Introduction to Artificial Intelligence

Game Playing

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Wintersemester 2003/2004

Outline

- **Perfect play**
- **Resource limits**
- α-β **pruning**
- **Games of chance**
- **Games of imperfect information**

Games vs. Search Problems

Game playing is a search problem

Defined by

- **Initial state**
- **Successor function**
- **Goal test**
- **Path cost / utility / payoff function**

Games vs. Search Problems

Game playing is a search problem

Defined by

- **Initial state**
- **Successor function**
- **Goal test**
- **Path cost / utility / payoff function**

Characteristics of game playing

- **“Unpredictable” opponent:**
  Solution is a strategy specifying a move for every possible opponent reply
- **Time limits:**
  Unlikely to find goal, must approximate

Game Playing

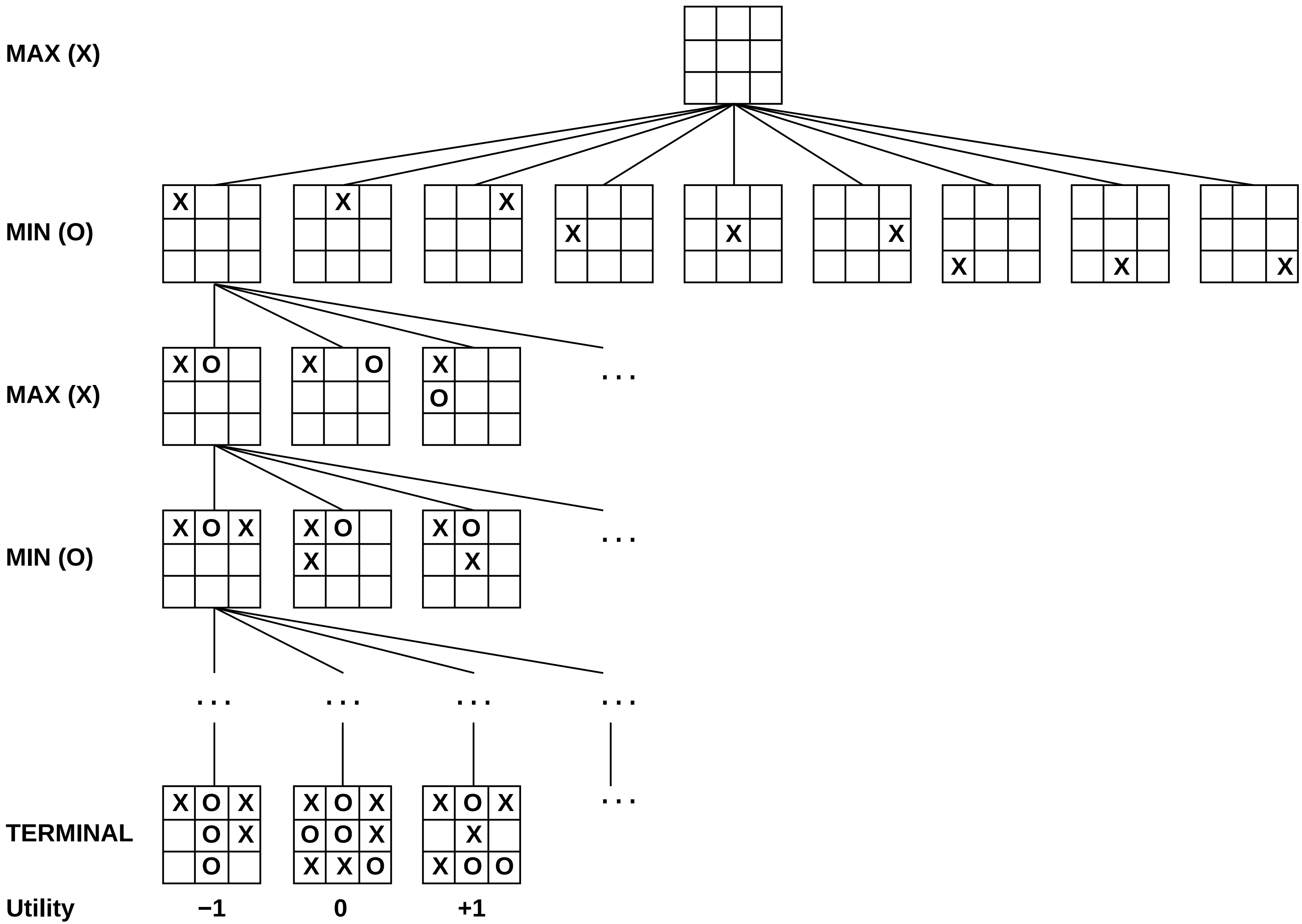
Plan of attack

- Computer considers possible lines of play [Babbage, 1846]
- Algorithm for perfect play [Zermelo, 1912; Von Neumann, 1944]
- Finite horizon, approximate evaluation [Zuse, 1945; Wiener, 1948; Shannon, 1950]
- First chess program [Turing, 1951]
- Machine learning to improve evaluation accuracy [Samuel, 1952–57]
- Pruning to allow deeper search [McCarthy, 1956]

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Game Tree: 2-Player / Deterministic / Turns



Minimax

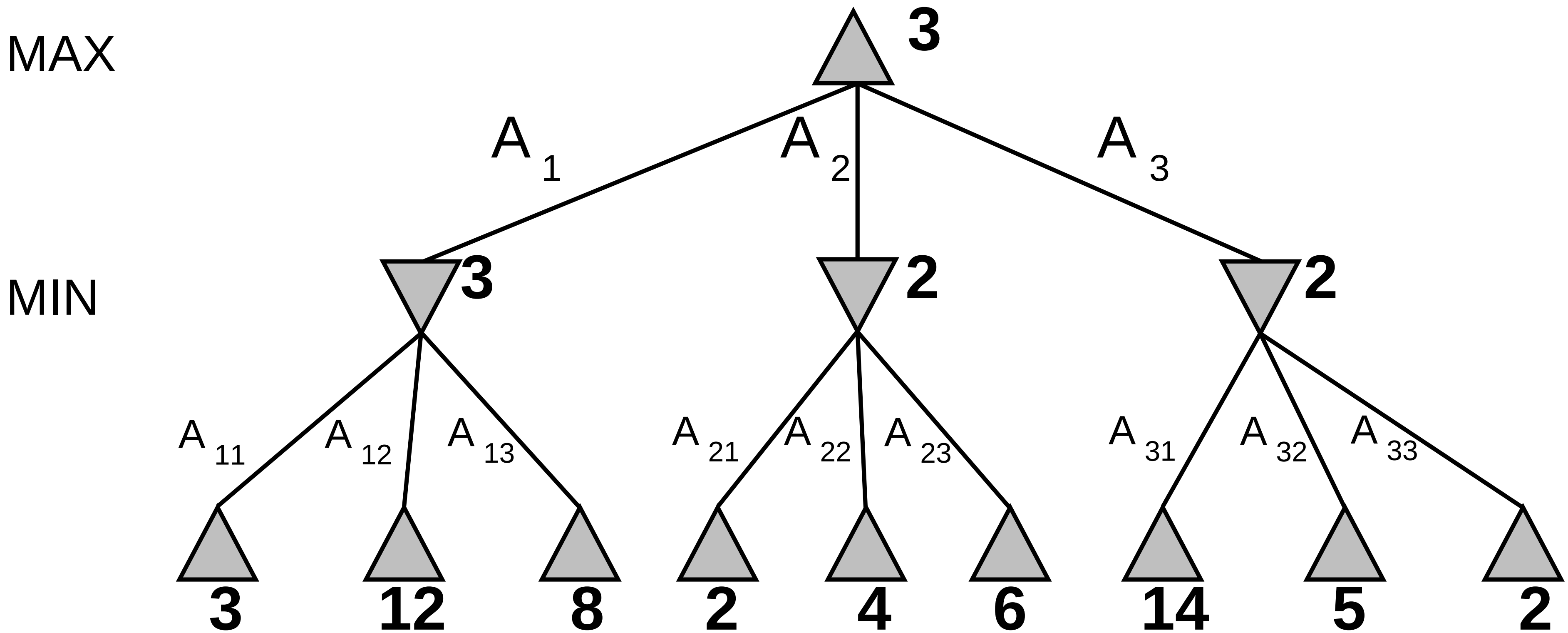
Perfect play for deterministic, perfect-information games

Idea

Choose move to position with highest minimax value,
i.e., best achievable payoff against best play

Minimax: Example

2-ply game



Minimax Algorithm

function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**

    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)

  end

  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state, game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**

    return UTILITY[*game*](*state*)

  else if MAX is to move in *state* **then**

    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)

  else

    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Properties of Minimax

Complete

Optimal

Time

Space

Properties of Minimax

Complete	**Yes, if tree is finite (chess has specific rules for this)**
Optimal	
Time	
Space	

Properties of Minimax

Complete	Yes, if tree is finite (chess has specific rules for this)
Optimal	Yes, against an optimal opponent. Otherwise??
Time	
Space	

Properties of Minimax

Complete Yes, if tree is finite (chess has specific rules for this)

Optimal Yes, against an optimal opponent. Otherwise??

Time $O(b^m)$ (depth-first exploration)

Space

Properties of Minimax

Complete	**Yes, if tree is finite (chess has specific rules for this)**
Optimal	**Yes, against an optimal opponent. Otherwise??**
Time	$O(b^m)$ **(depth-first exploration)**
Space	$O(bm)$ **(depth-first exploration)**

Properties of Minimax

Complete Yes, if tree is finite (chess has specific rules for this)

Optimal Yes, against an optimal opponent. Otherwise??

Time $O(b^m)$ (depth-first exploration)

Space $O(bm)$ (depth-first exploration)

Note

Finite strategy can exist even in an infinite tree

Resource Limits

Complexity of chess

$b \approx 35$, $m \approx 100$ for “reasonable” games

Exact solution completely infeasible

Resource Limits

Complexity of chess

$b \approx 35$, $m \approx 100$ for "reasonable" games

Exact solution completely infeasible

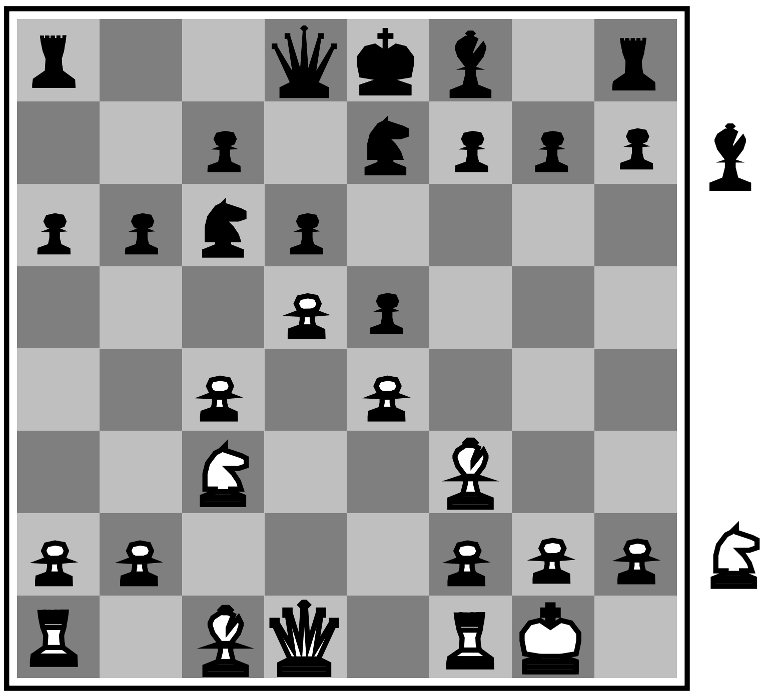
Standard approach

- **Cutoff test**
  e.g., depth limit (perhaps add quiescence search)

