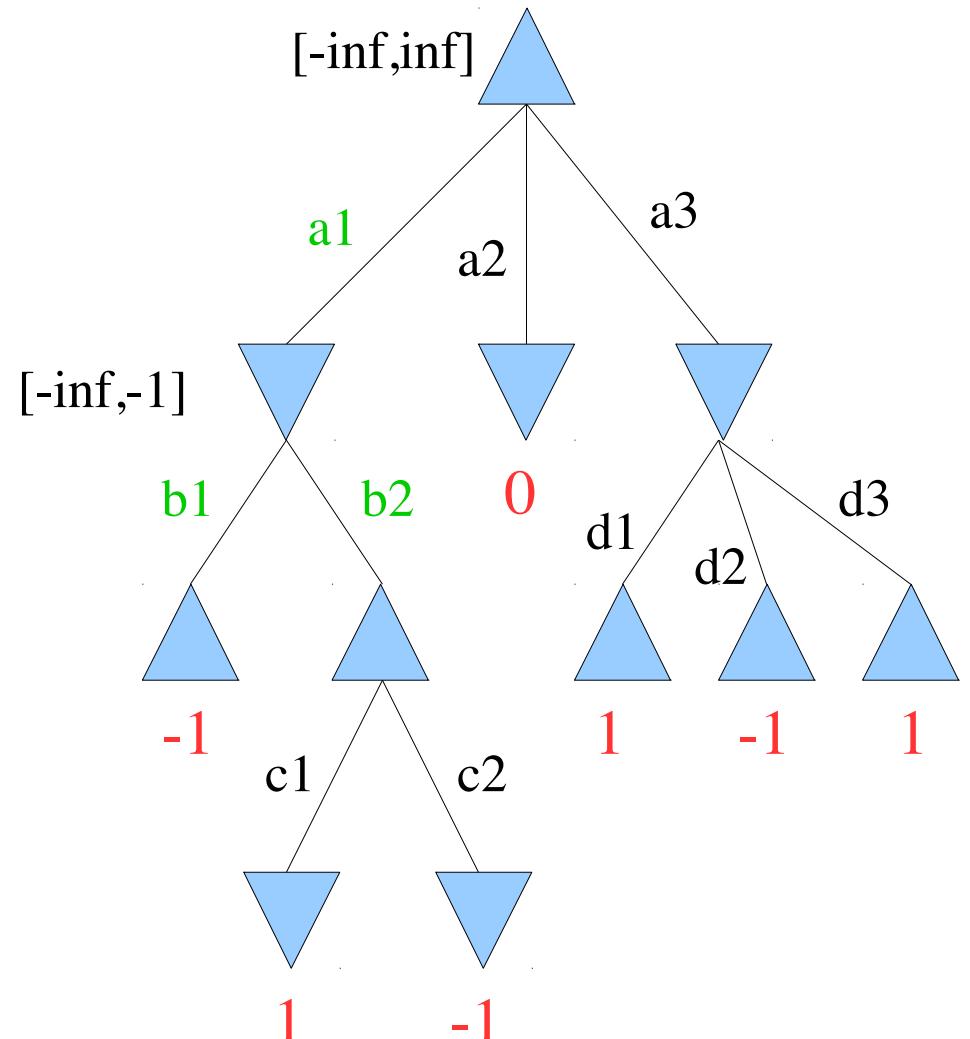


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

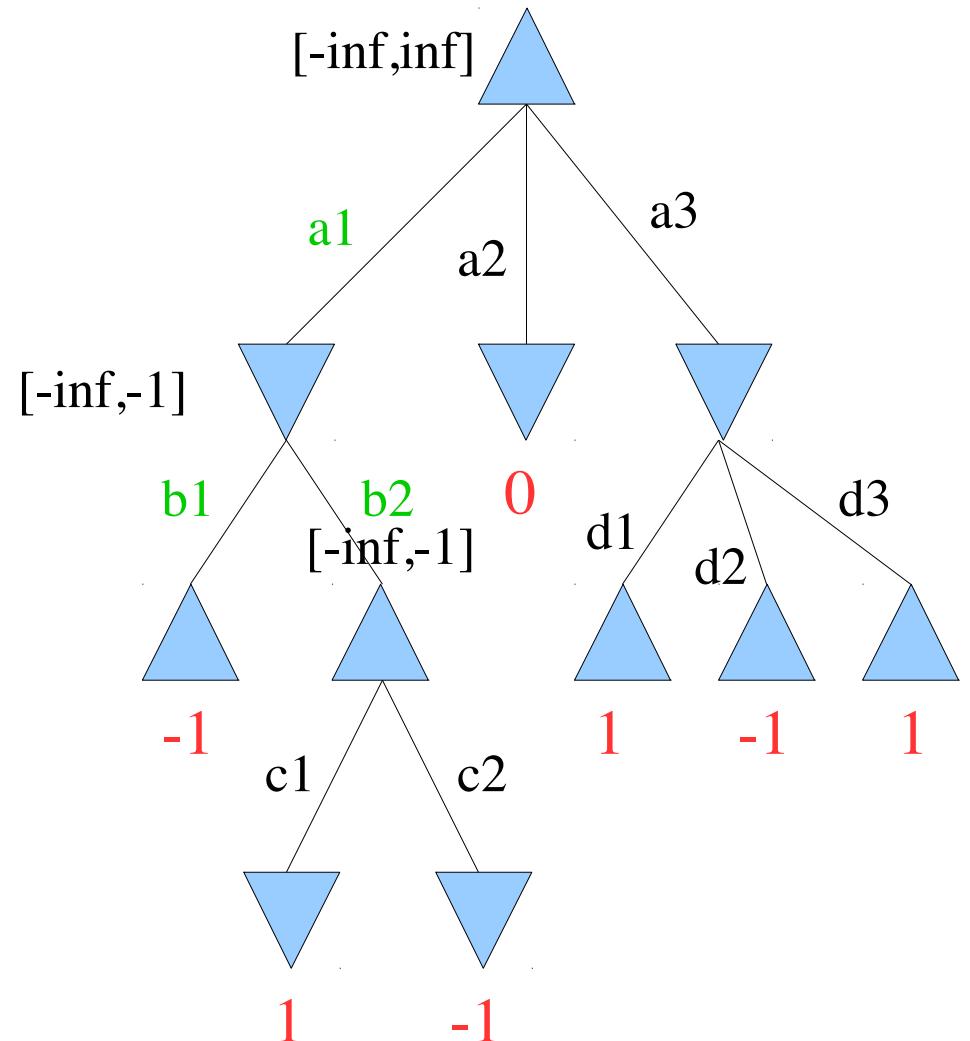


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

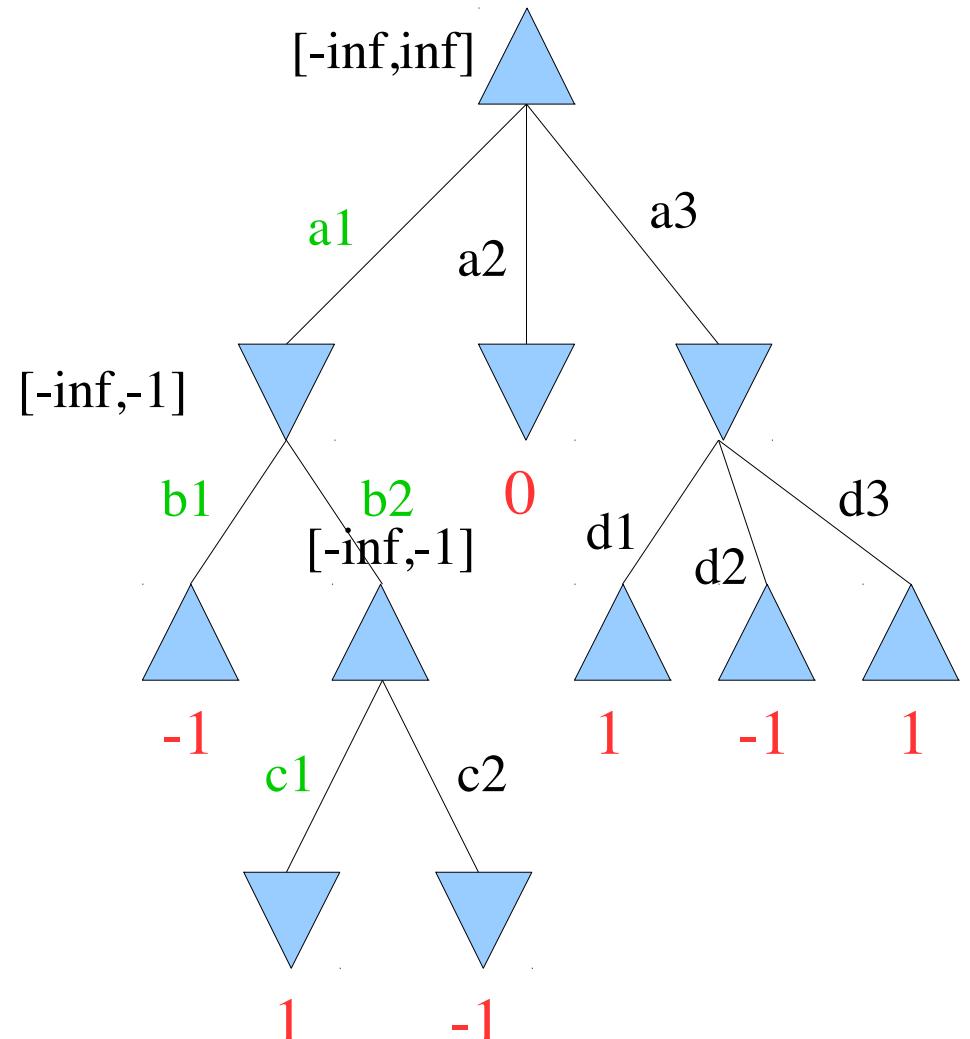


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

