

---

# Game Playing

Philipp Koehn

29 September 2015



# Outline



- Games
- Perfect play
  - minimax decisions
  - $\alpha$ - $\beta$  pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information

# games

# Games vs. Search Problems



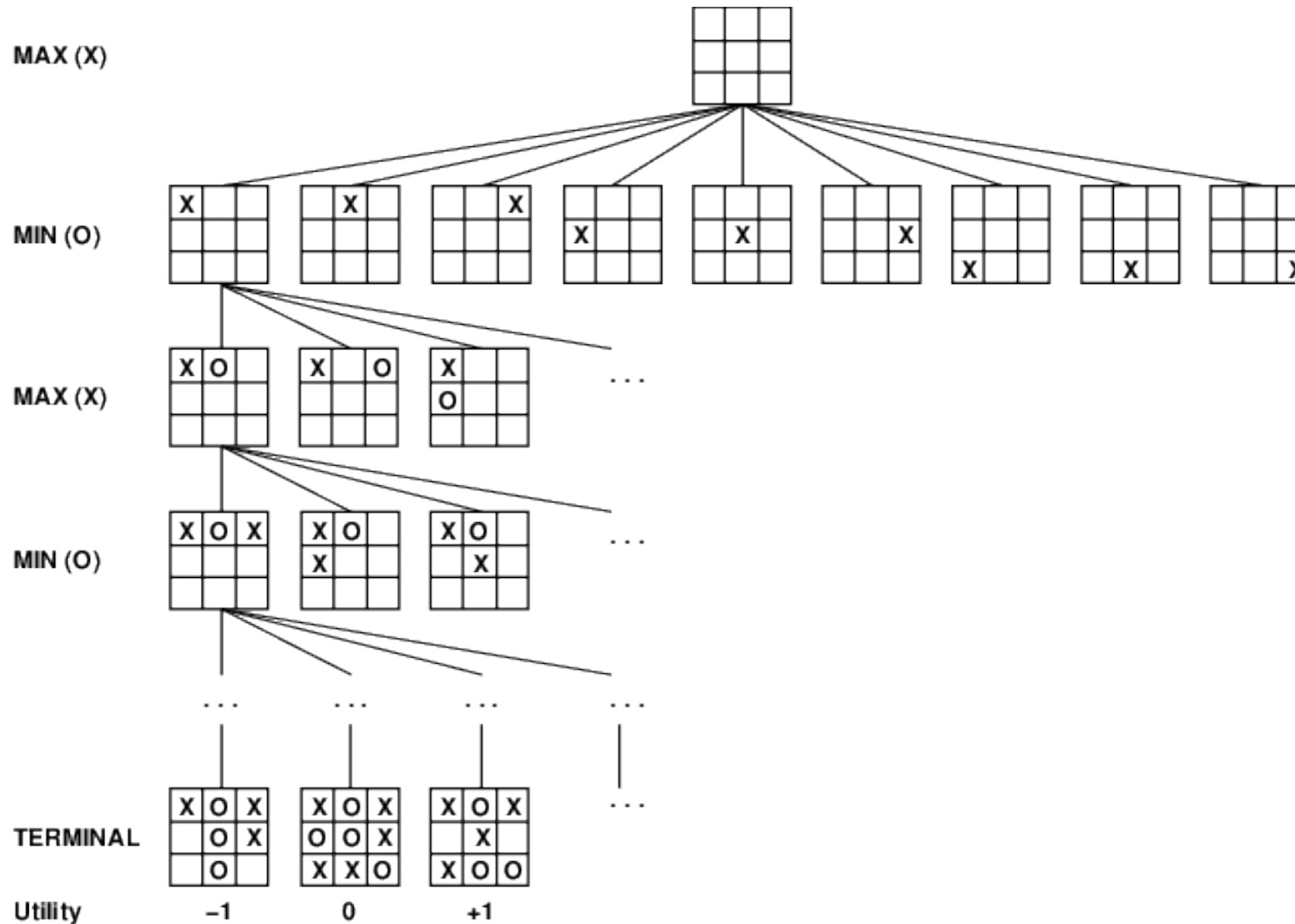
- “Unpredictable” opponent  $\Rightarrow$  solution is a **strategy** specifying a move for every possible opponent reply
- Time limits  $\Rightarrow$  unlikely to find goal, must approximate
- Plan of attack:
  - computer considers possible lines of play (Babbage, 1846)
  - algorithm for perfect play (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



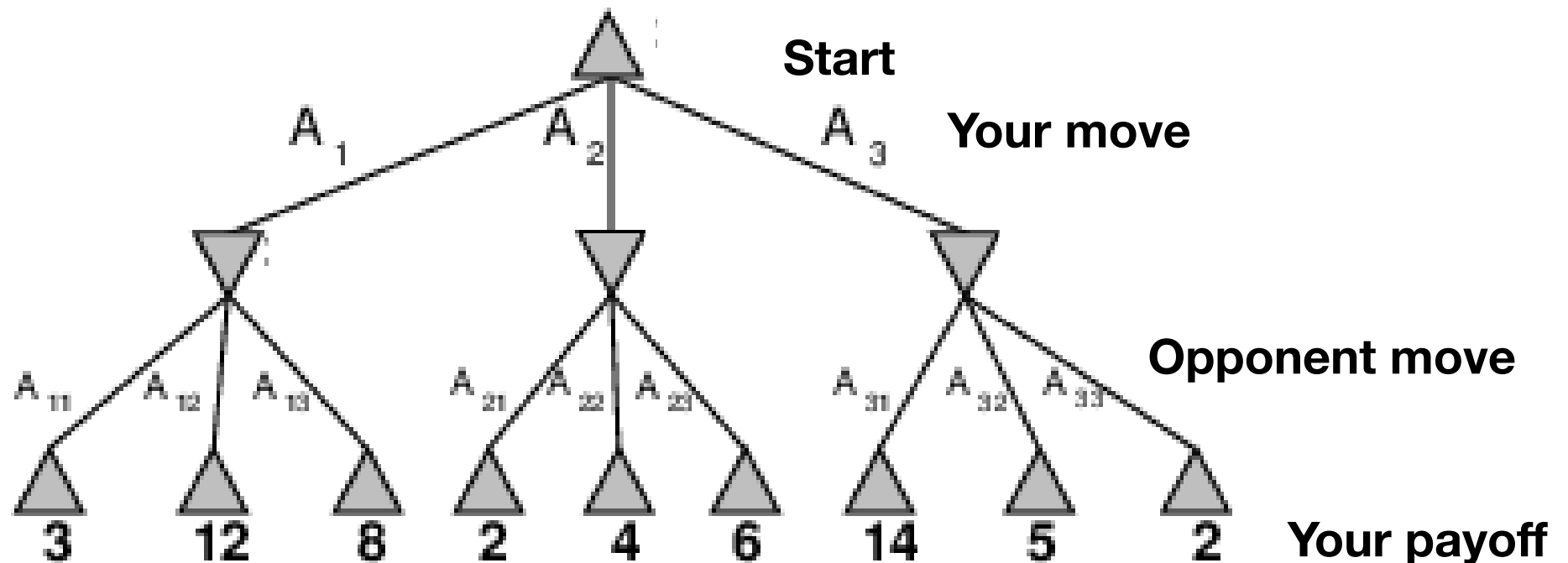
	<b>deterministic</b>	<b>chance</b>
<b>perfect information</b>	Chess Checkers Go Othello	Backgammon Monopoly
<b>imperfect information</b>	battleships Blind Tic Tac Toe	Bridge Poker Scrabble

# Game Tree (2-player, Deterministic, Turns)



# Simple Game Tree

- 2 player game
- Each player has one move
- You move first
- Goal: optimize your payoff (utility)