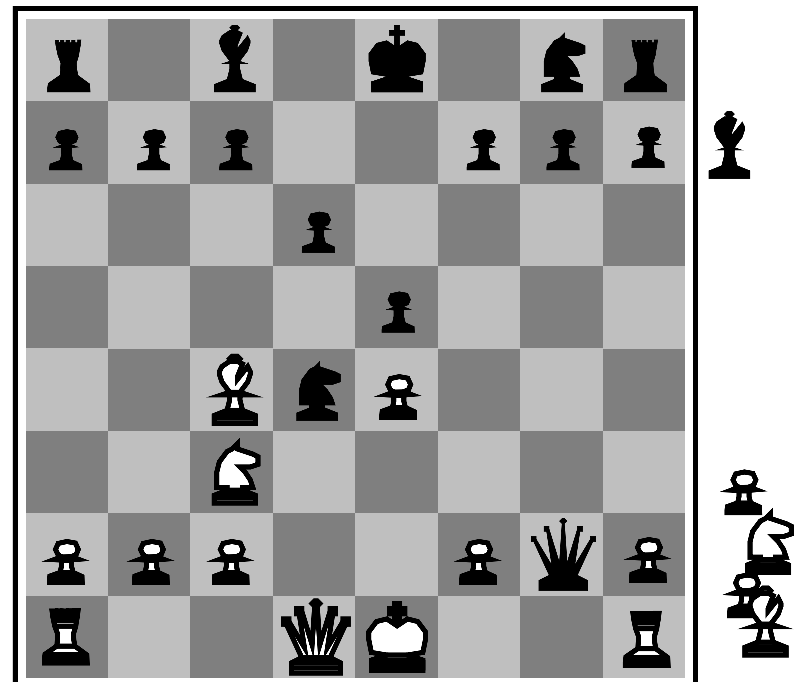
- **Evaluation function**
  Estimates desirability of position

Evaluation Functions

Estimate desirability of position



Black to move
White slightly better



White to move
Black winning

Evaluation Functions

Typical evaluation function for chess

Weighted sum of features

$$\textsc{Eval}(s) \quad = \quad w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

Evaluation Functions

Typical evaluation function for chess

Weighted sum of features

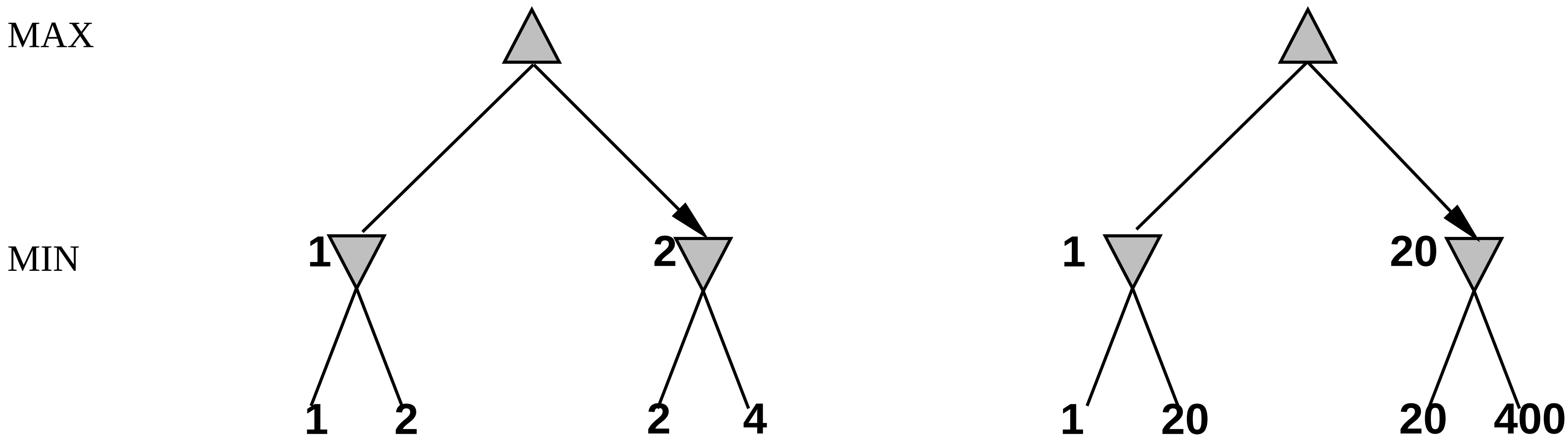
$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

Example

$$w_1 = 9$$

$f_1(s)$ = **(number of white queens) − (number of black queens)**

Digression: Exact Values Do Not Matter



- **Behaviour is preserved under any monotonic transformation of EVAL**
- **Only the order matters:**
  payoff in deterministic games acts as an ordinal utility function

Cutting Off Search

Does it work in practice?

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

Cutting Off Search

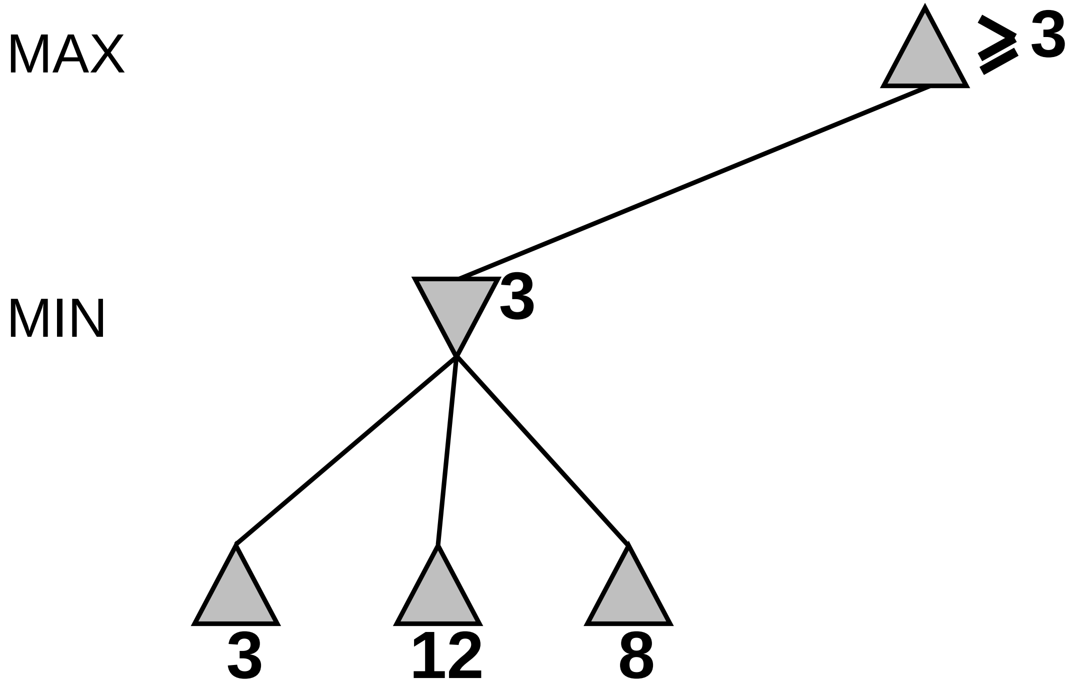
Does it work in practice?

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

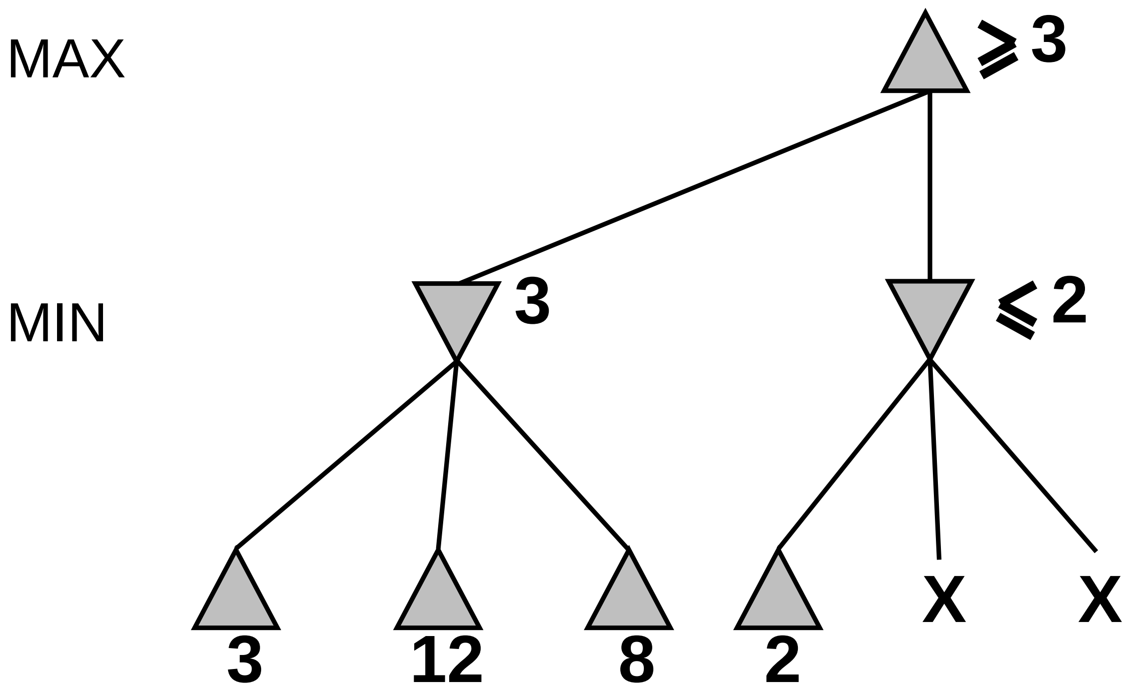
Not really, because ...

4-ply	$\approx$	**human novice**	**(hopeless chess player)**
8-ply	$\approx$	**typical PC, human master**	
12-ply	$\approx$	**Deep Blue, Kasparov**	