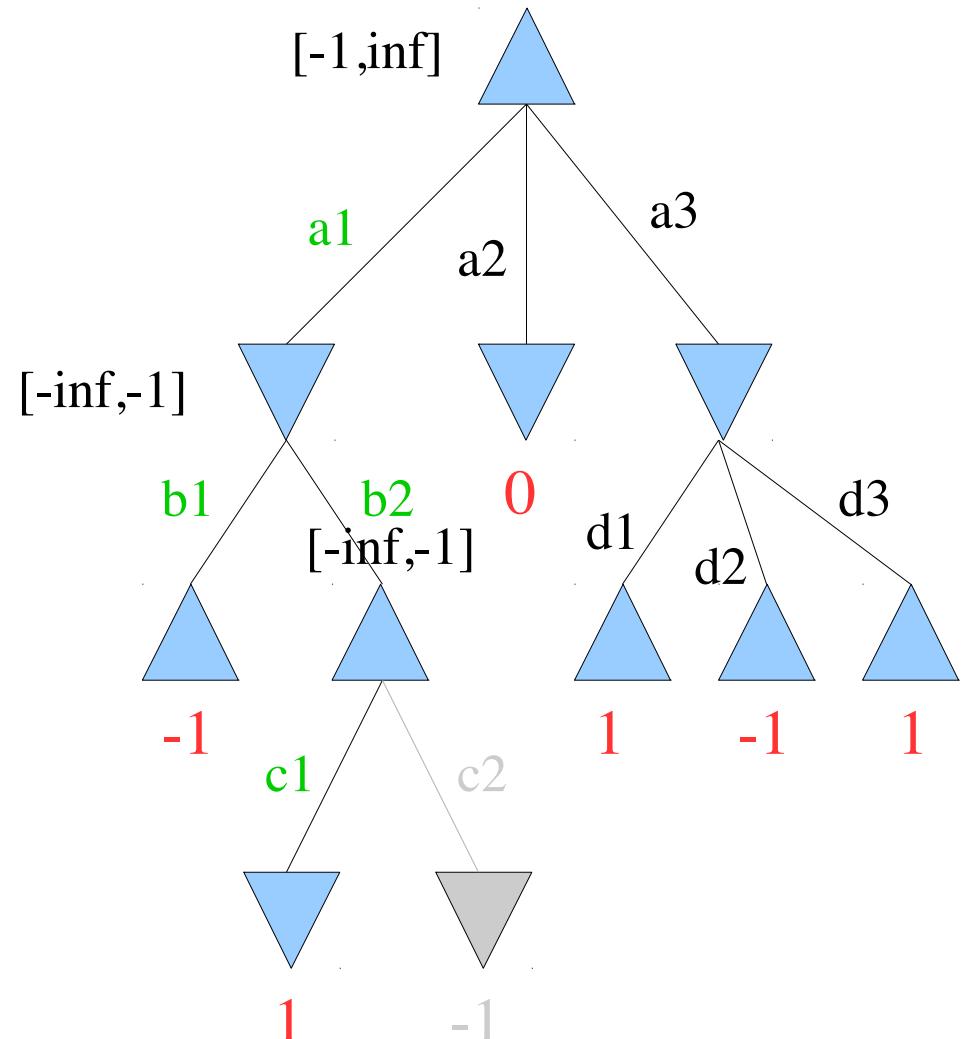


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
  return the 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
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    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
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