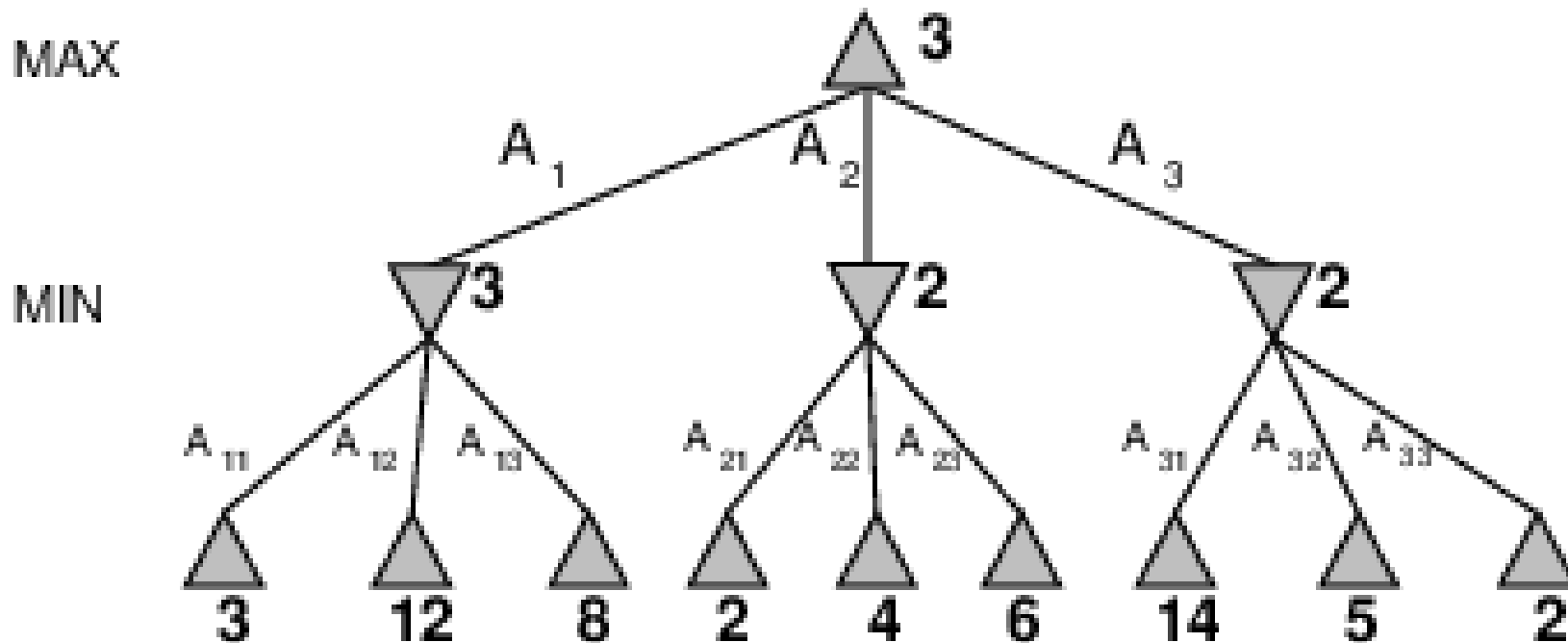
# minimax



# Minimax



- Perfect play for deterministic, perfect-information games
- Idea: choose move to position with highest **minimax value**  
= best achievable payoff against best play
- E.g., 2-player game, one move each:



# Minimax Algorithm



**function** MINIMAX-DECISION(*state*) **returns** *an action*

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

**return** the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

**function** MAX-VALUE(*state*) **returns** *a utility value*

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

$v \leftarrow -\infty$

**for** *a, s* in SUCCESSORS(*state*) **do**  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

**return** *v*

---

**function** MIN-VALUE(*state*) **returns** *a utility value*

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

$v \leftarrow \infty$

**for** *a, s* in SUCCESSORS(*state*) **do**  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

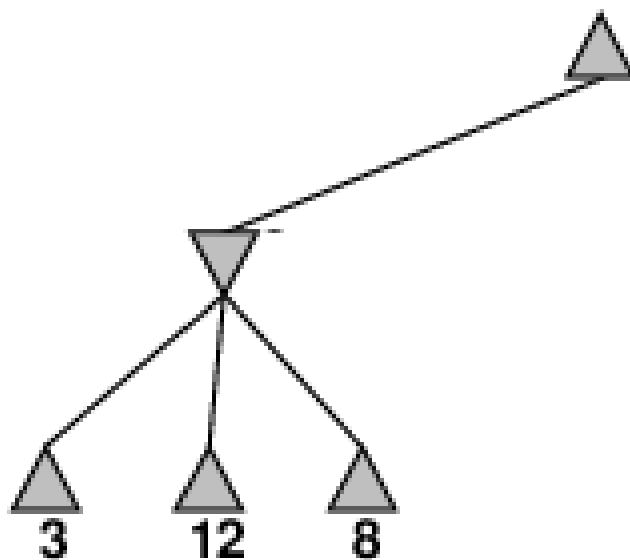
**return** *v*

# Properties of Minimax

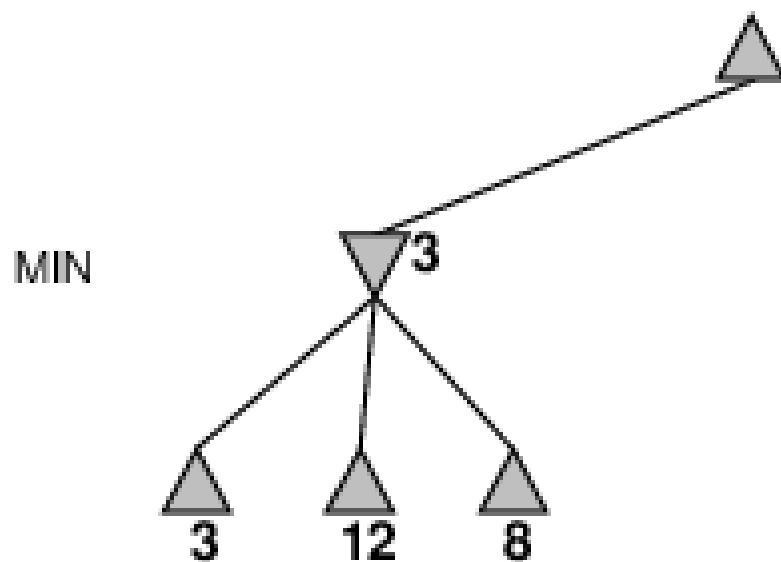


- **Complete?** ■ Yes, if tree is finite
- **Optimal?** ■ Yes, against an optimal opponent. Otherwise??
- **Time complexity?** ■  $O(b^m)$
- **Space complexity?** ■  $O(bm)$  (depth-first exploration)■
- For Chess,  $b \approx 35$ ,  $m \approx 100$  for “reasonable” games  
⇒ exact solution completely infeasible
- But do we need to explore every path?