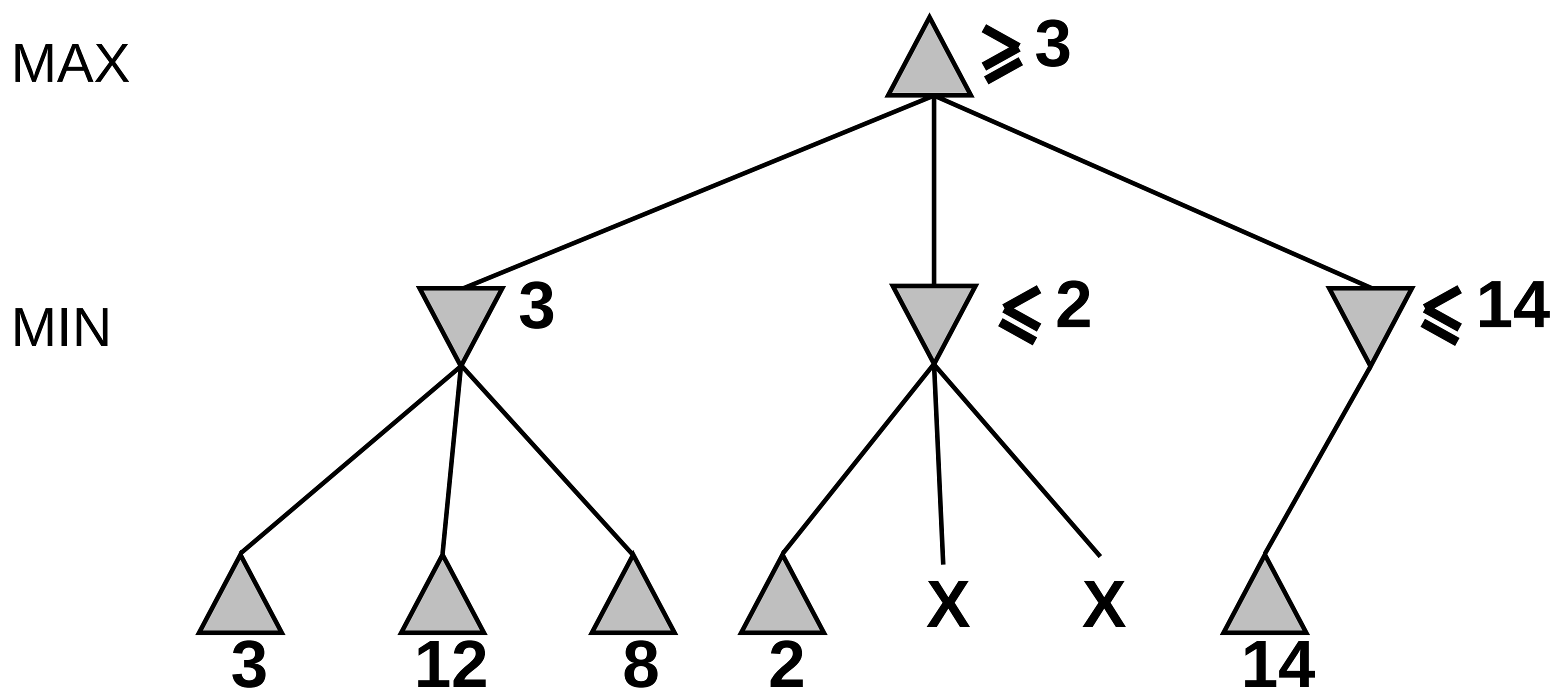
α-β Pruning Example



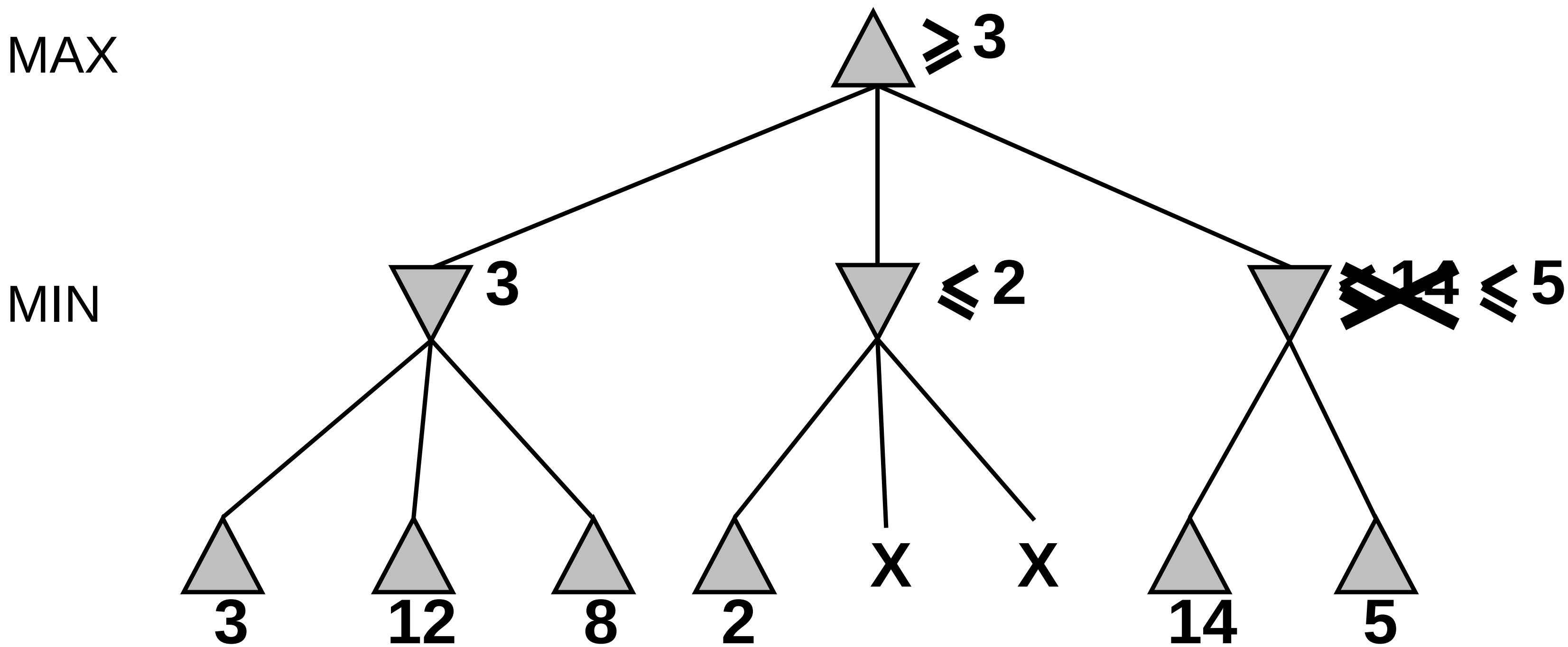
α-β Pruning Example



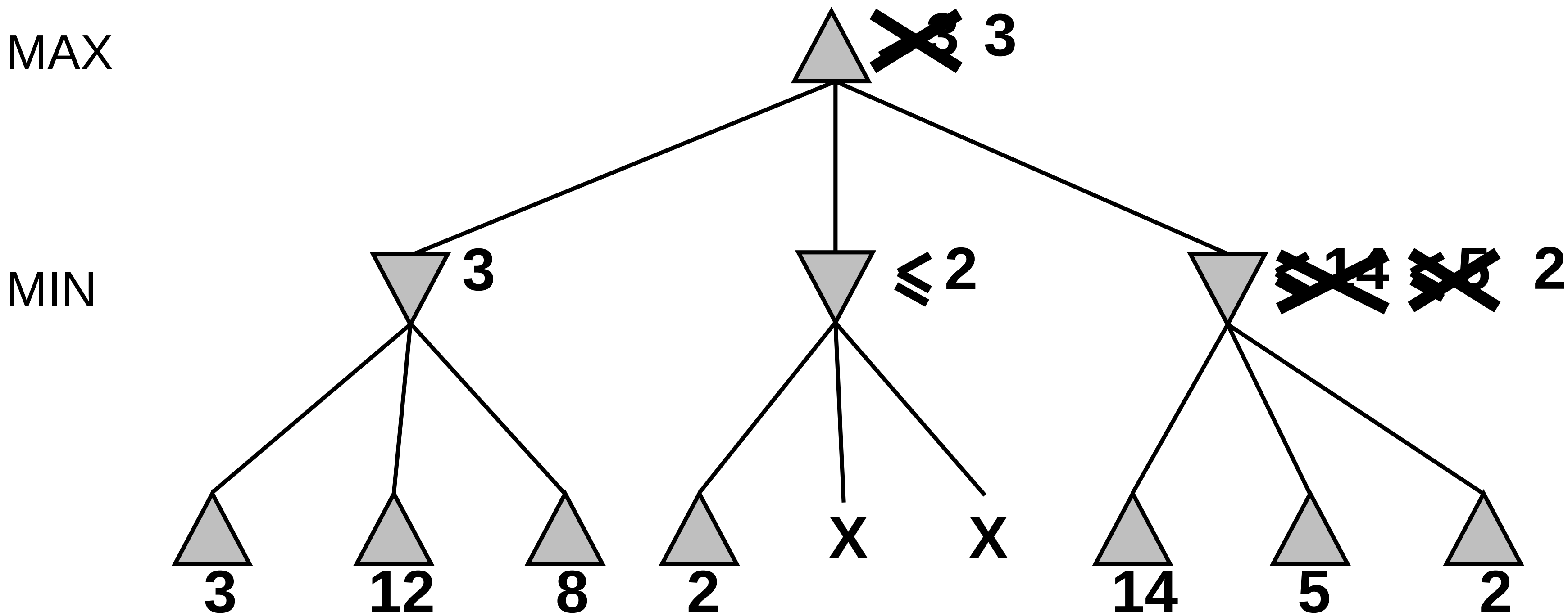
α-β Pruning Example



α-β Pruning Example



α-β Pruning Example



Properties of α-β

Effects of pruning

- **Reduces the search space**
- **Does not affect final result**

Properties of α-β

Effects of pruning

- Reduces the search space
- Does not affect final result

Effectiveness

Good move ordering improves effectiveness

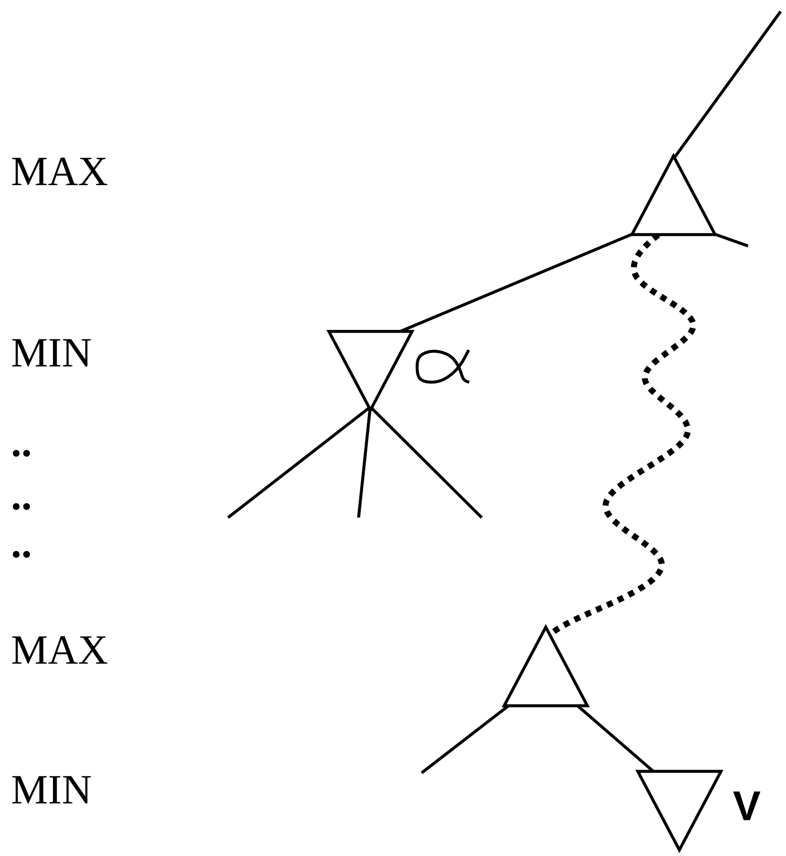
Time complexity with "perfect ordering": $O(b^{m/2})$

Doubles depth of search

For chess:

Can easily reach depth 8 and play good chess

The Idea of α-β



α is the best value (to MAX) found so far off the current path

If value x of some node below V is known to be less than α,

then value of V is known to be at most x, i.e., less than α,

therefore MAX will avoid node V

Consequence

No need to expand further nodes below V

The α-β Algorithm

function MAX-VALUE(*state, game,* α, β) **returns** the minimax value of *state*

inputs: *state* /* current state in game */

game /* game description */

α /* the best score for MAX along the path to *state* */

β /* the best score for MIN along the path to *state* */

if CUTOFF-TEST(*state*) **then return** EVAL(*state*)

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

$\alpha \leftarrow$ MAX(α, MIN-VALUE(*s, game,* α, β))

if $\alpha \geq \beta$ **then return** β

end

return α

The α-β Algorithm

function MIN-VALUE(*state, game*, α, β) **returns** the minimax value of *state*

inputs: *state* /* current state in game */

game /* game description */

α /* the best score for MAX along the path to *state* */

β /* the best score for MIN along the path to *state* */

if CUTOFF-TEST(*state*) **then return** EVAL(*state*)

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

$\beta \leftarrow$ MIN(β, MAX-VALUE(*s, game*, α, β))

if $\beta \leq \alpha$ **then return** α

end

return β

Deterministic Games in Practice

Checkers

Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Deterministic Games in Practice

Checkers

Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess

Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Deterministic Games in Practice