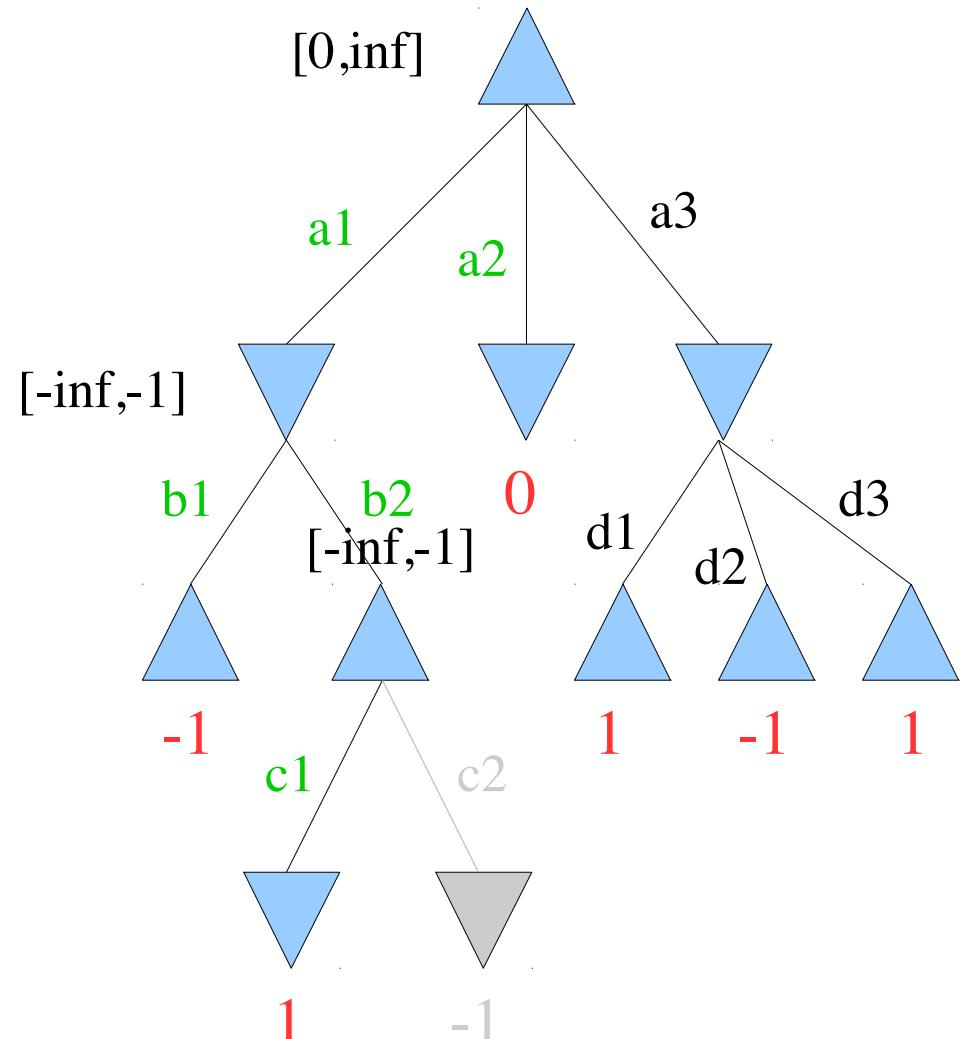


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

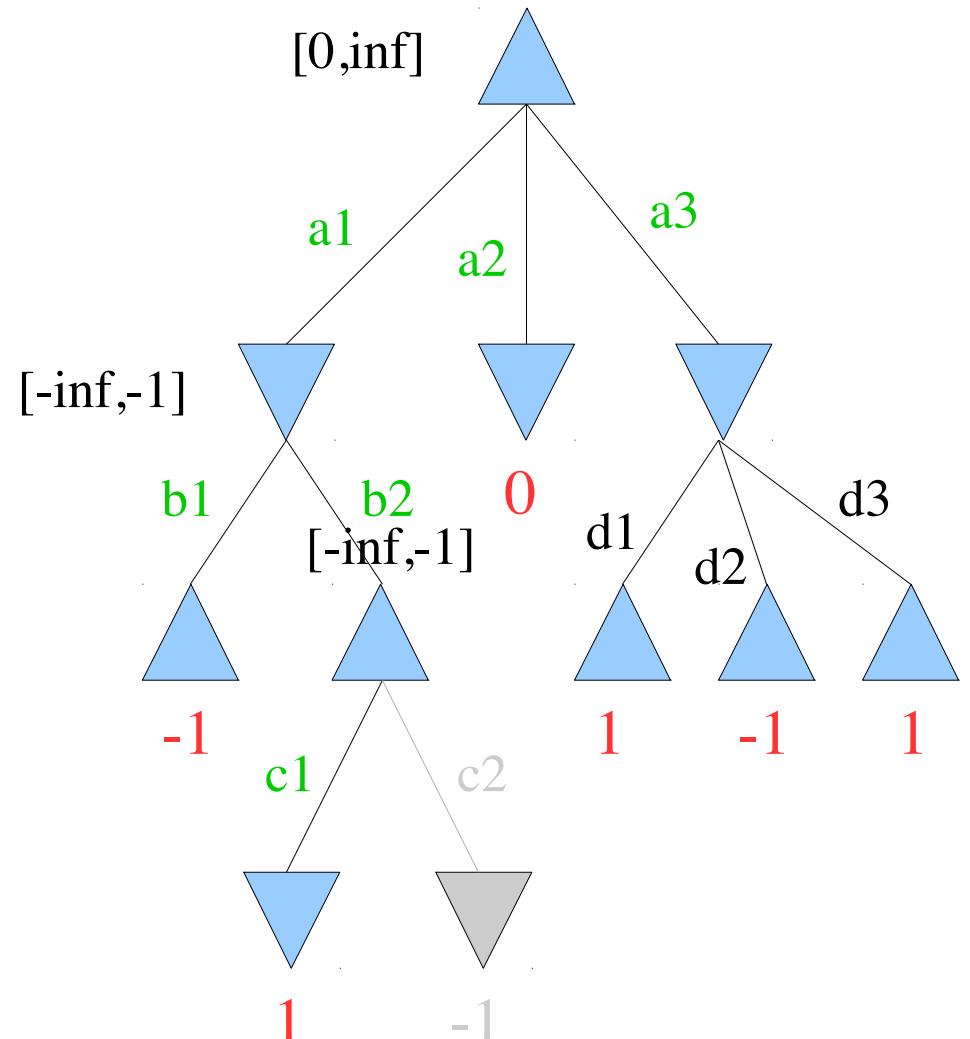


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

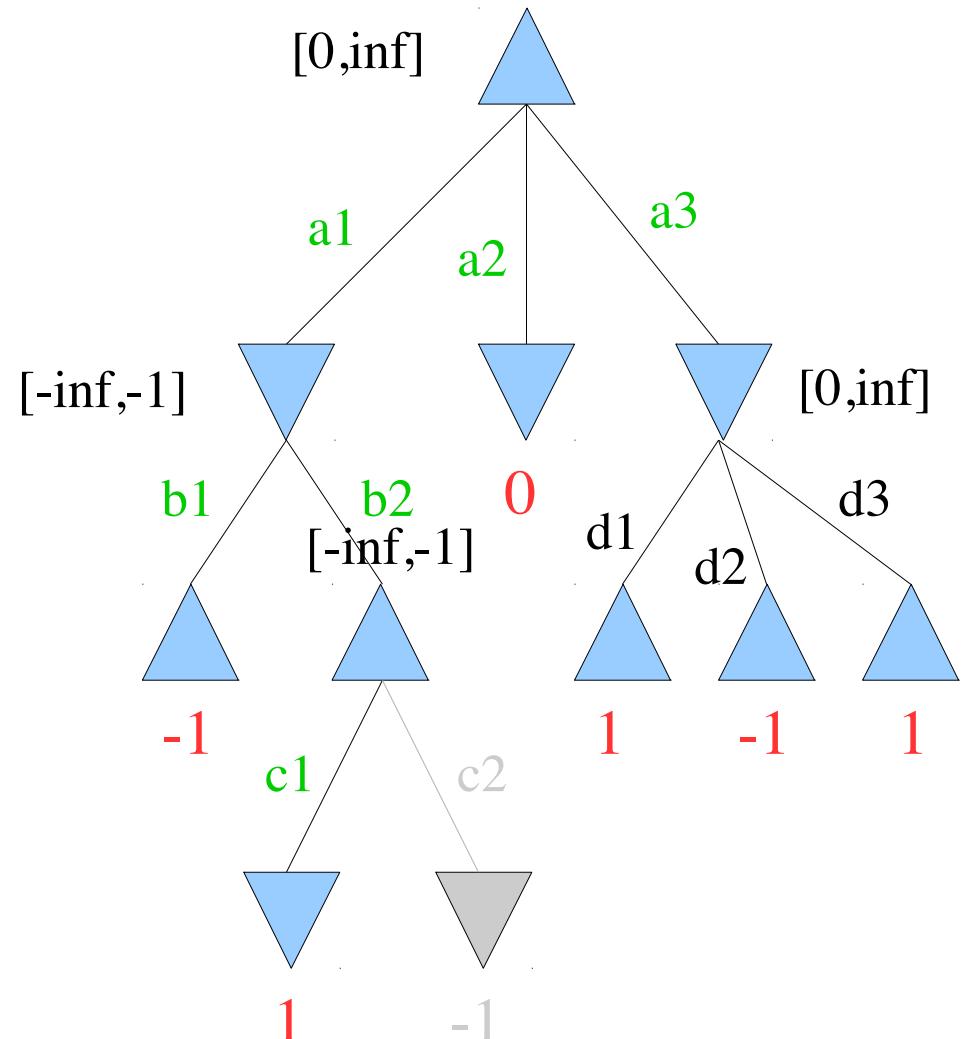


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