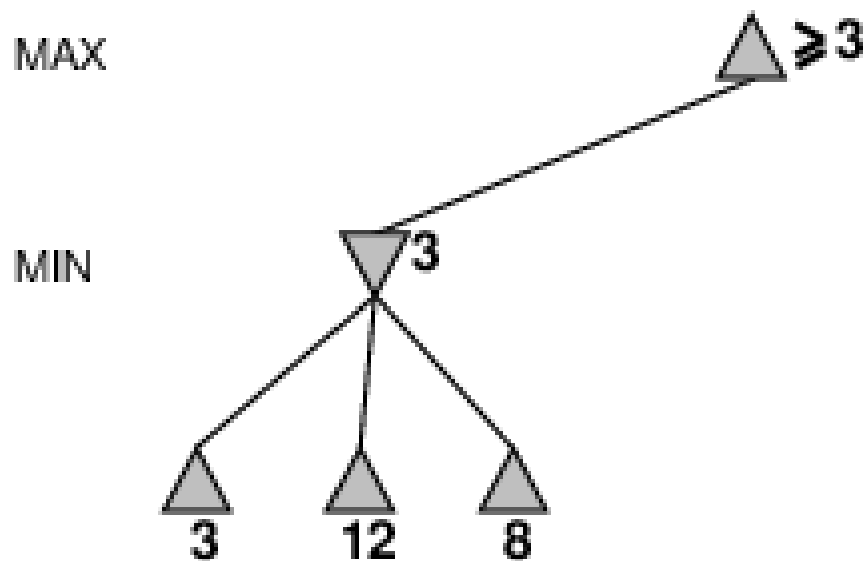
# $\alpha$ - $\beta$ Pruning Example



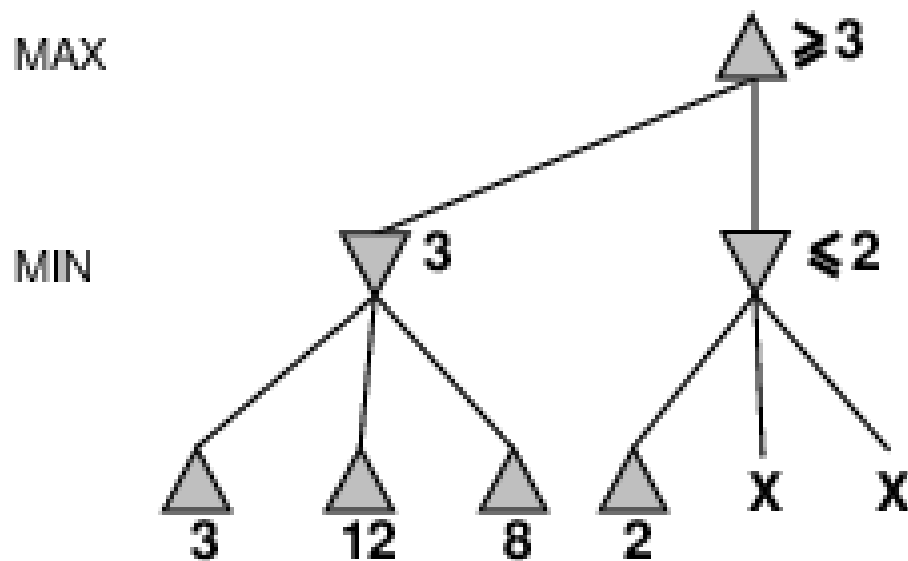
# $\alpha$ - $\beta$ Pruning Example



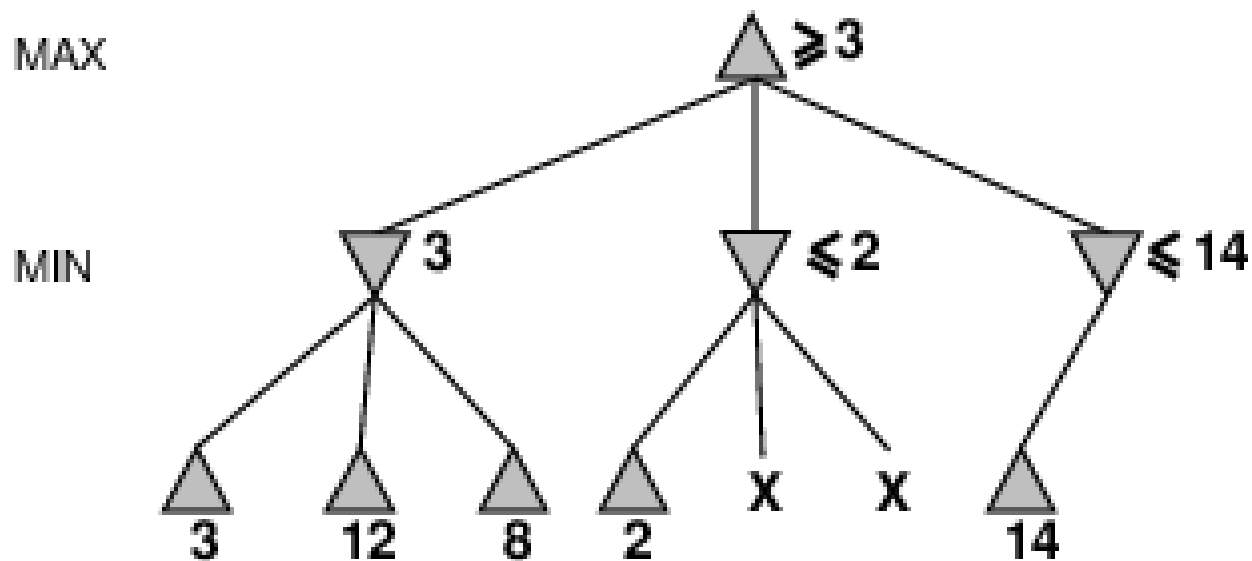
# $\alpha$ - $\beta$ Pruning Example



# $\alpha$ - $\beta$ Pruning Example

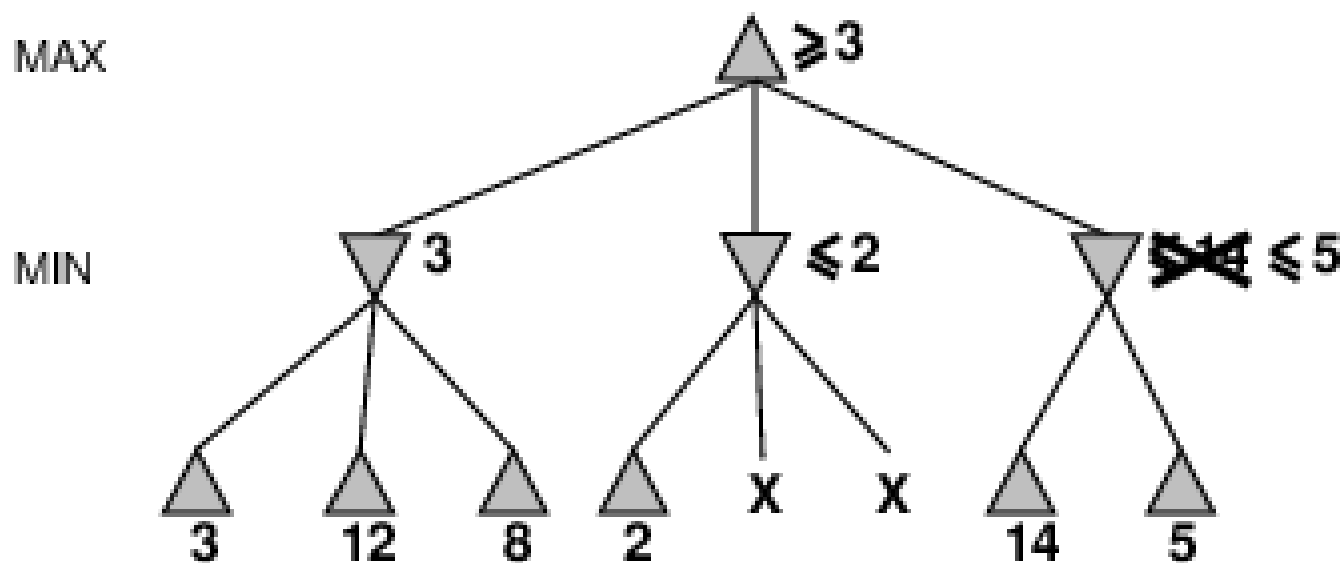


# $\alpha$ - $\beta$ Pruning Example

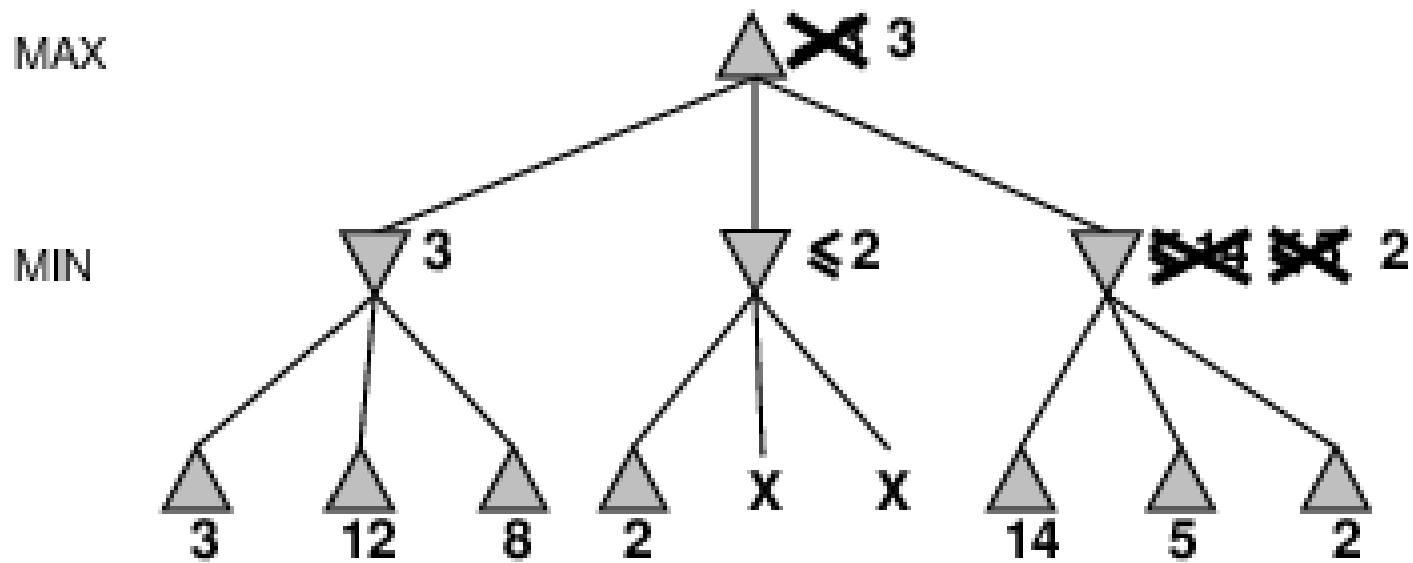




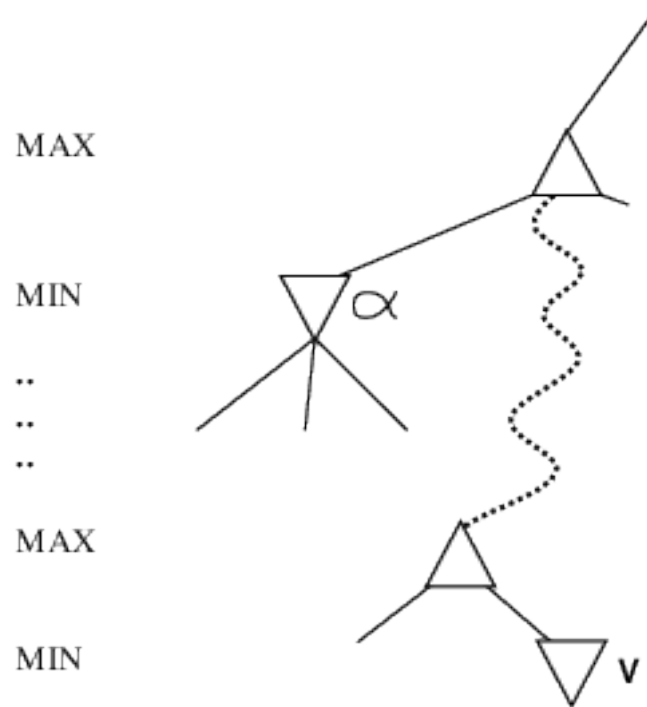
# $\alpha$ - $\beta$ Pruning Example



# $\alpha$ - $\beta$ Pruning Example



# Why is it Called $\alpha$ - $\beta$ ?



- $\alpha$  is the best value (to MAX) found so far off the current path
- If  $V$  is worse than  $\alpha$ , MAX will avoid it  $\Rightarrow$  prune that branch
- Define  $\beta$  similarly for MIN

# The $\alpha$ - $\beta$ Algorithm

**function** ALPHA-BETA-DECISION(*state*) **returns** an action  
**return** the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

---

**function** MAX-VALUE(*state*,  $\alpha$ ,  $\beta$ ) **returns** *a utility value*

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

$\alpha$ , the value of the best alternative for MAX along the path to *state*

$\beta$ , the value of the best alternative for MIN along the path to *state*

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

$v \leftarrow -\infty$

**for** *a*, *s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

**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*

same as MAX-VALUE but with roles of  $\alpha$ ,  $\beta$  reversed

# Properties of $\alpha$ - $\beta$



- Safe: Pruning **does not** affect final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering,” time complexity =  $O(b^{m/2})$   
⇒ **doubles** solvable depth
- A simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)
- Unfortunately,  $35^{50}$  is still impossible!

# Solved Games

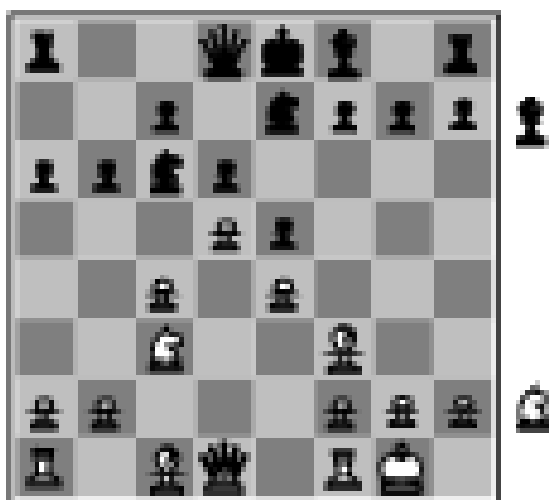
- A game is solved if optimal strategy can be computed
- Tic Tac Toe can be trivially solved
- Biggest solved game: Checkers
  - proof by Schaeffer in 2007
  - both players can force at least a draw
- Most games (Chess, Go, etc.) too complex to be solved

# resource limits

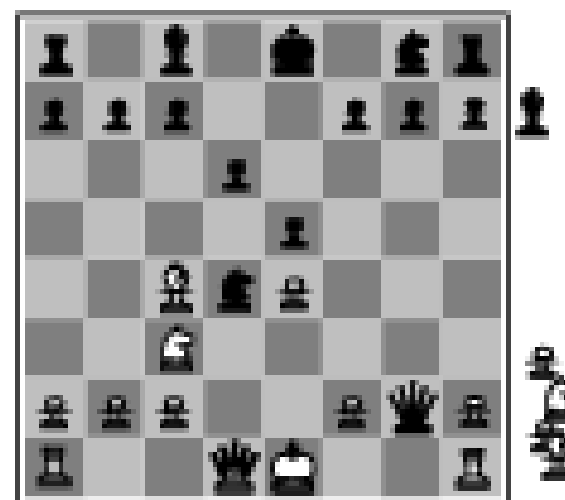
- Standard approach:
  - Use CUTOFF-TEST instead of TERMINAL-TEST  