Checkers

Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

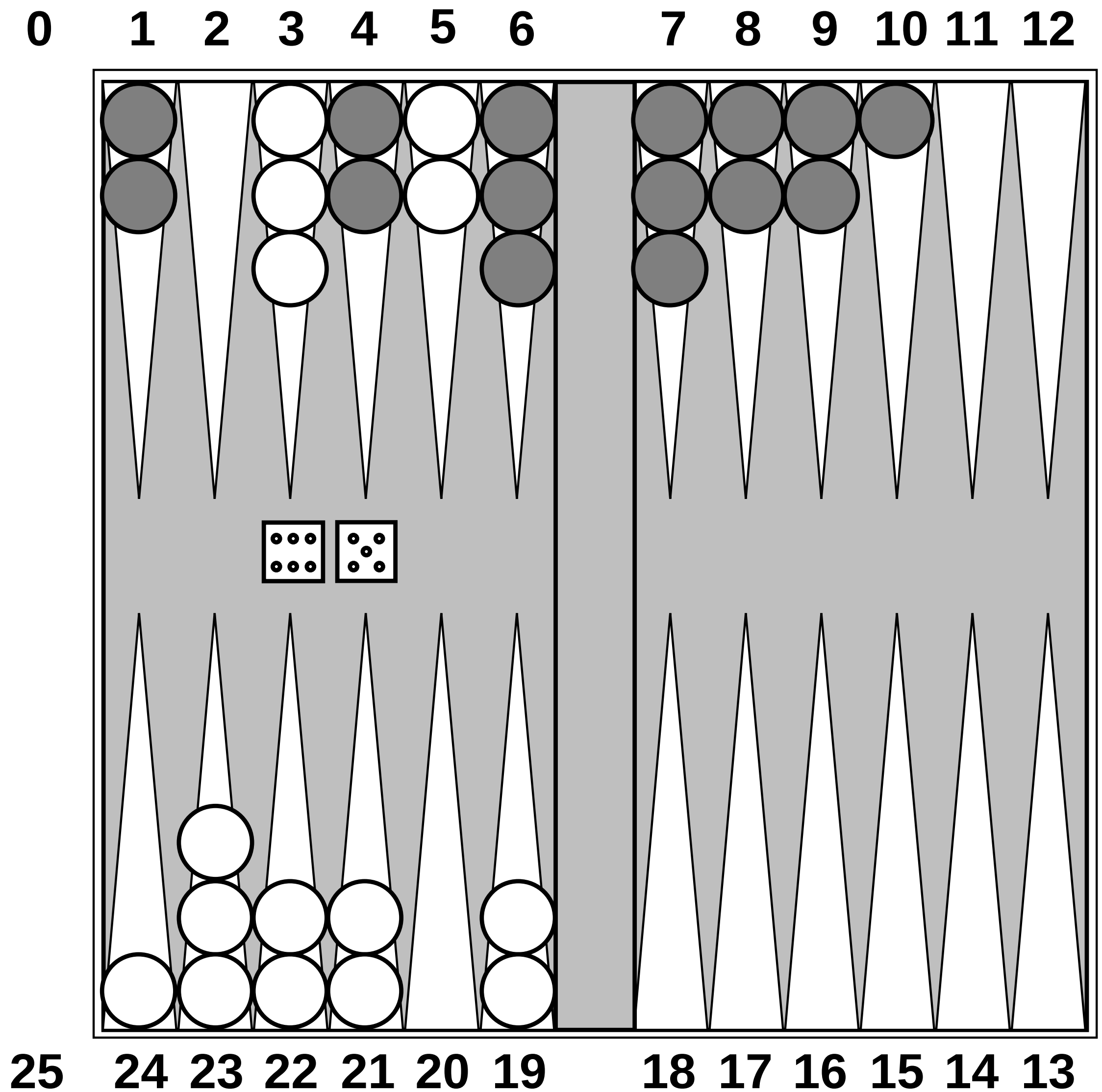
Chess

Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Go

Human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

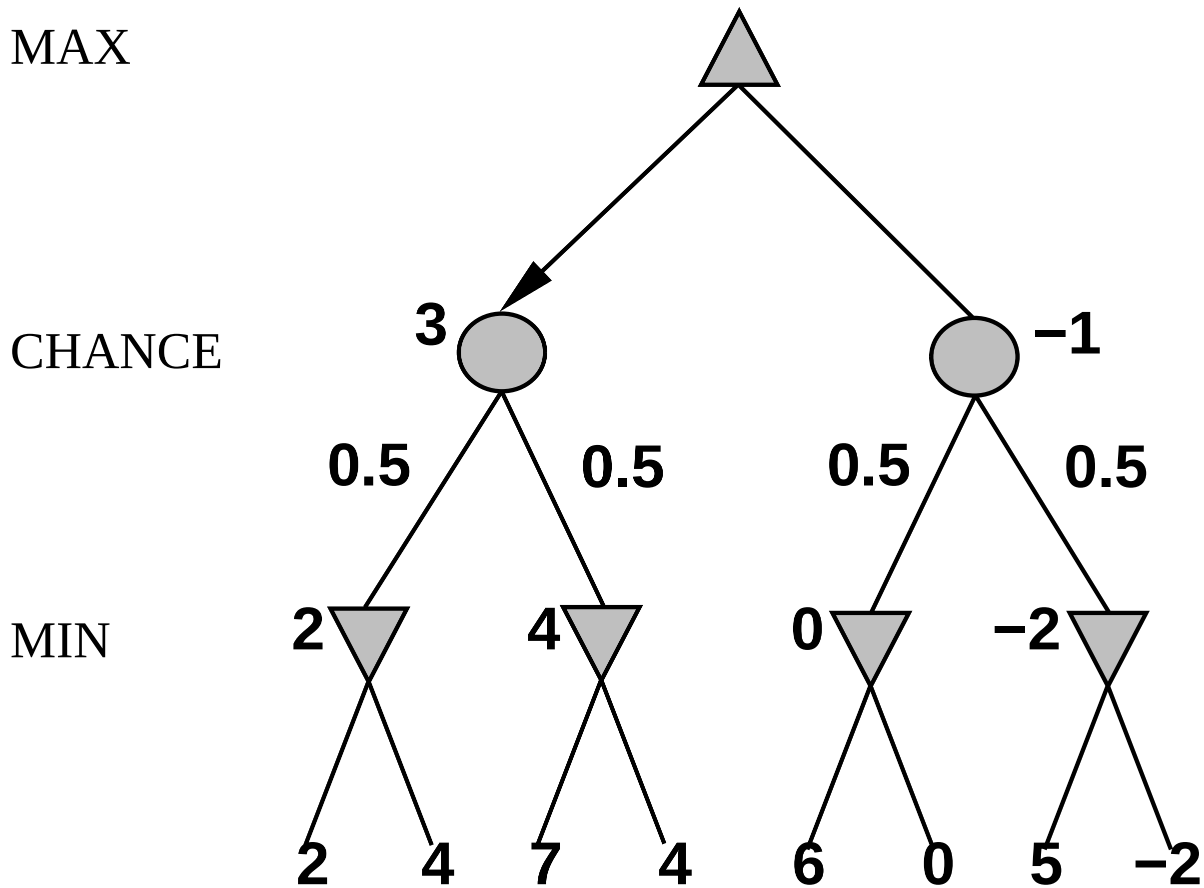
Nondeterministic Games: Backgammon



Nondeterministic Games in General

Chance introduced by dice, card-shuffling, etc.

Simplified example with coin-flipping



Algorithm for Nondeterministic Games

ExpectMinimax gives perfect play

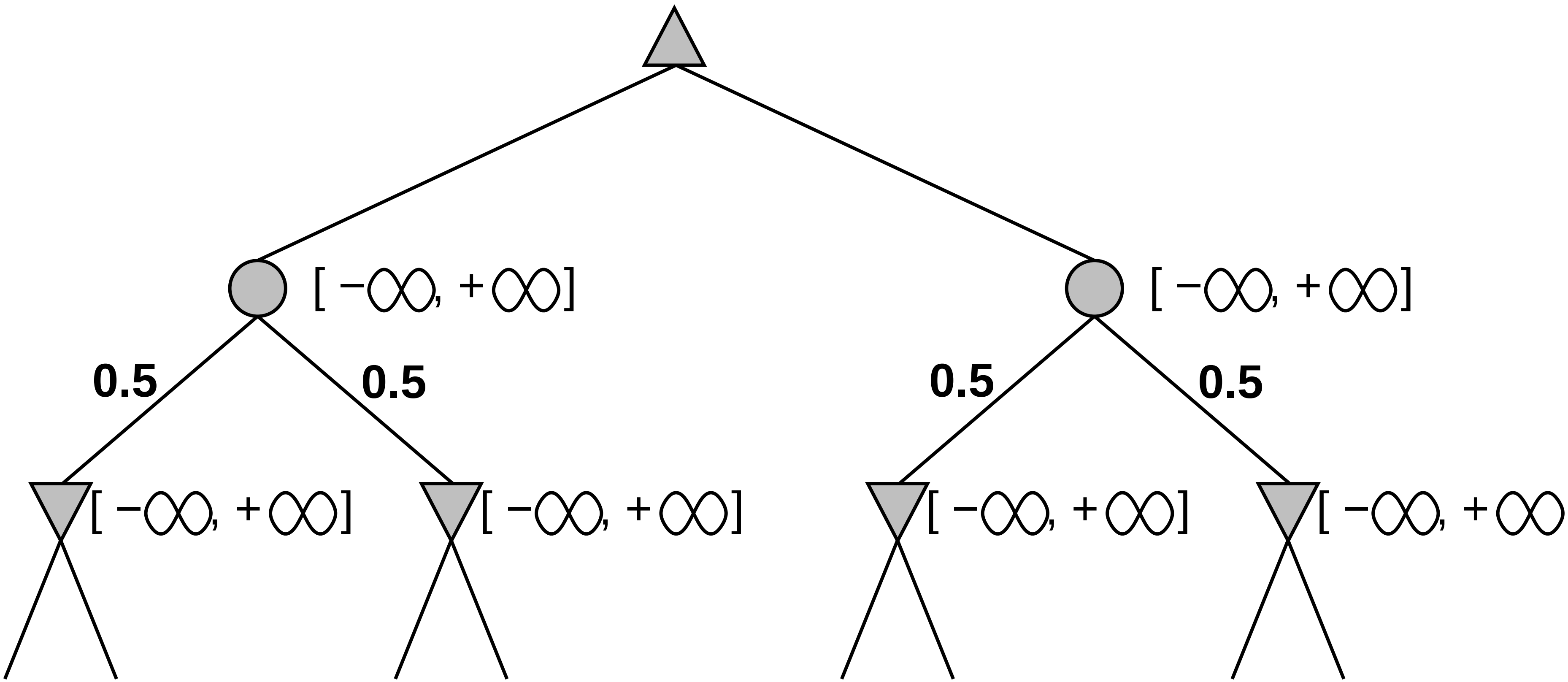
if *state* **is a Max node then**
return the highest ExpectMinimax value of Successors(*state***)**

if *state* **is a Min node then**
return the lowest ExpectiMinimax value of Successors(*state***)**

if *state* **is a chance node then**
return average of ExpectMinimax value of Successors(*state***)**

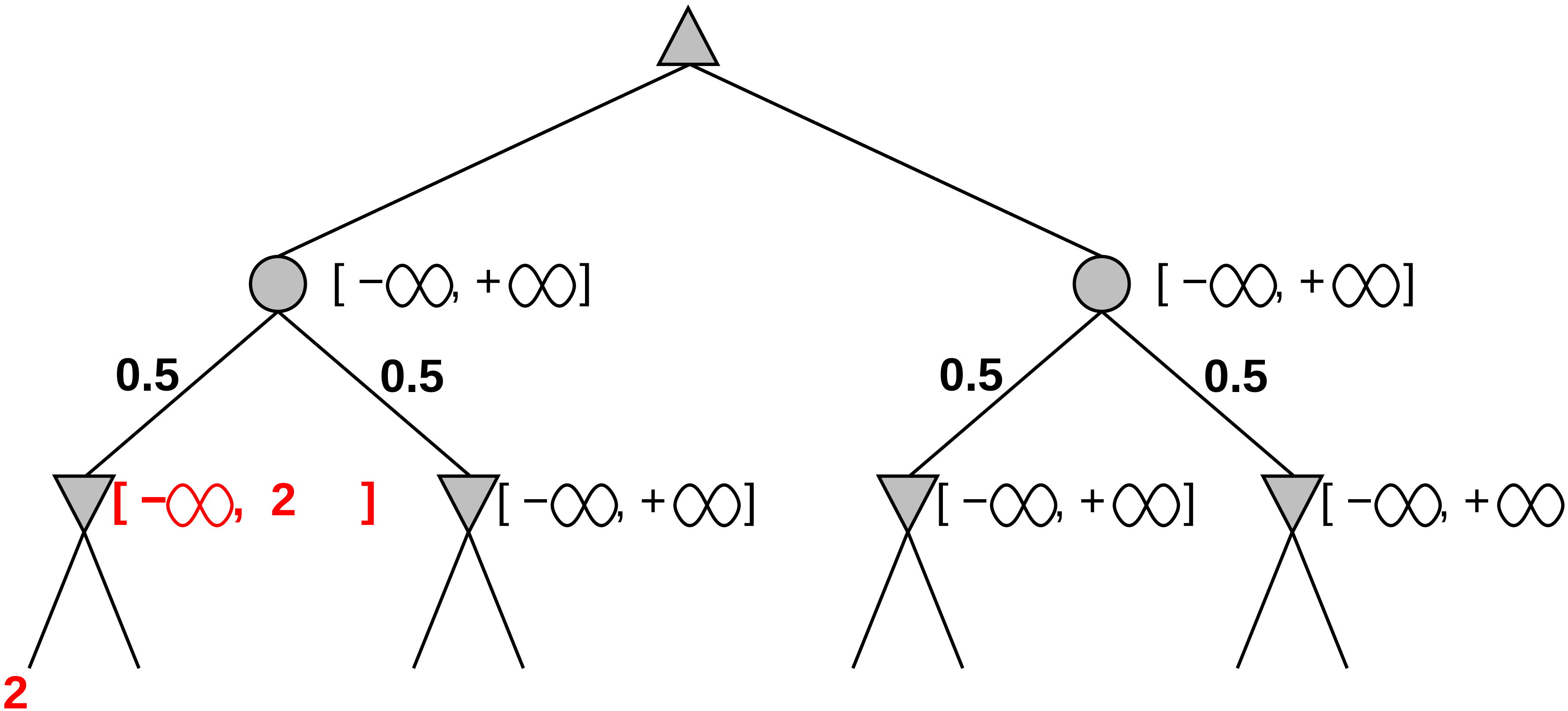
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