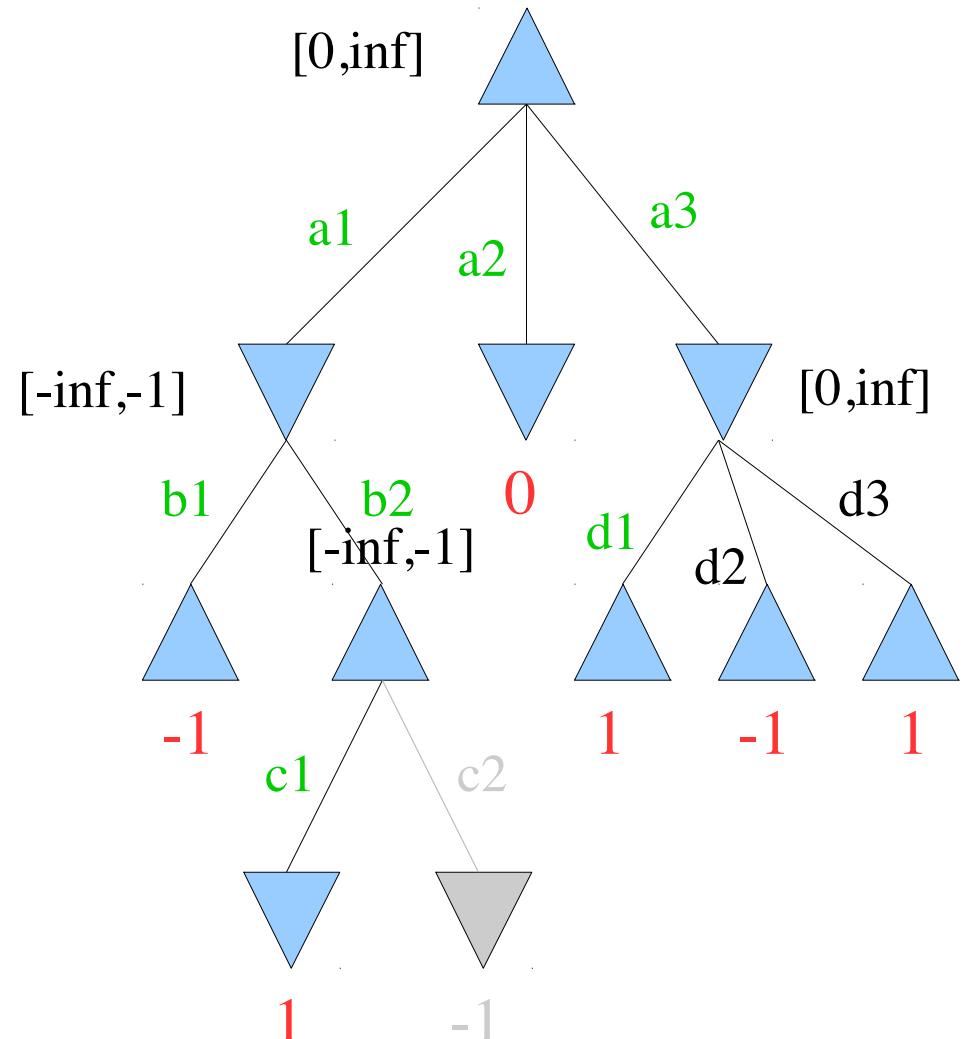


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

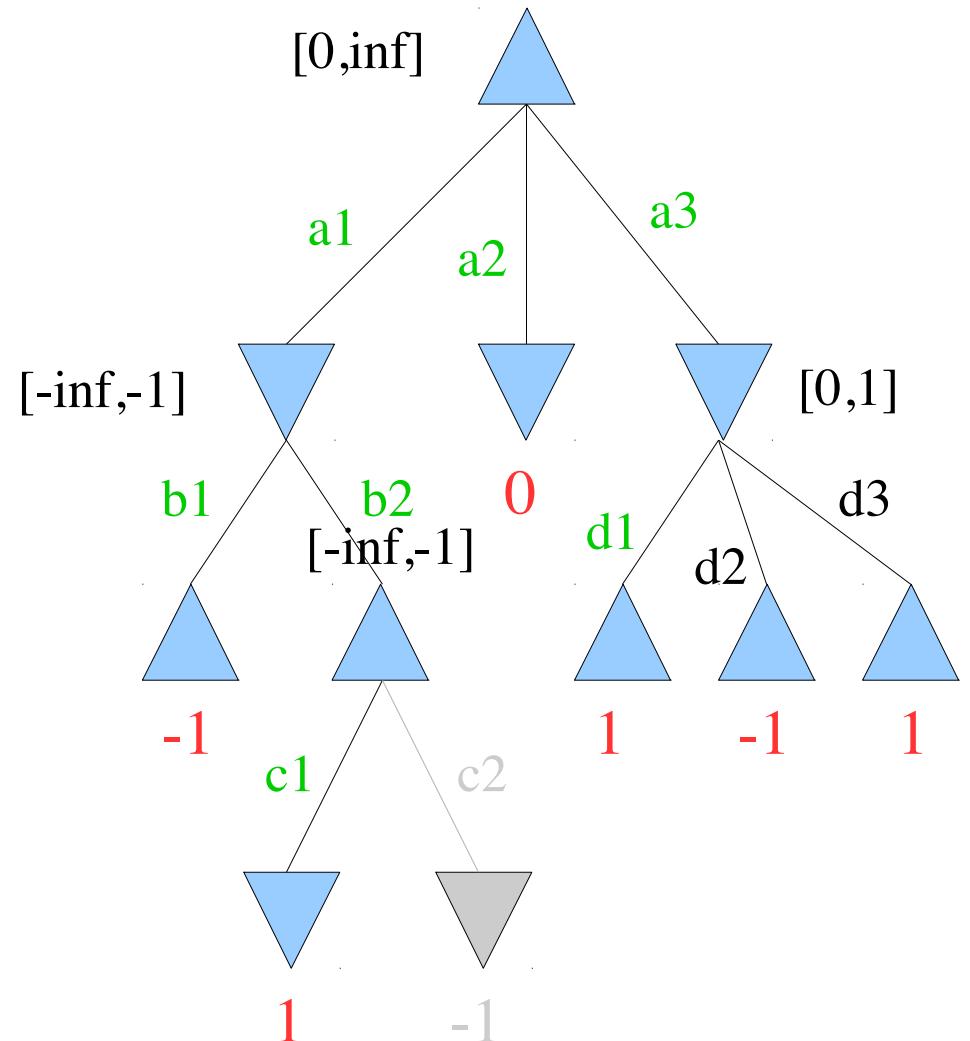


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

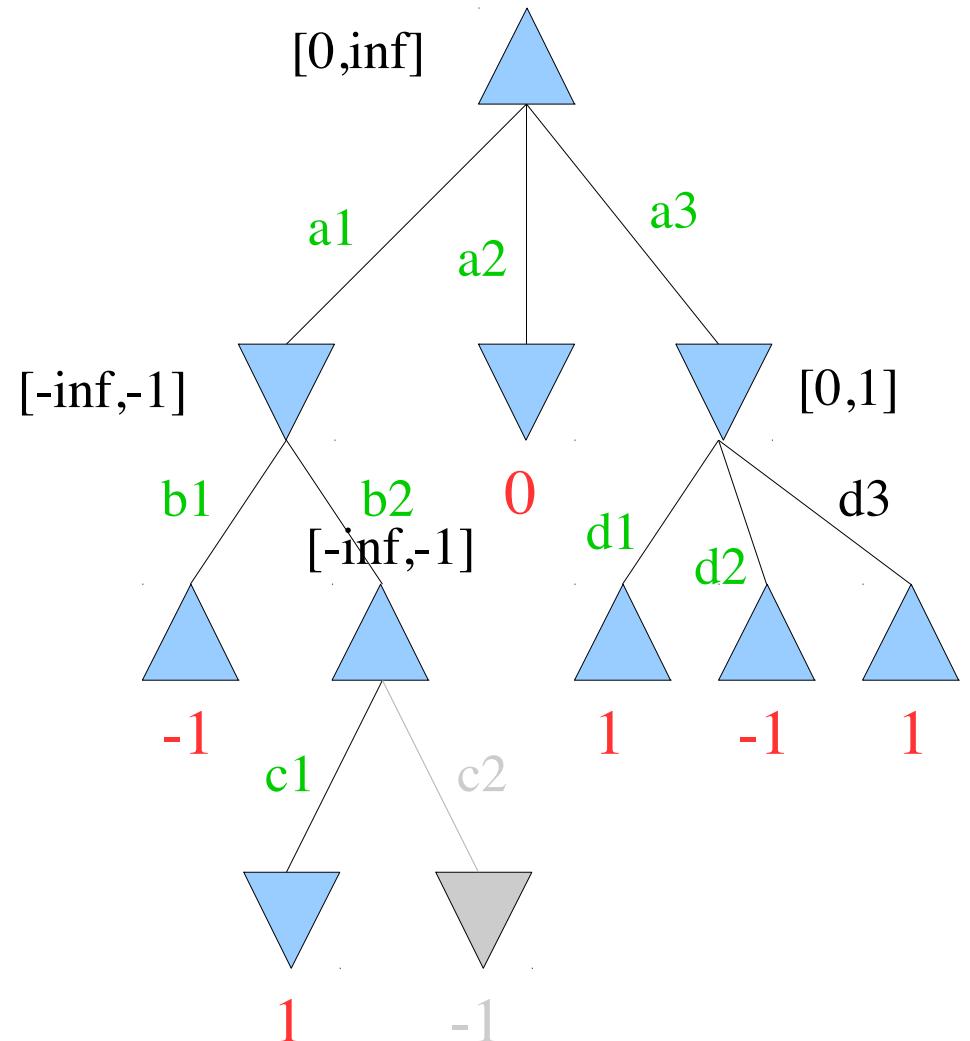


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