e.g., depth limit (perhaps add **quiescence search**)
  - Use EVAL instead of UTILITY  
i.e., **evaluation function** that estimates desirability of position
- Suppose we have 100 seconds, explore  $10^4$  nodes/second
  - ⇒  $10^6$  nodes per move  $\approx 35^{8/2}$
  - ⇒  $\alpha$ - $\beta$  reaches depth 8 ⇒ pretty good Chess program



# Evaluation Functions



Black to move  
White slightly better



White to move  
Black winning

- For Chess, typically **linear** weighted sum of **features**

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

e.g.,  $f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$

# Evaluation Function for Chess



- Long experience of playing Chess
- ⇒ Evaluation of positions included in Chess strategy books
- bishop is worth 3 pawns
  - knight is worth 3 pawns
  - rook is worth 5 pawns
  - good pawn position is worth 0.5 pawns
  - king safety is worth 0.5 pawns
  - etc.
- Pawn count → weight for features

# Learning Evaluation Functions

- Designing good evaluation functions requires a lot of expertise
- Machine learning approach
  - collect a large database of games play
  - note for each game who won
  - try to predict game outcome from features of position⇒ learned weights
- May also learn evaluation functions from self-play

# Some Concerns



- Quiescence
  - position evaluation not reliable if board is unstable
  - e.g., Chess: queen will be lost in next move
  - deeper search of game-changing moves required
- Horizon Effect
  - adverse move can be delayed, but not avoided
  - search may prefer to delay, even if costly

# Forward Pruning



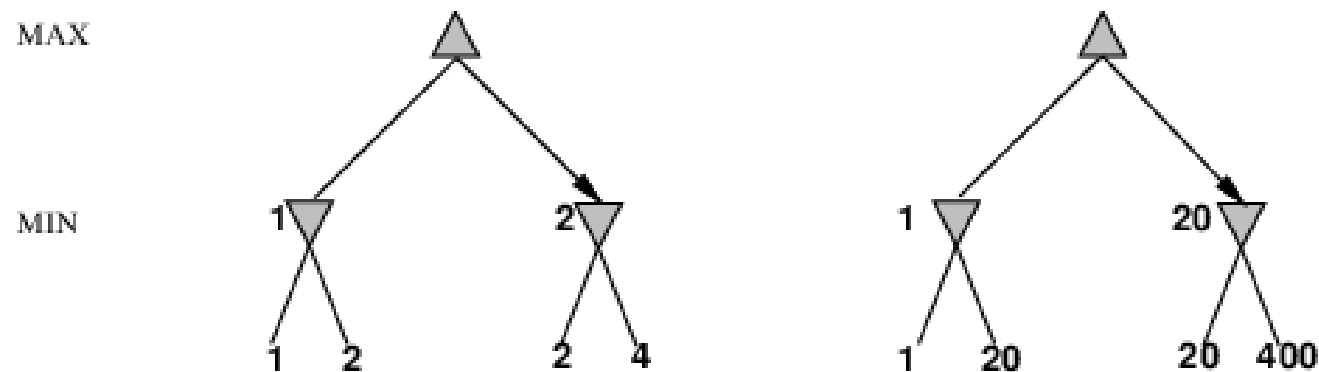
- Idea: avoid computation on clearly bad moves
- Cut off searches with bad positions before they reach max-depth
- Risky: initially inferior positions may lead to better positions
- Beam search: explore fixed number of promising moves deeper

# Lookup instead of Search



- Library of opening moves
  - even expert Chess players use standard opening moves
  - these can be memorized and followed until divergence
- End game
  - if only few pieces left, optimal final moves may be computed
  - Chess end game with 6 pieces left solved in 2006
  - can be used instead of evaluation function