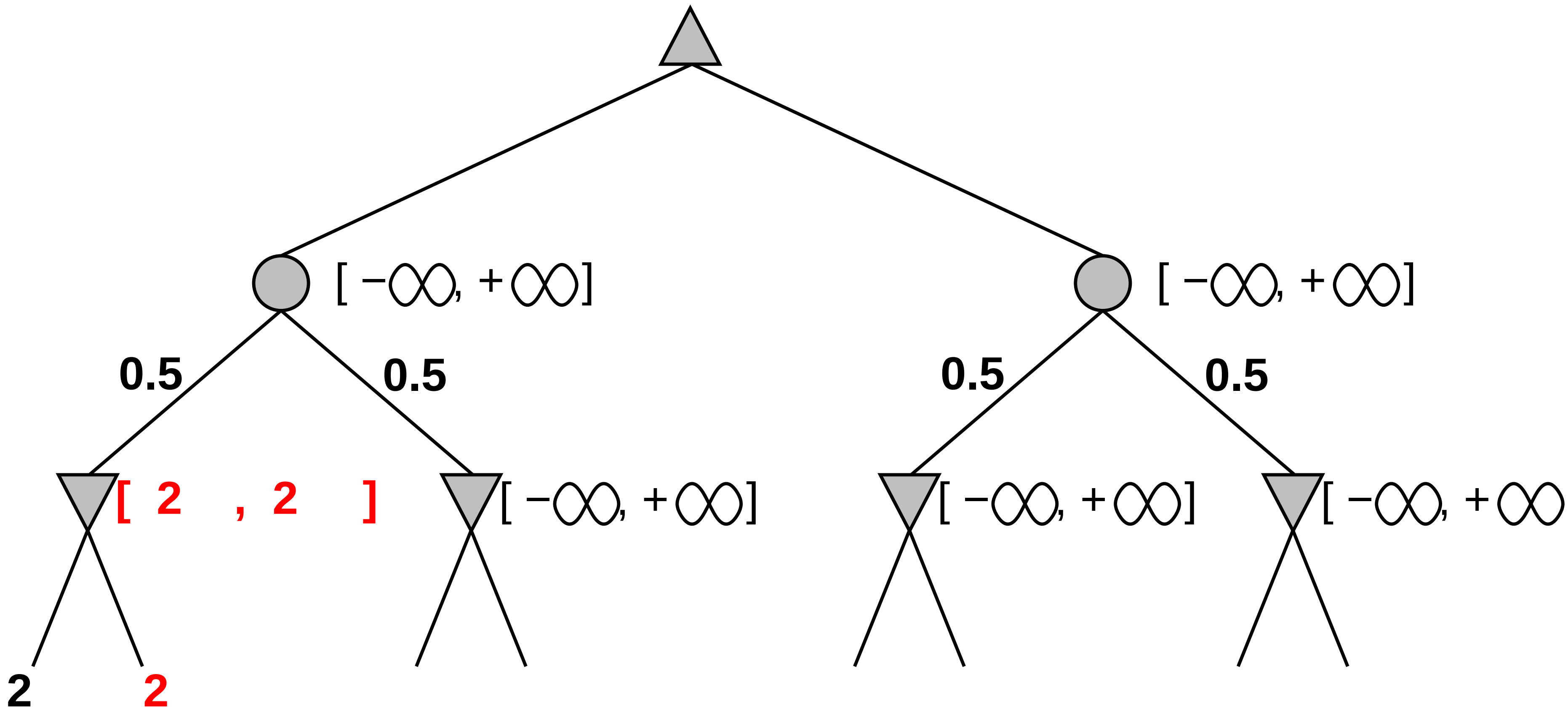
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



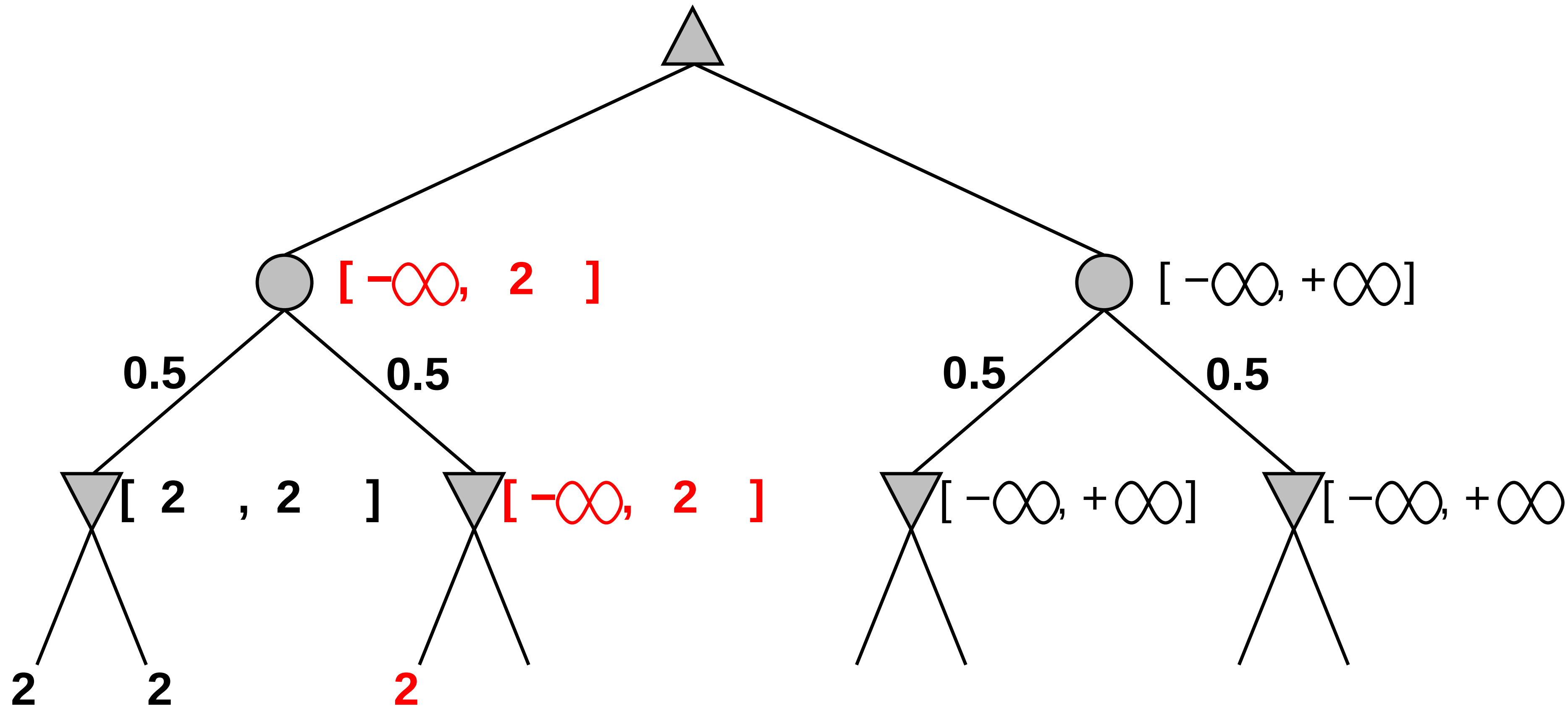
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



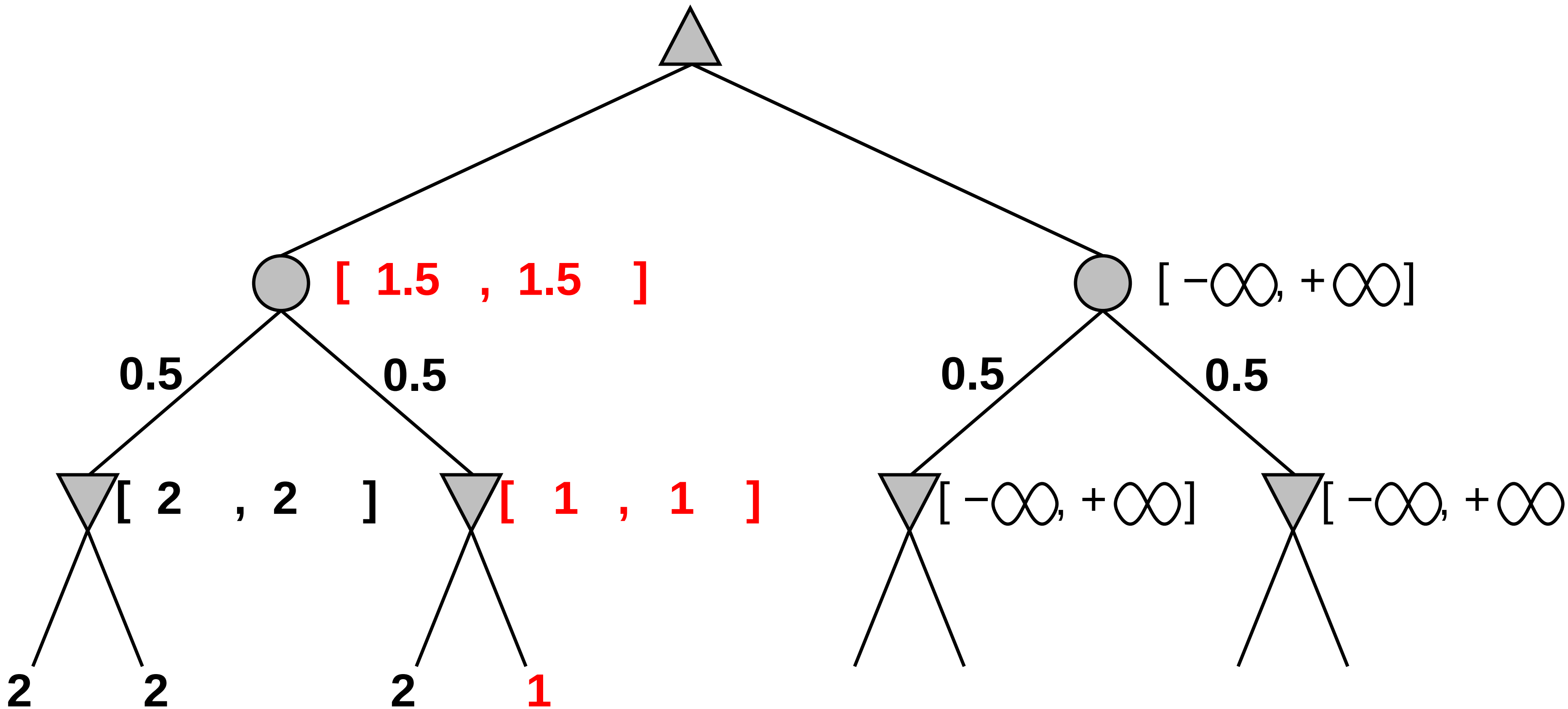
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