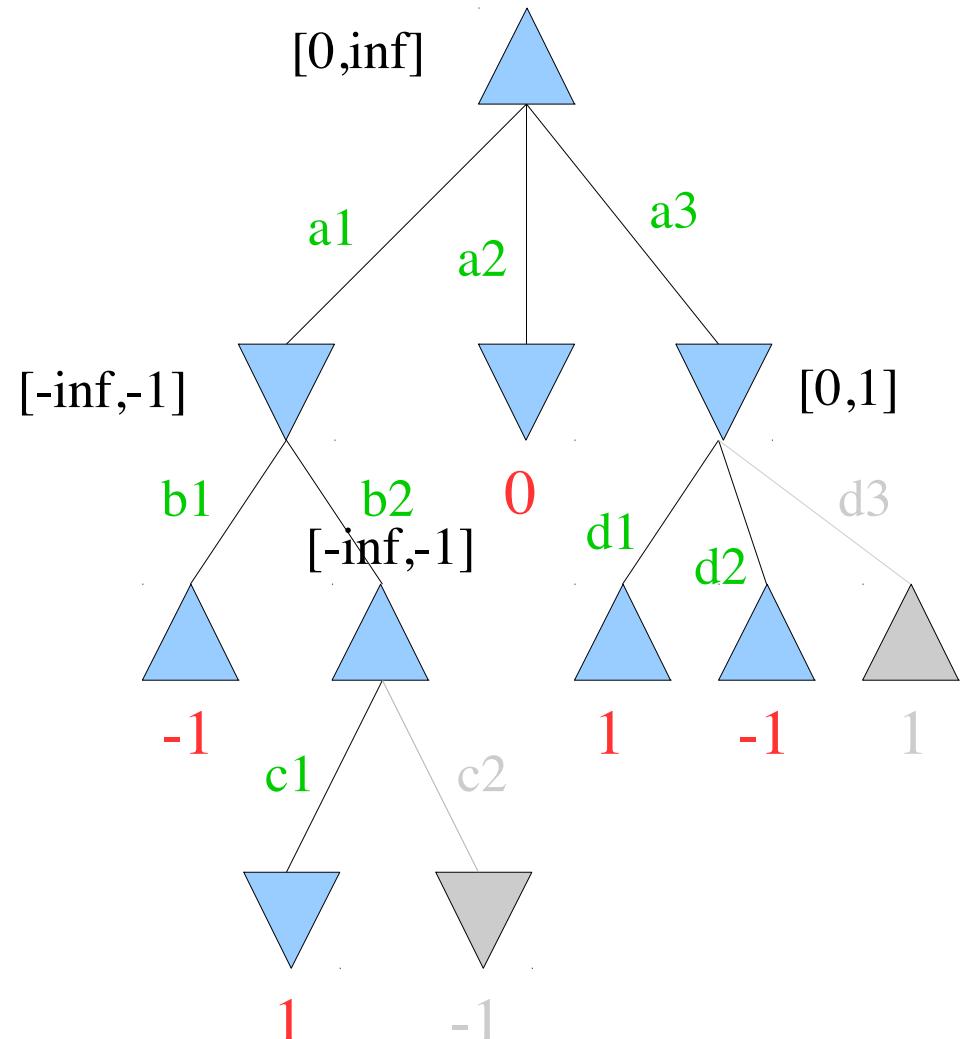


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the 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),  $\alpha$ ,  $\beta$ ))
    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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