# Digression: Exact Values do not Matter



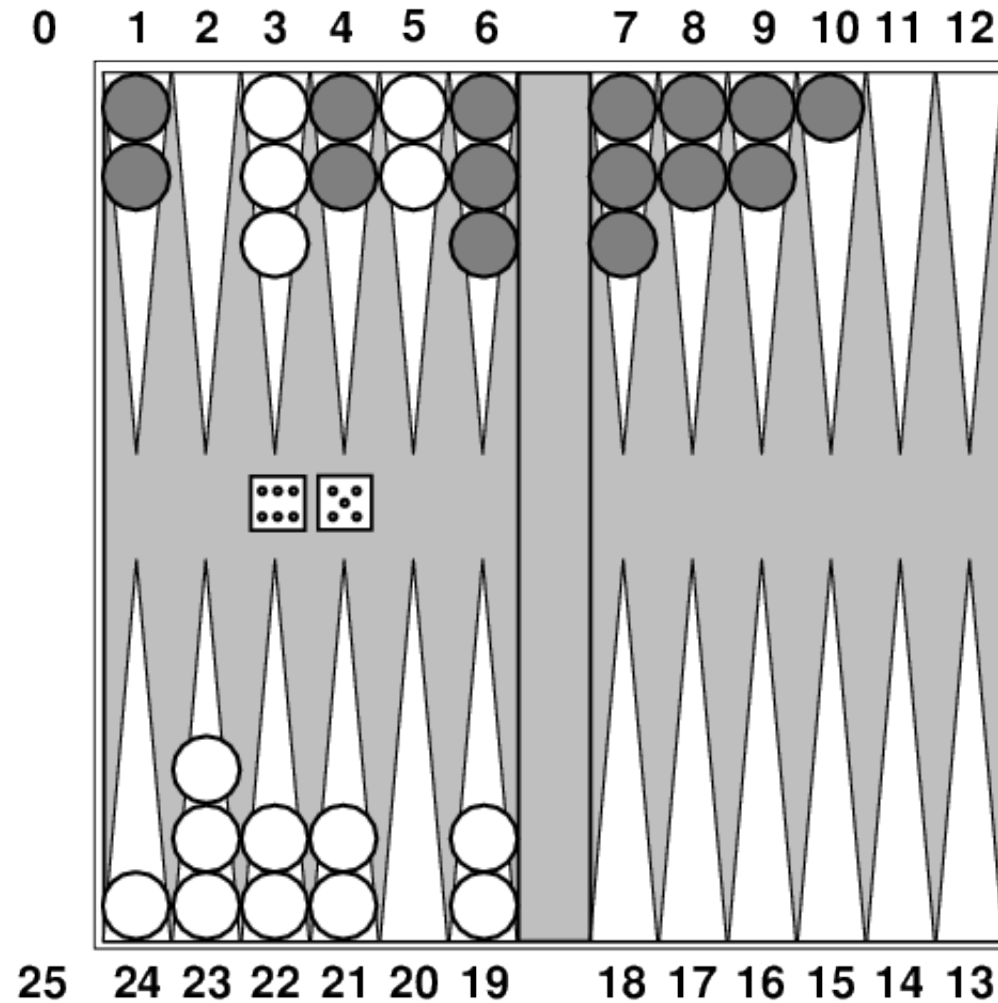
- Behaviour is preserved under any **monotonic** transformation of EVAL
- Only the order matters:  
payoff in deterministic games acts as an **ordinal utility** function

- **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. Weakly solved in 2007 by Schaeffer (guaranteed draw).
- **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.
- **Othello:** human champions refuse to compete against computers, who are too good.
- **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.



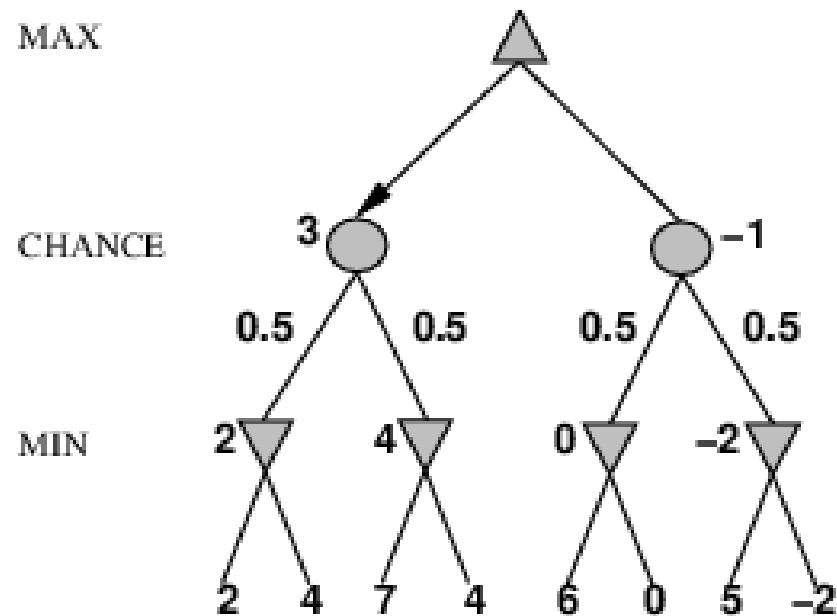
# games of chance

# Nondeterministic Games: Backgammon



# Nondeterministic Games in General

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping:

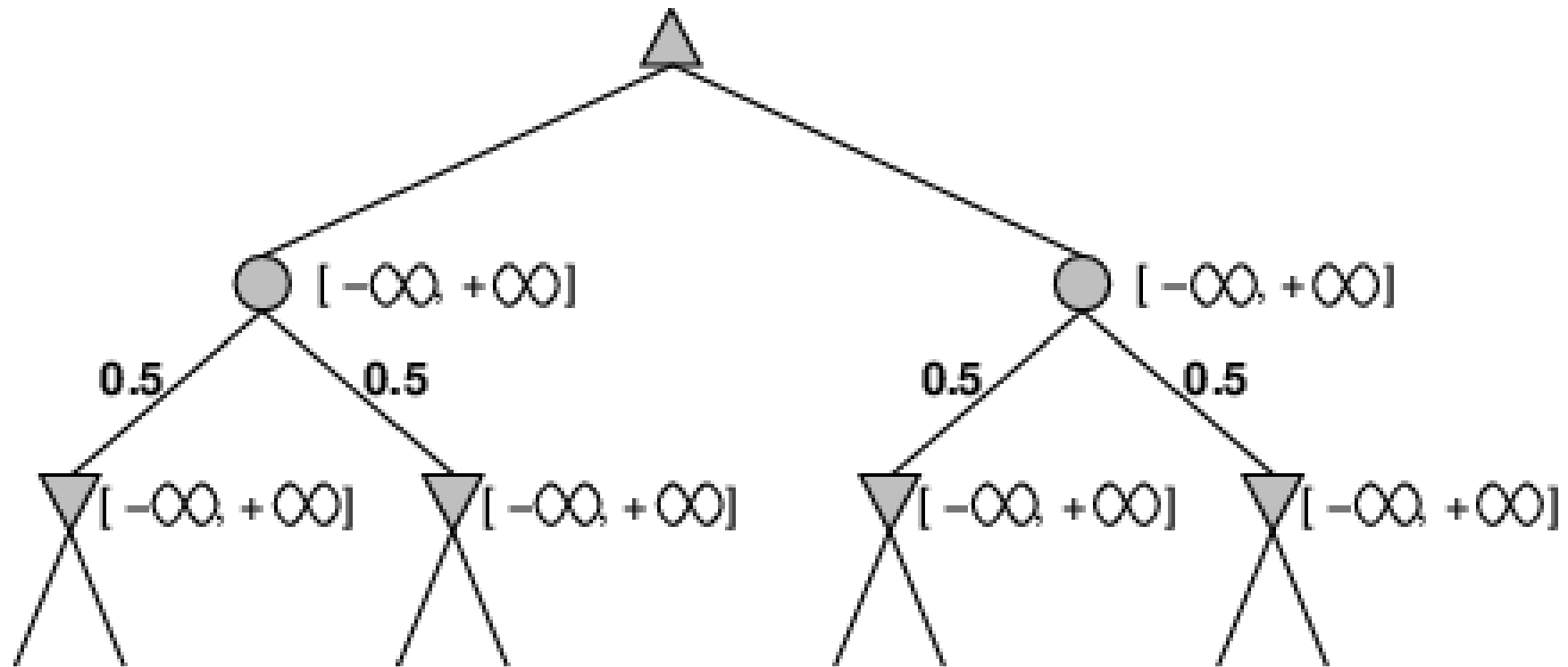


# Algorithm for Nondeterministic Games

- EXPECTIMINIMAX gives perfect play
- Just like MINIMAX, except we must also handle chance nodes:
  - ...
  - if** *state* is a MAX node **then**
    - return** the highest EXPECTIMINIMAX-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 EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
  - ...