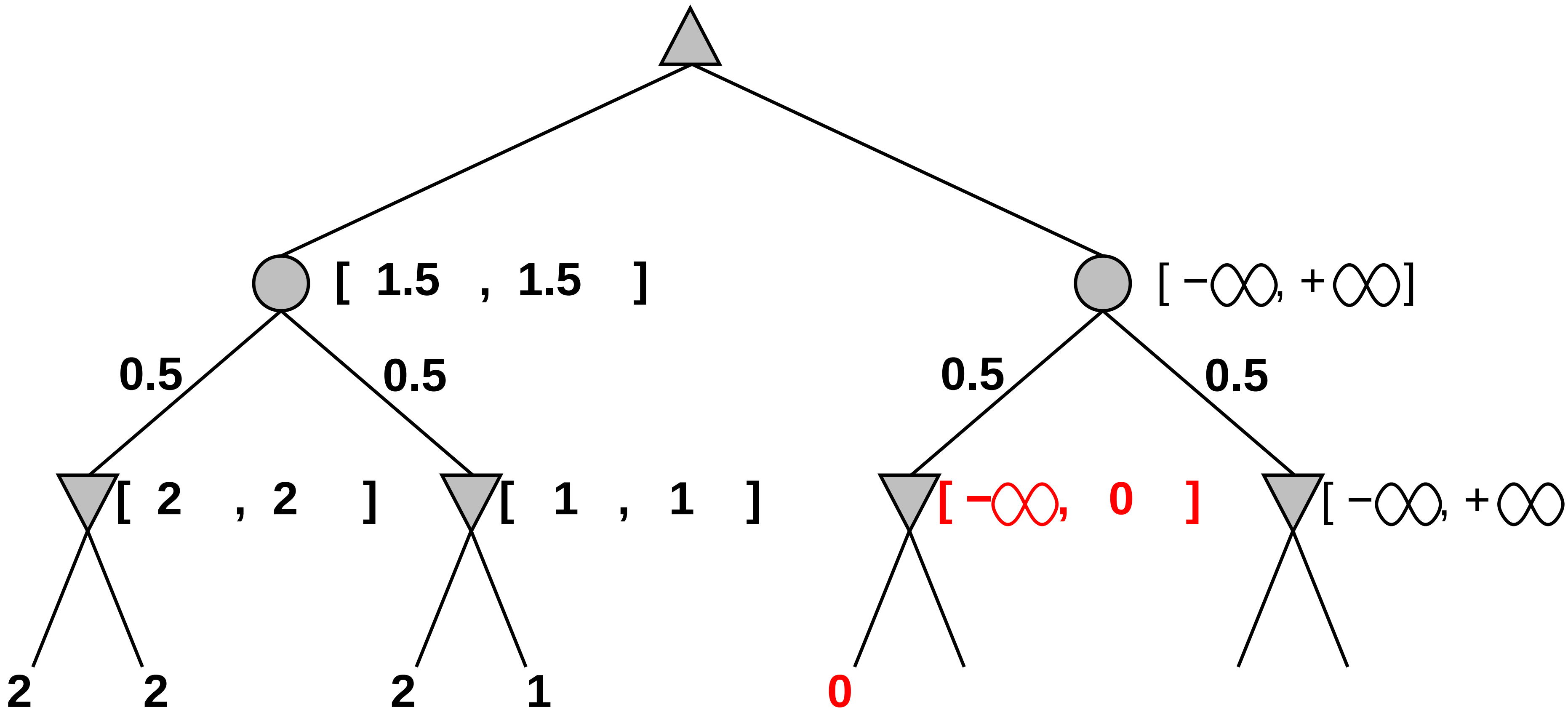
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



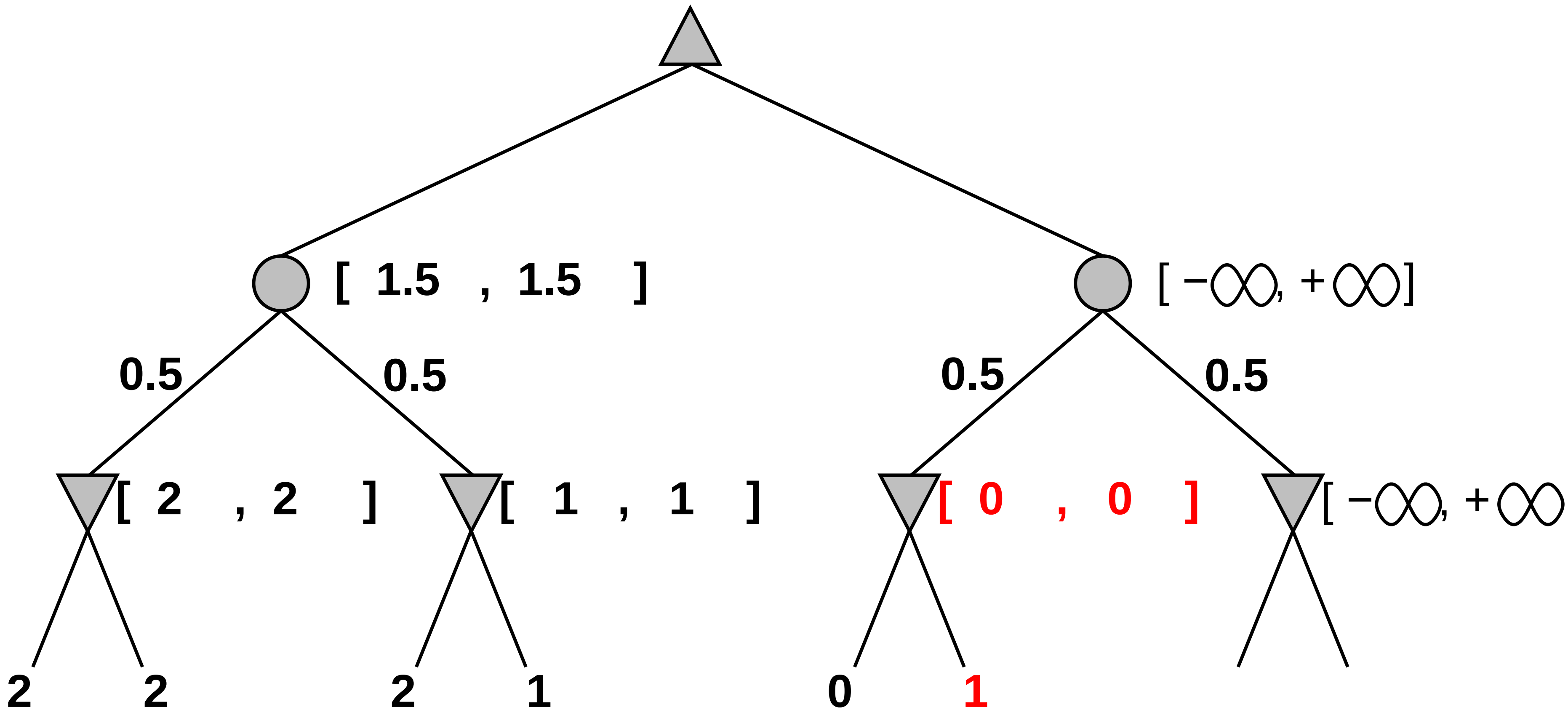
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



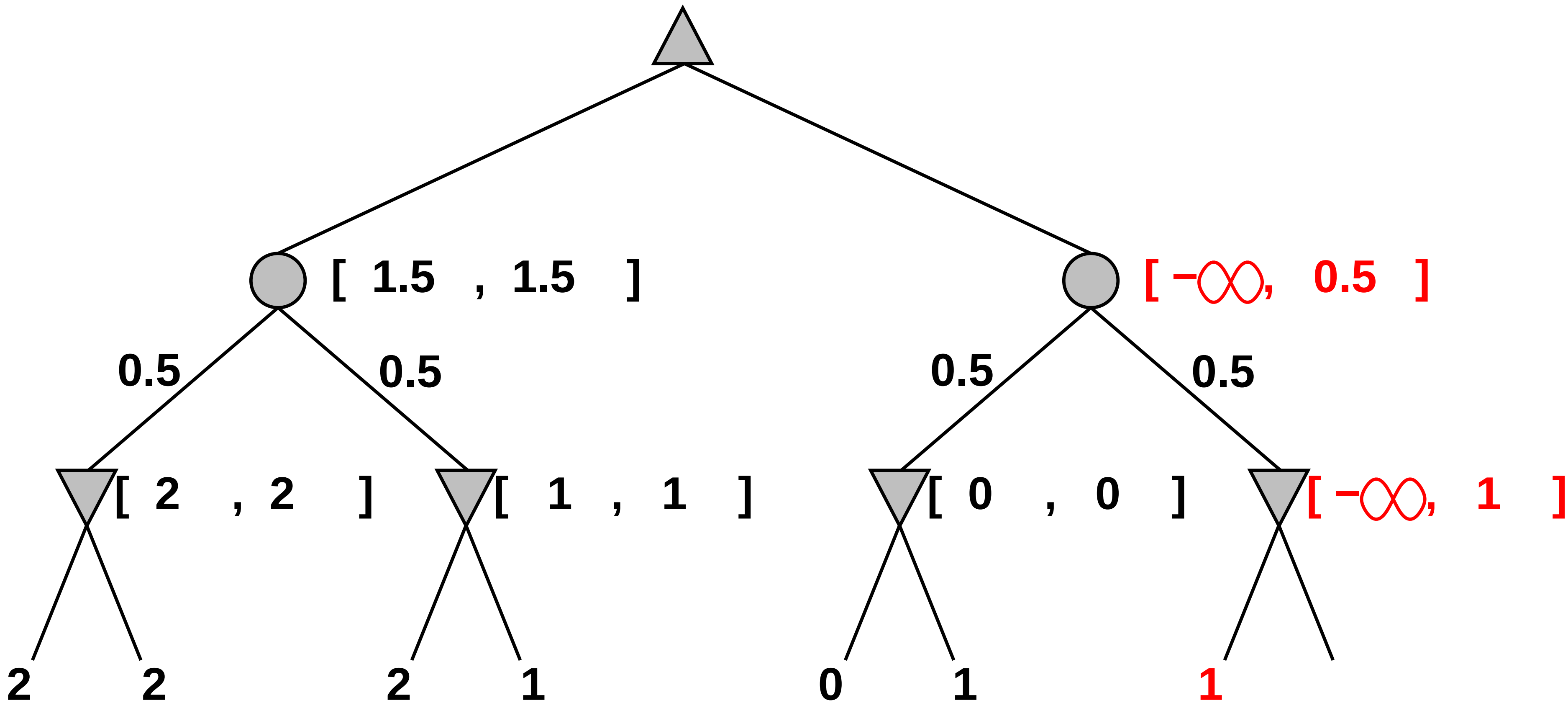
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