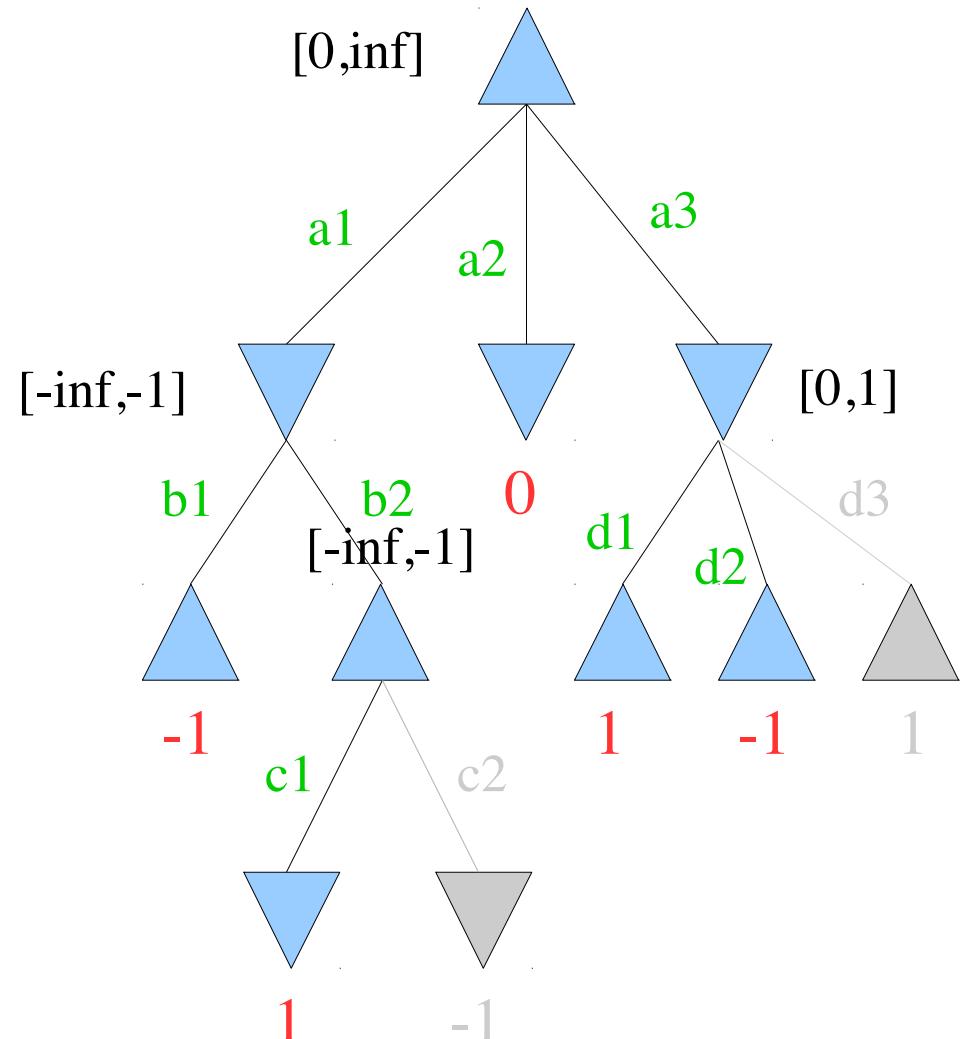


a- β pruning algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the action in ACTIONS(state) with value v
```

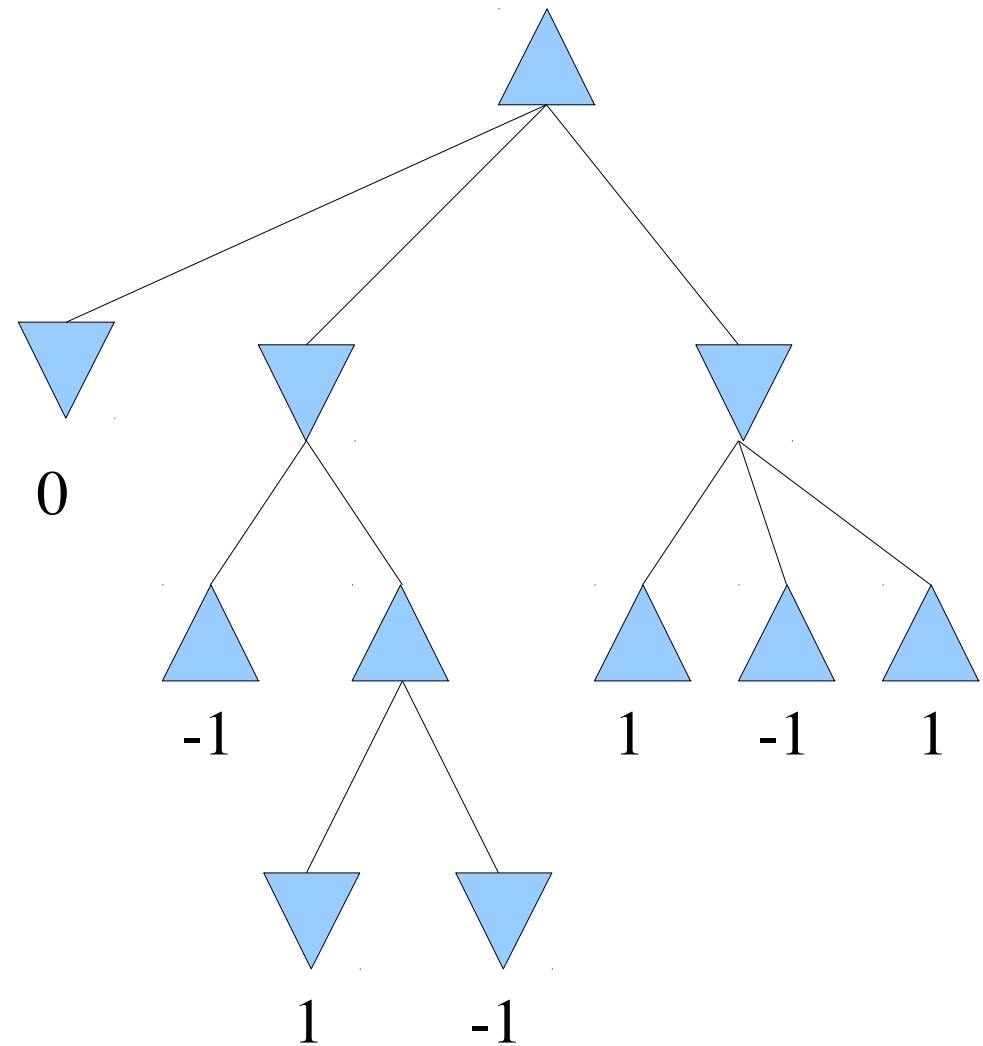
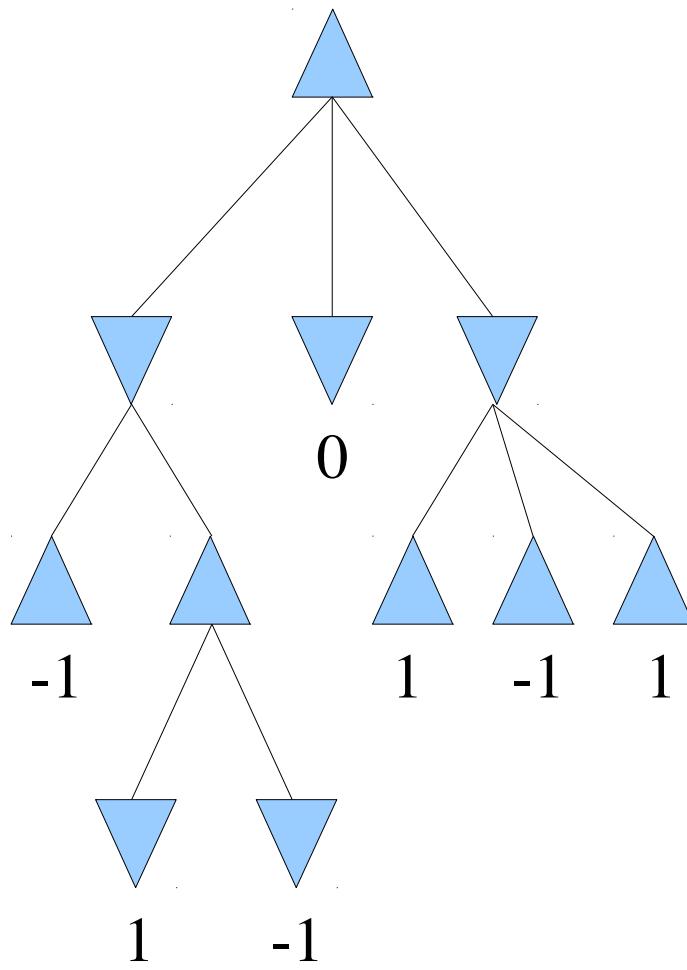
```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow -\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\geq \beta$  then return 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  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```



a- β pruning algorithm

- Order matters



Properties of α - β pruning

- Complexity - depends heavily on order of evaluation.
- With perfect ordering $O(b^{m/2})$
- → Can then look twice as far into the game as minimax in the same time

Reordering in a- β pruning

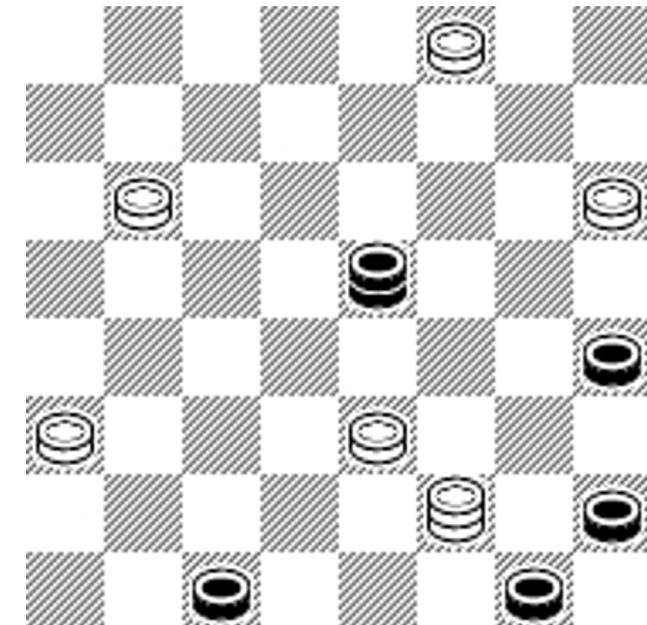
- Try likely the best first
- Chess: captures, threats, forward moves, backward moves
- Estimate through limited dfs
- Repeat killer moves

Deterministic games: Chess

- DeepBlue defeated human world champion Gary Kasparov in a six game match 1997.
- DeepBlue searched 200 million positions/s.
- Up to depth 40 in search.