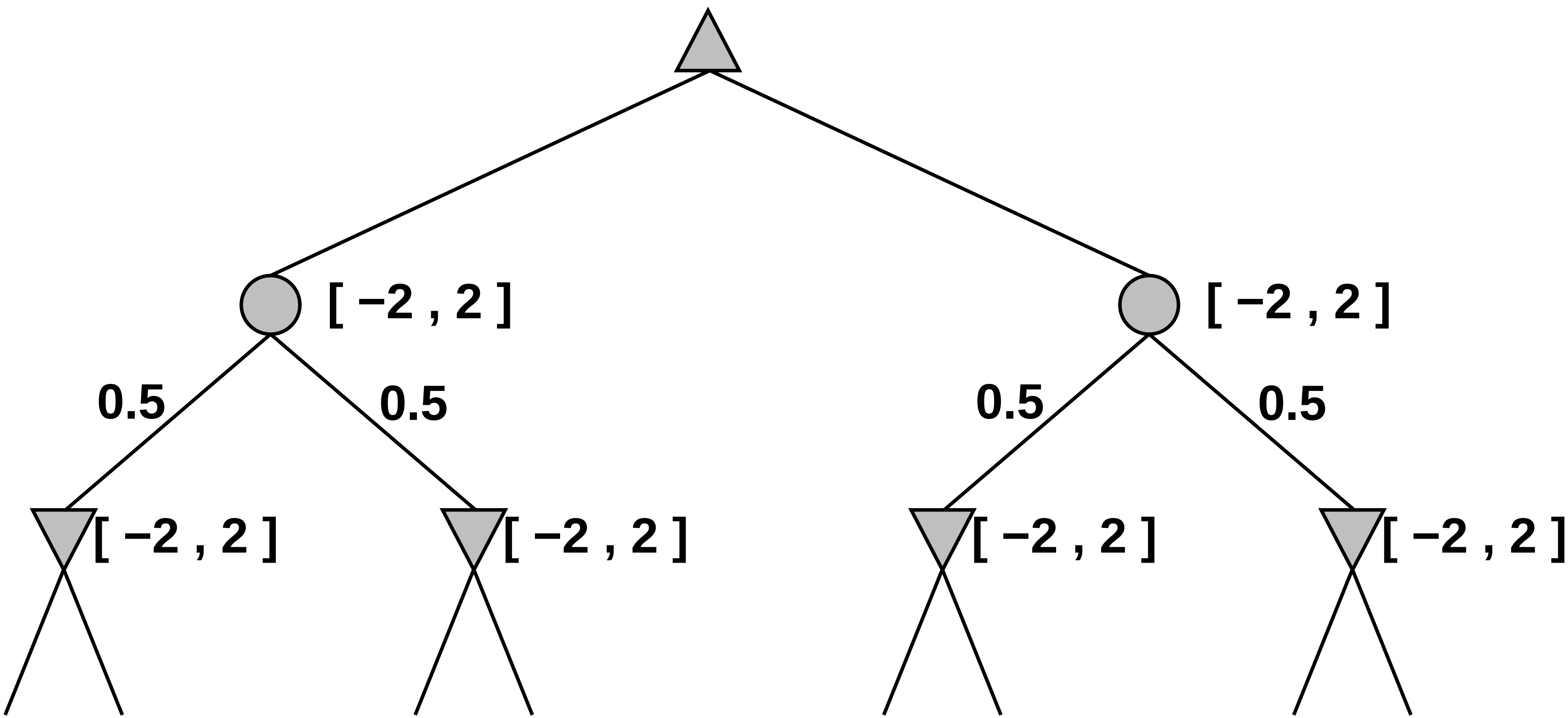
Pruning in Nondeterministic Game Trees

A version of α-β pruning is possible



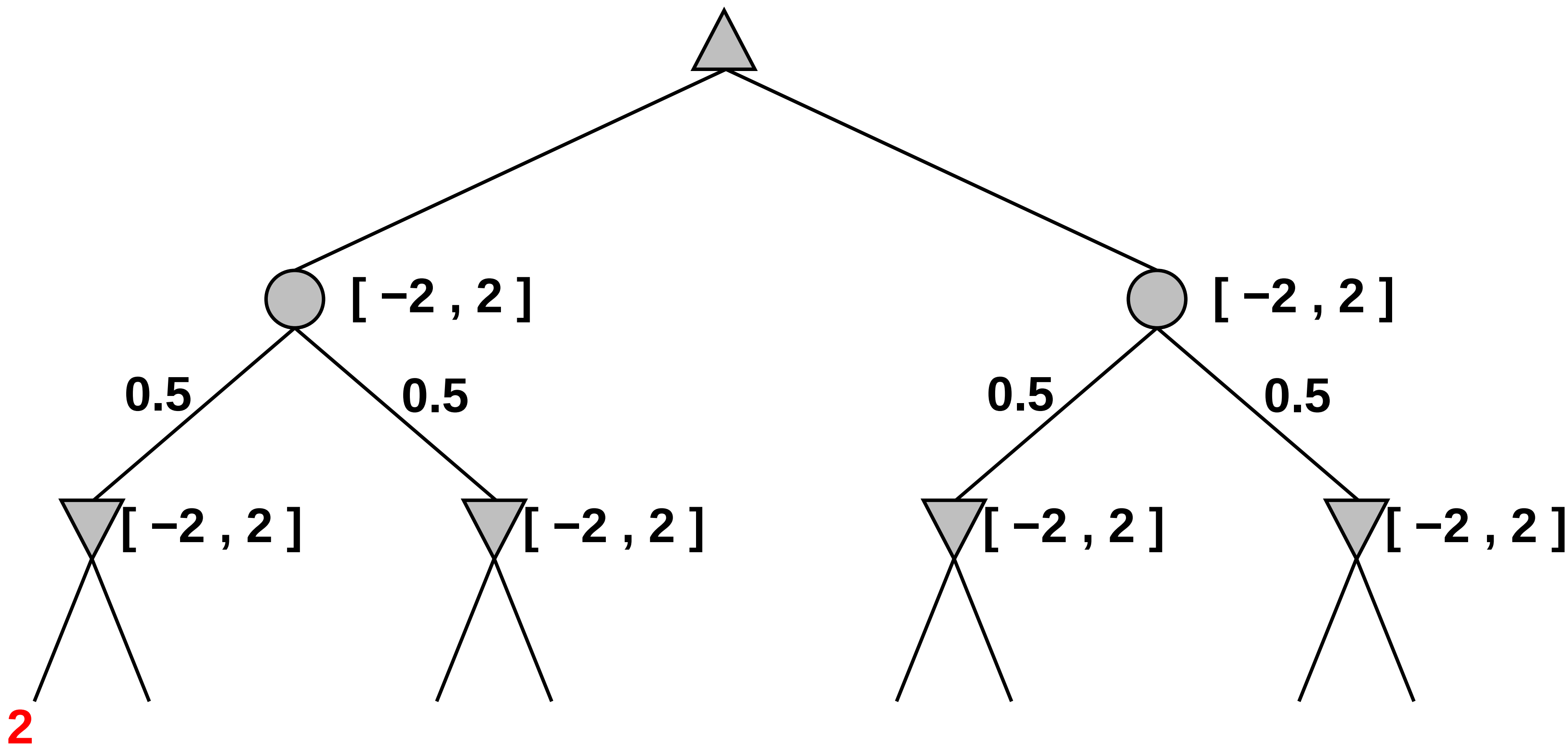
Pruning Continued

More pruning occurs if we can bound the leaf values



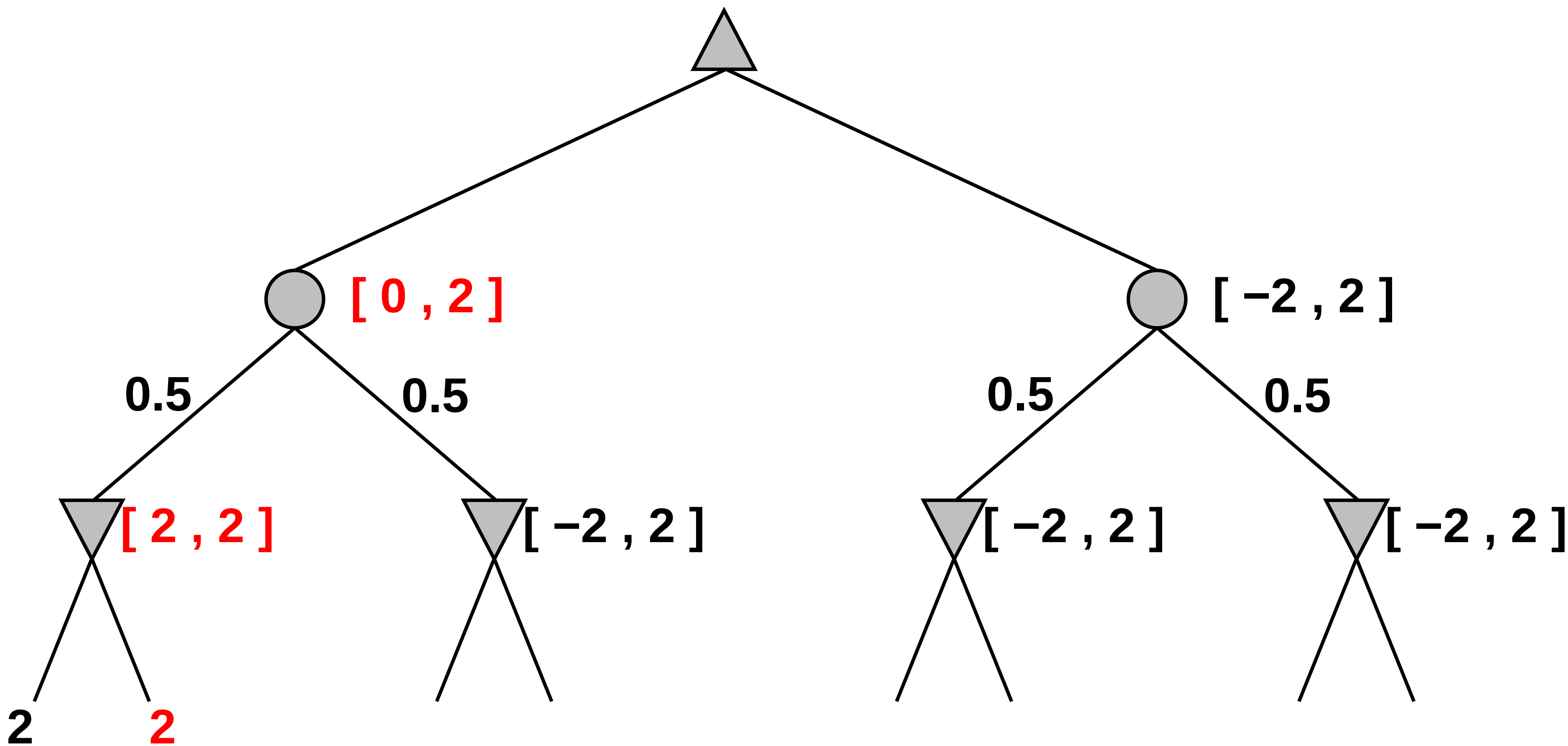
Pruning Continued

More pruning occurs if we can bound the leaf values



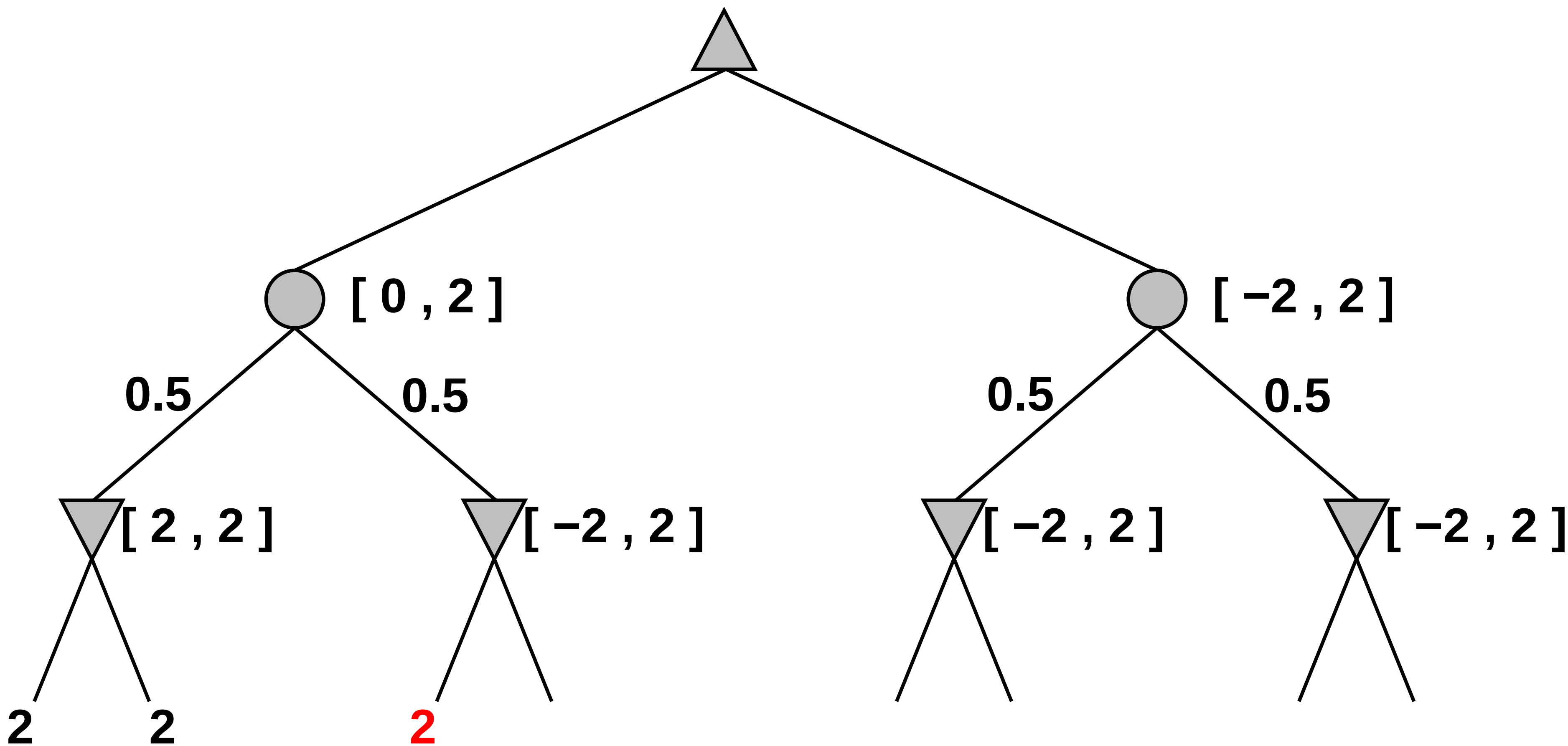
Pruning Continued

More pruning occurs if we can bound the leaf values



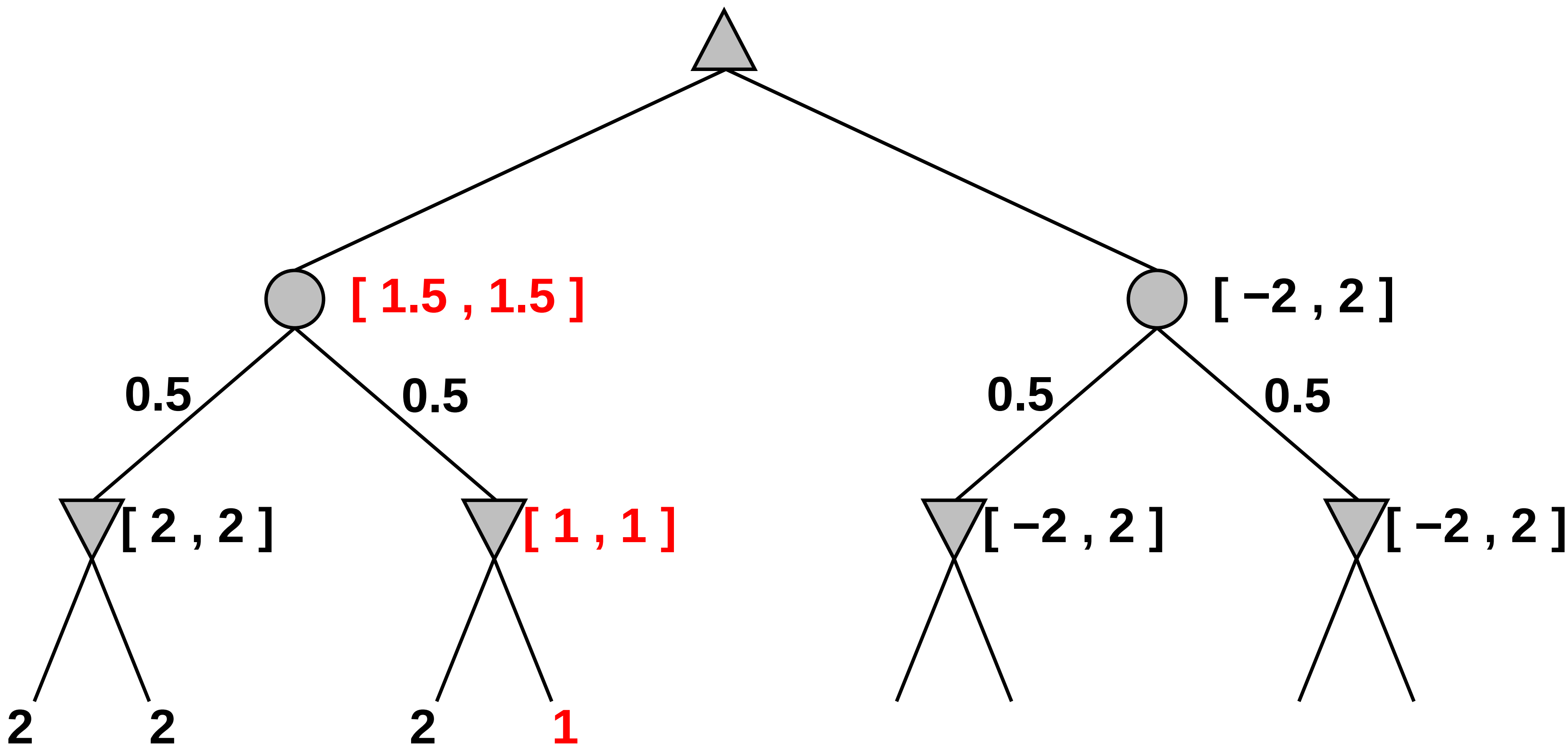
Pruning Continued

More pruning occurs if we can bound the leaf values



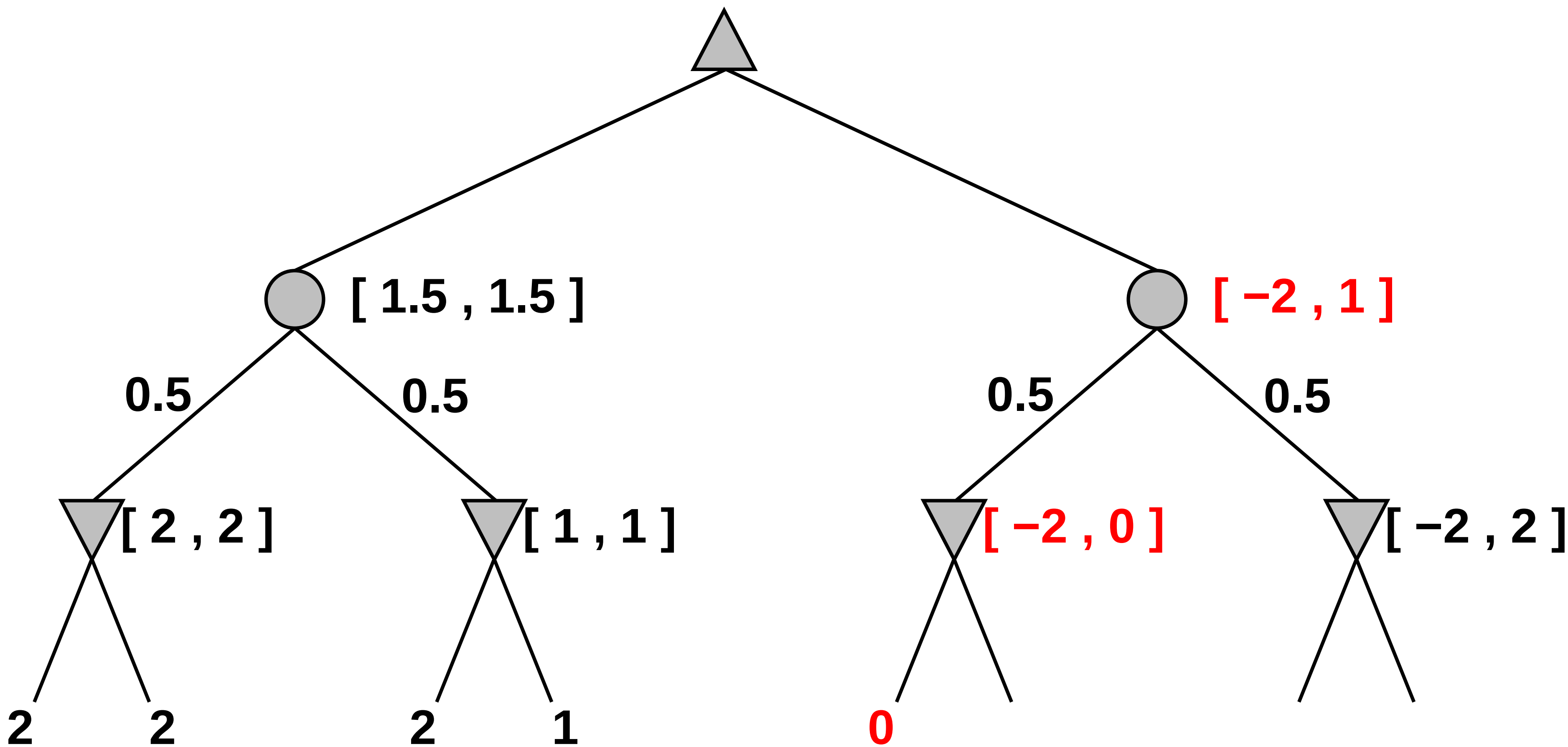
Pruning Continued

More pruning occurs if we can bound the leaf values



Pruning Continued

More pruning occurs if we can bound the leaf values



Nondeterministic Games in Practice

Problem

α-β pruning is much less effective

Dice rolls increase b

21 possible rolls with 2 dice

Backgammon

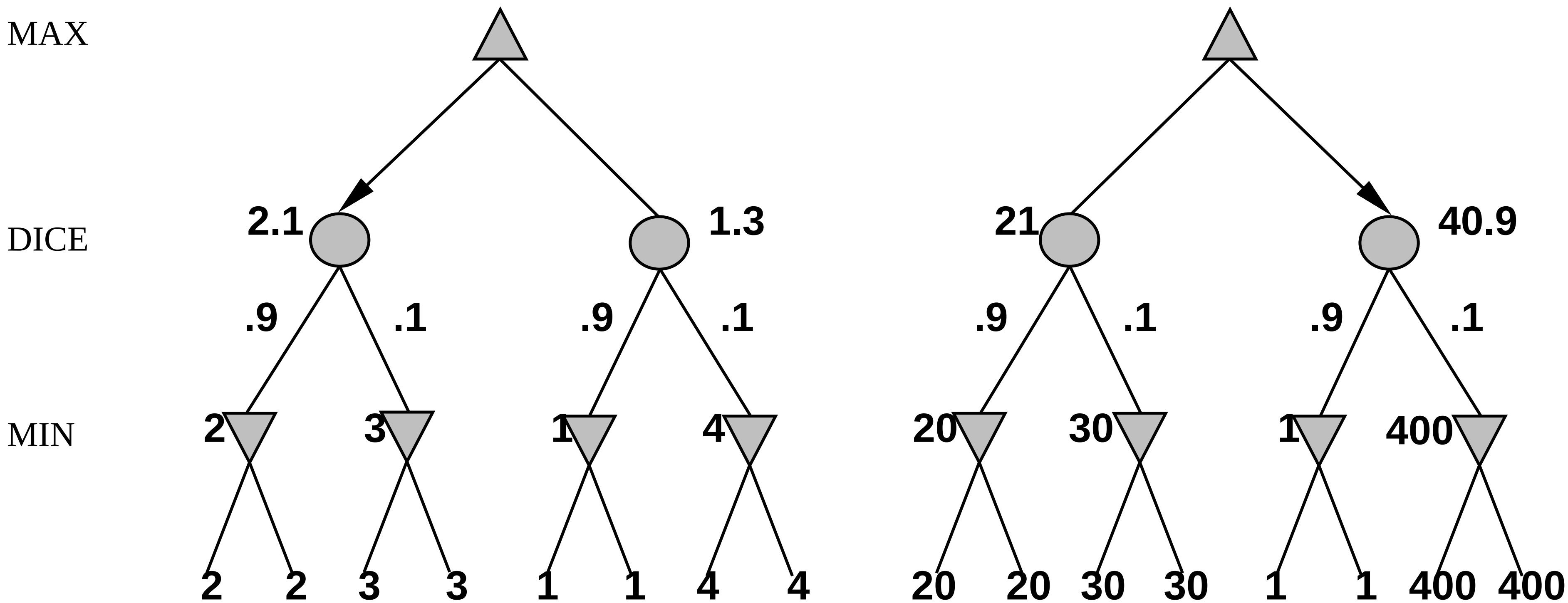
$\approx$ 20 legal moves

$$\text{depth } 4 \quad = \quad 20^4 \times 21^3 \quad \approx \quad 1.2 \times 10^9$$

TDGAMMON

Uses depth-2 search + very good EVAL $\approx$ world-champion level

Digression: Exact Values DO Matter



Behaviour is preserved only by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff

Games of Imperfect Information

Typical examples

Card games: Bridge, poker, skat, etc.

Note

Like having one big dice roll at the beginning of the game

Games of Imperfect Information

Idea for computing best action

Compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals

Requires information on probability the different deals

Special case

If an action is optimal for all deals, it's optimal.

Games of Imperfect Information

Idea for computing best action

Compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals

Requires information on probability the different deals

Special case

If an action is optimal for all deals, it's optimal.

Bridge

GIB, current best bridge program, approximates this idea by

- **generating 100 deals consistent with bidding information**
- **picking the action that wins most tricks on average**

Commonsense Example

Day 1

Road A leads to a small heap of gold pieces 10 **points**

Road B leads to a fork:

- **take the left fork and you'll find a mound of jewels** 100 **points**
- **take the right fork and you'll be run over by a bus** -1000 **points**

Best action: Take road B (100 **points)**

Commonsense Example

Day 1

Road A leads to a small heap of gold pieces 10 **points**

Road B leads to a fork:

- **take the left fork and you'll find a mound of jewels** 100 **points**
- **take the right fork and you'll be run over by a bus** -1000 **points**

Best action: Take road B (100 points)

Day 2

Road A leads to a small heap of gold pieces 10 **points**

Road B leads to a fork:

- **take the left fork and you'll be run over by a bus** -1000 **points**
- **take the right fork and you'll find a mound of jewels** 100 **points**

Best action: Take road B (100 points)

Commonsense Example

Day 3

Road A leads to a small heap of gold pieces (10 points)

Road B leads to a fork:

- **guess correctly and you'll find a mound of jewels** 100 **points**
- **guess incorrectly and you'll be run over by a bus** -1000 **points**

Best action: Take road A **(**10 **points)**

NOT: Take road B **(**$\frac{-1000+100}{2} = -450$ **points)**

Proper Analysis

Note

Value of an actions is NOT the average of values for actual states computed with perfect information

With partial observability, value of an action depends on the information state the agent is in

Proper Analysis

Note

Value of an actions is NOT the average of values for actual states computed with perfect information

With partial observability, value of an action depends on the information state the agent is in

Leads to rational behaviors such as

- **Acting to obtain information**
- **Signalling to one's partner**
- **Acting randomly to minimize information disclosure**

Summary

- Games are to AI as grand prix racing is to automobile design
- Games are fun to work on (and dangerous)
- They illustrate several important points about AI
  - perfection is unattainable, must approximate
  - it is a good idea to think about what to think about
  - uncertainty constrains the assignment of values to states