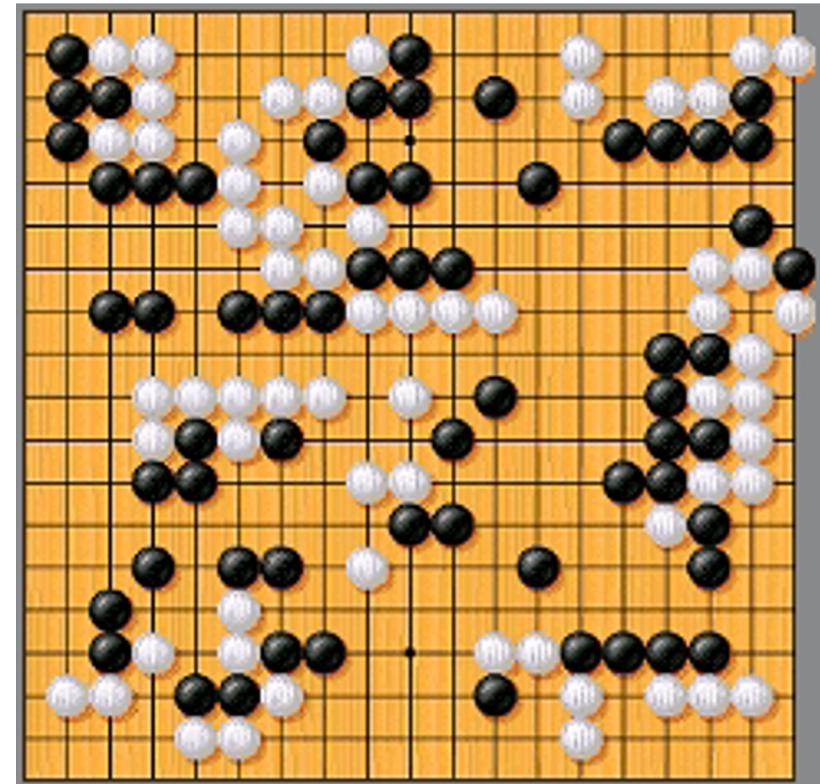
Deterministic games: Checkers

- “Solved” in 2007
- 18 years of computing on a cluster with 15+ machines
 - endgame table (≤ 10 pieces)
- → 39 trillion entries
- search+end game+computer → success
- Your A-B-level assignment!



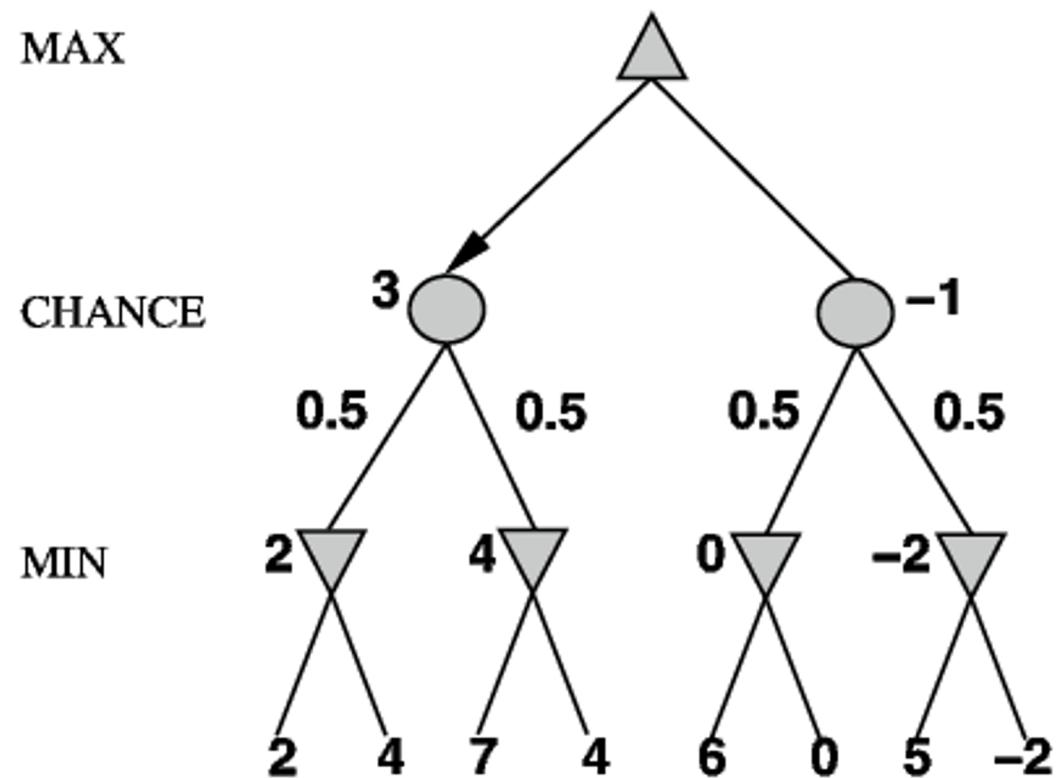
Deterministic games: Go

- Enormous branch factor ($b > 300$)
- AlphaGo – a slightly different principle



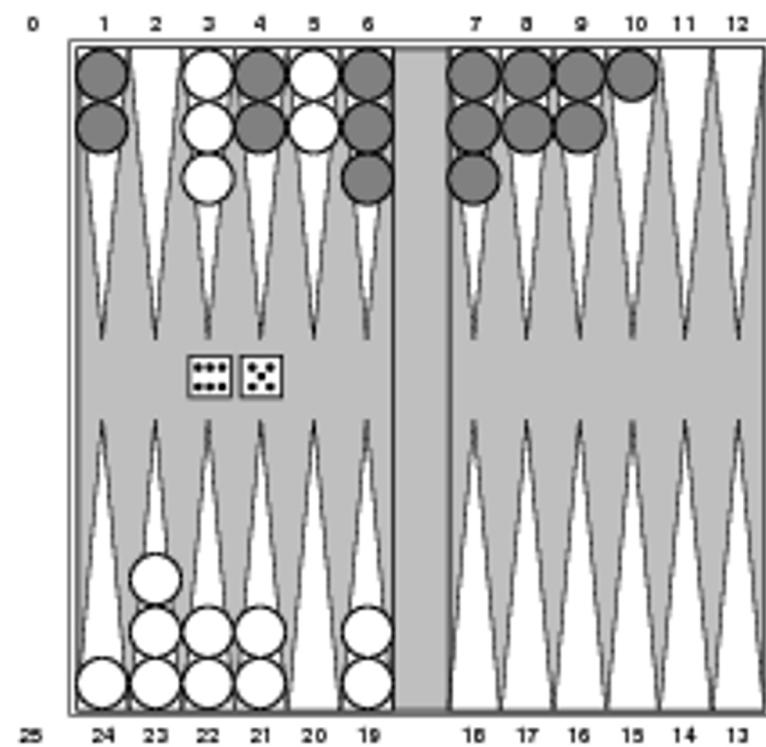
Non-deterministic games

- In nondeterministic games an element of chance is introduced
 - e.g. a coin-flip
- Expectiminimax



Non-deterministic games: Backgammon

- First game where a computer defeated a human world champion.



Games with imperfect information

- E.g., card games, where opponent's initial cards are unknown
- Typically one can compute a probability for each possible deal
- Idea
 - compute the minimax value of each action in each deal
 - choose the action with highest expected value over all deals

Summary games

- Competitive environment
- Four components: initial state, actions, terminal test, and utility function
- Perfect information allow use of minimax
→ pruning to optimize search
- Use of evaluation functions to allow depth limited search
- With an element of chance the expected value can be used similar to minimax (expectimax)
- Today computers are as good or better than humans at many different games