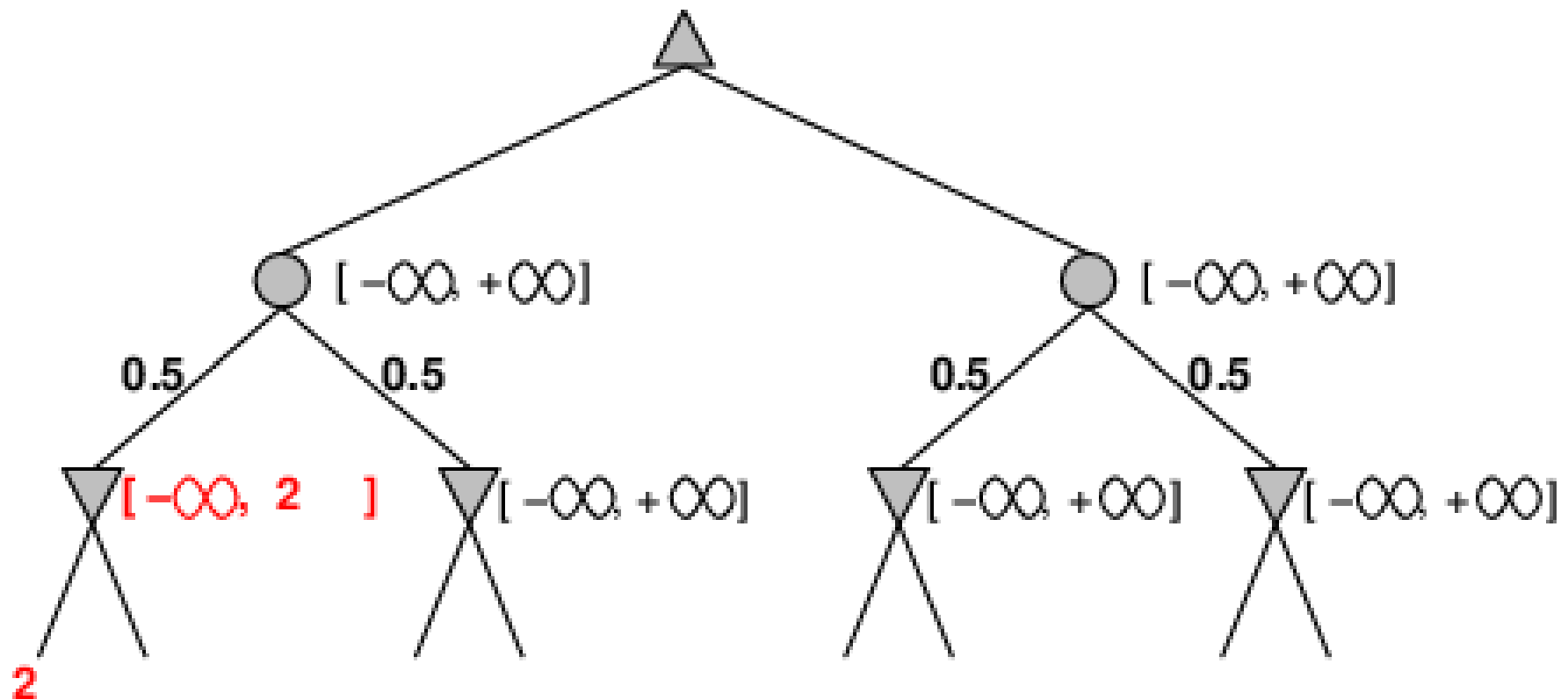
# Pruning in Nondeterministic Game Trees

A version of  $\alpha$ - $\beta$  pruning is possible:



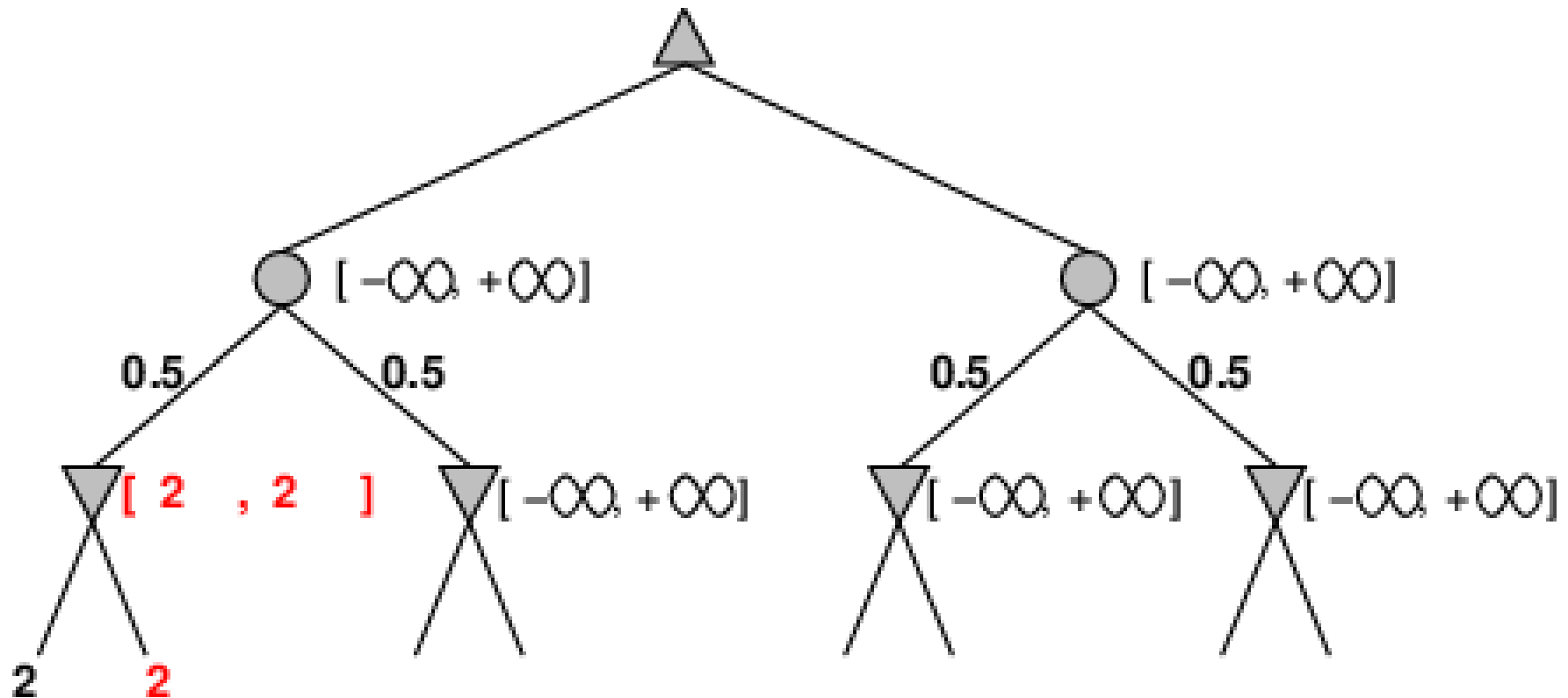
# Pruning in Nondeterministic Game Trees

A version of  $\alpha$ - $\beta$  pruning is possible:



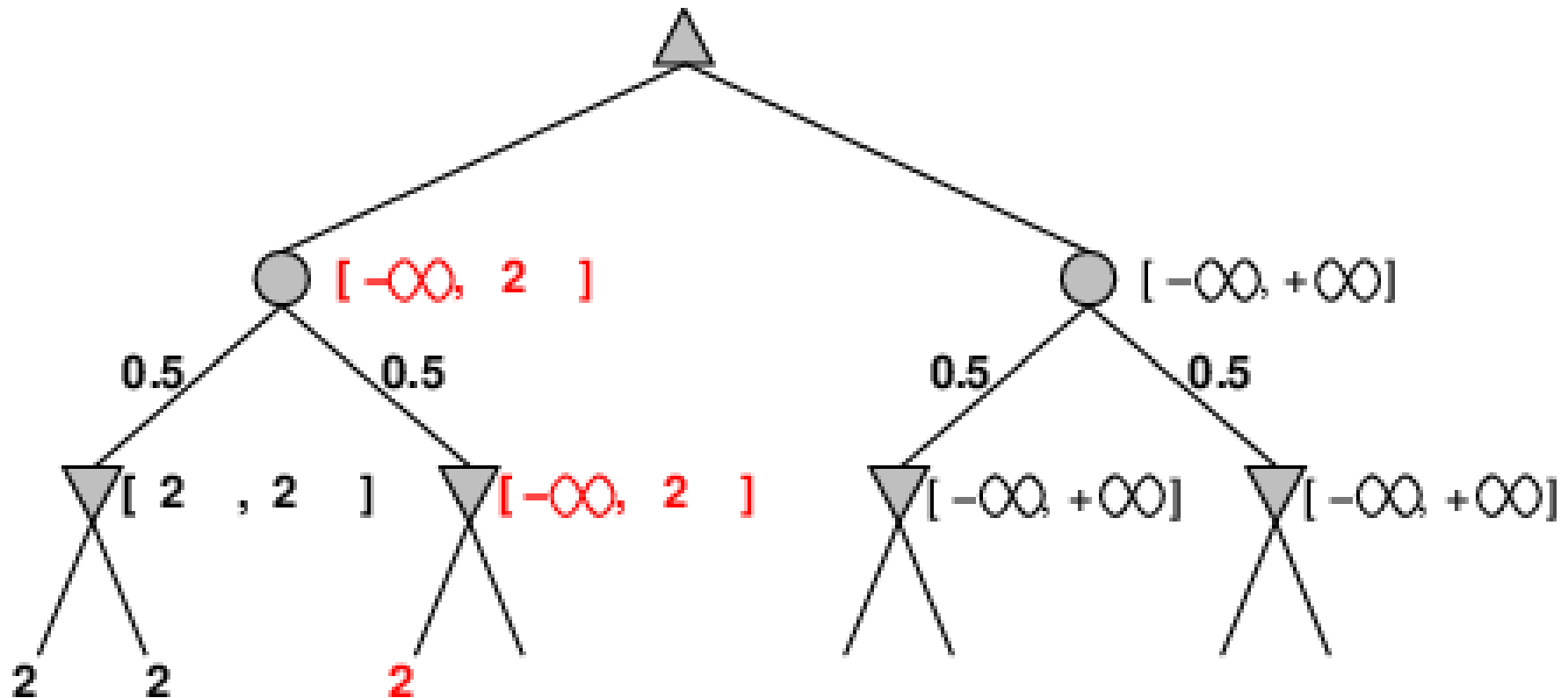
# Pruning in Nondeterministic Game Trees

A version of  $\alpha$ - $\beta$  pruning is possible:



# Pruning in Nondeterministic Game Trees

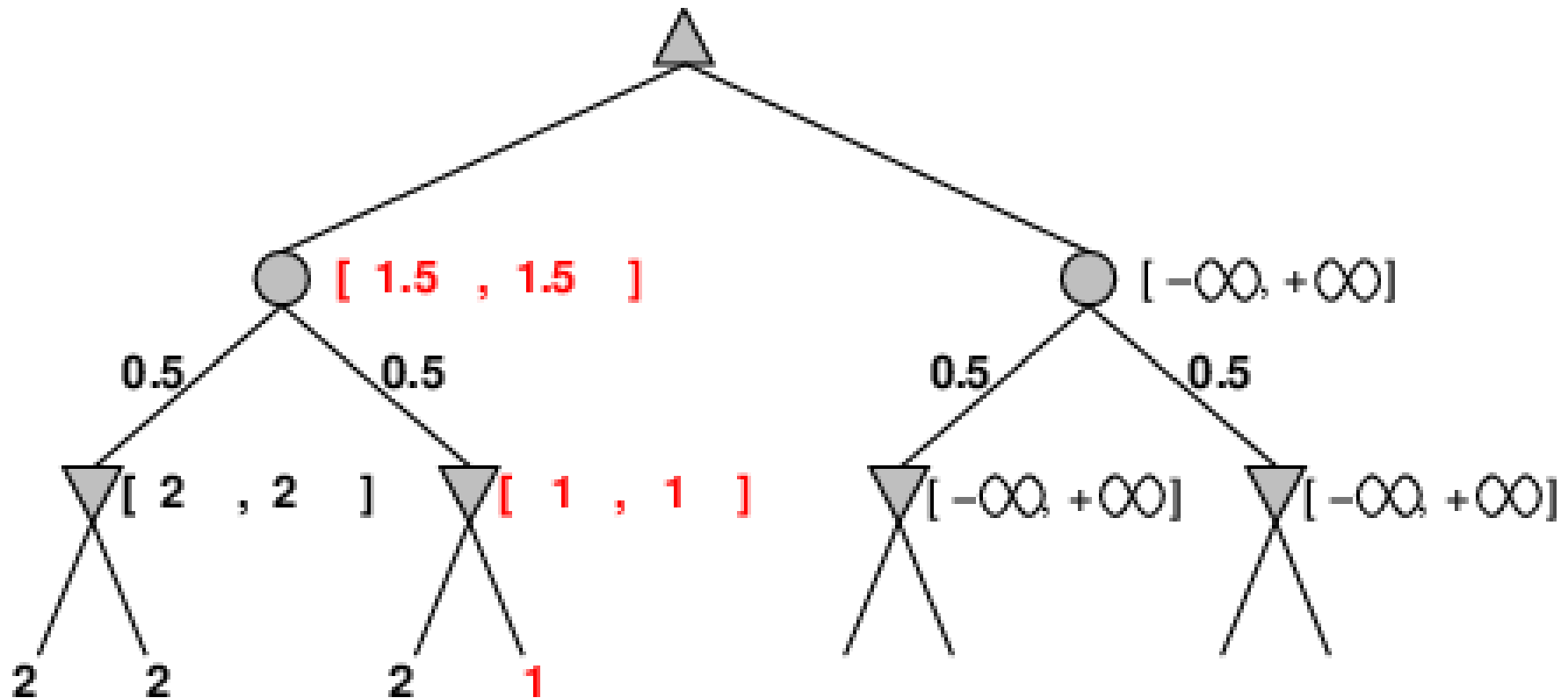
A version of  $\alpha$ - $\beta$  pruning is possible:





# Pruning in Nondeterministic Game Trees

A version of  $\alpha$ - $\beta$  pruning is possible:

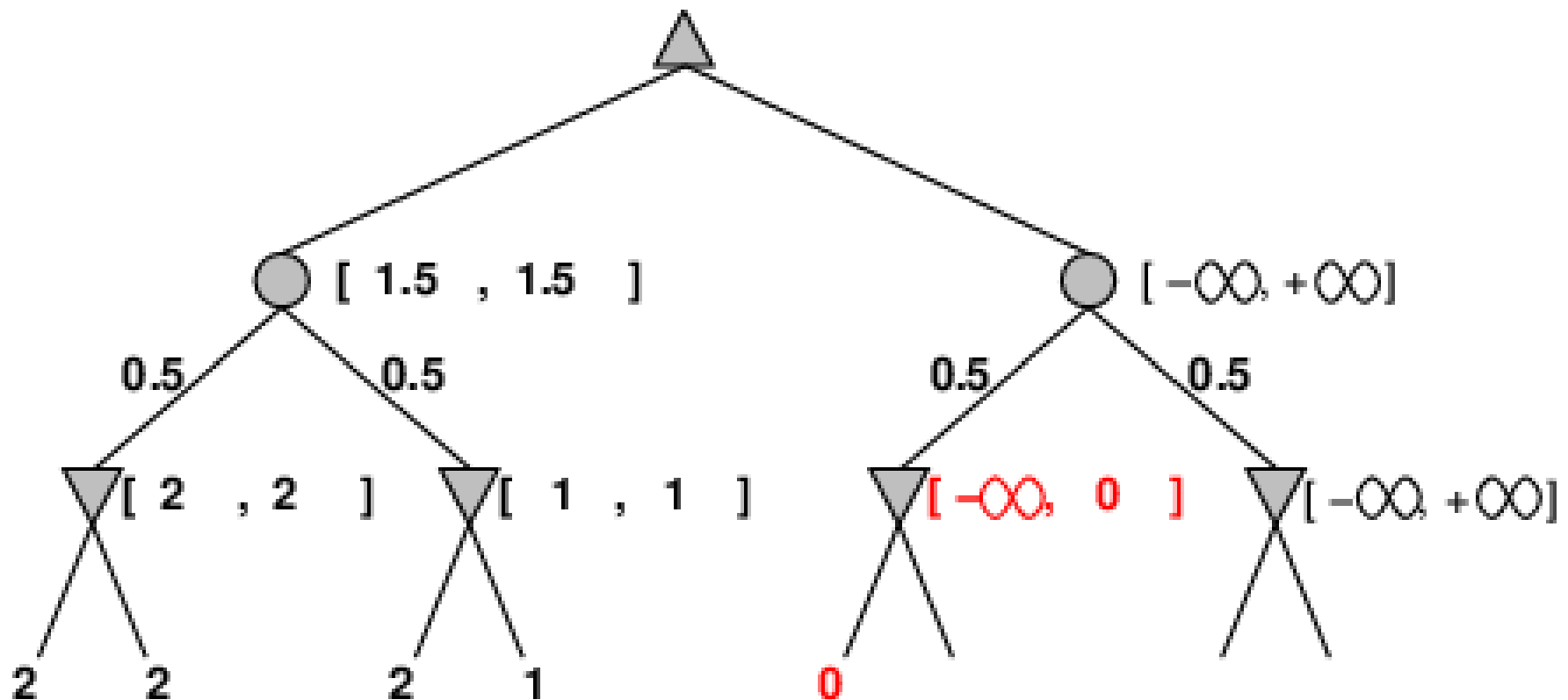


# Pruning in Nondeterministic Game Trees

41

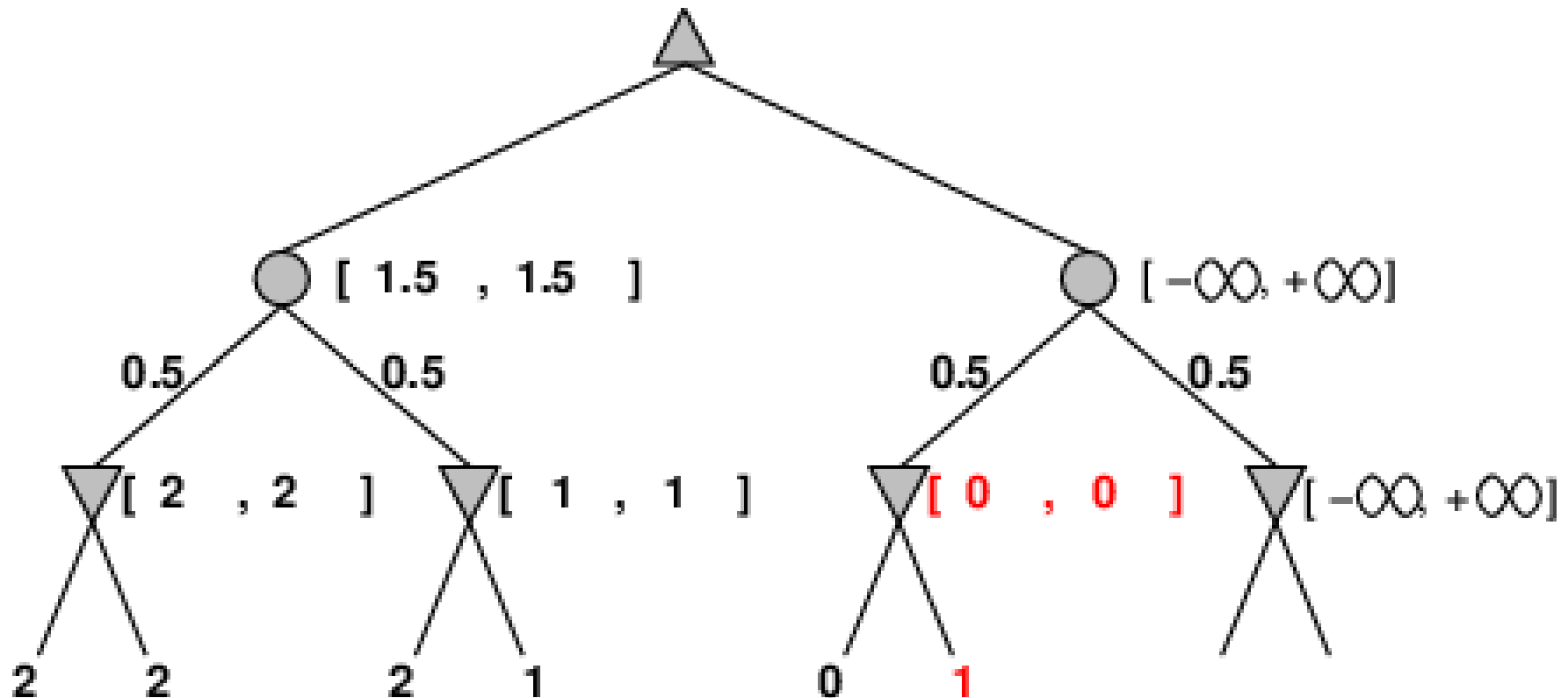


A version of  $\alpha$ - $\beta$  pruning is possible:



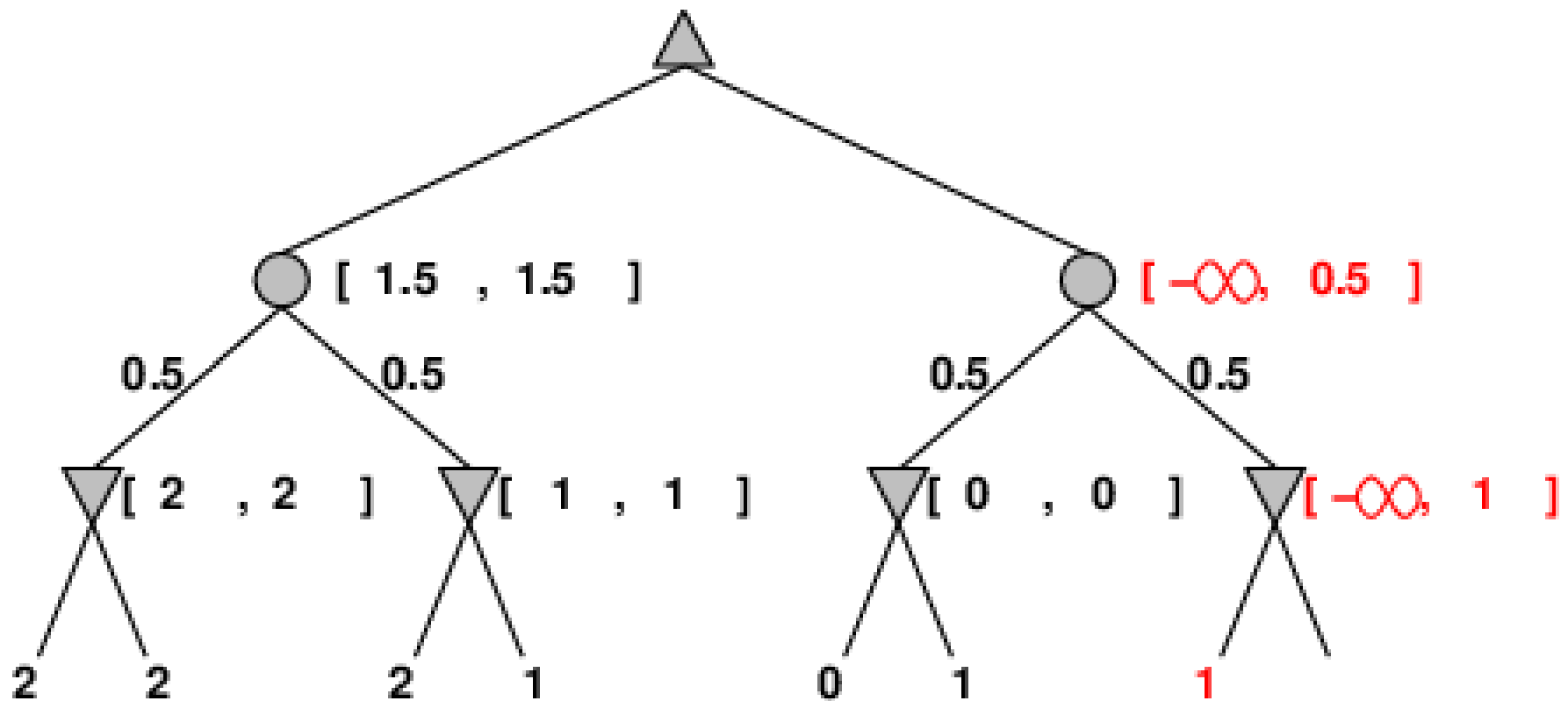
# Pruning in Nondeterministic Game Trees

A version of  $\alpha$ - $\beta$  pruning is possible:



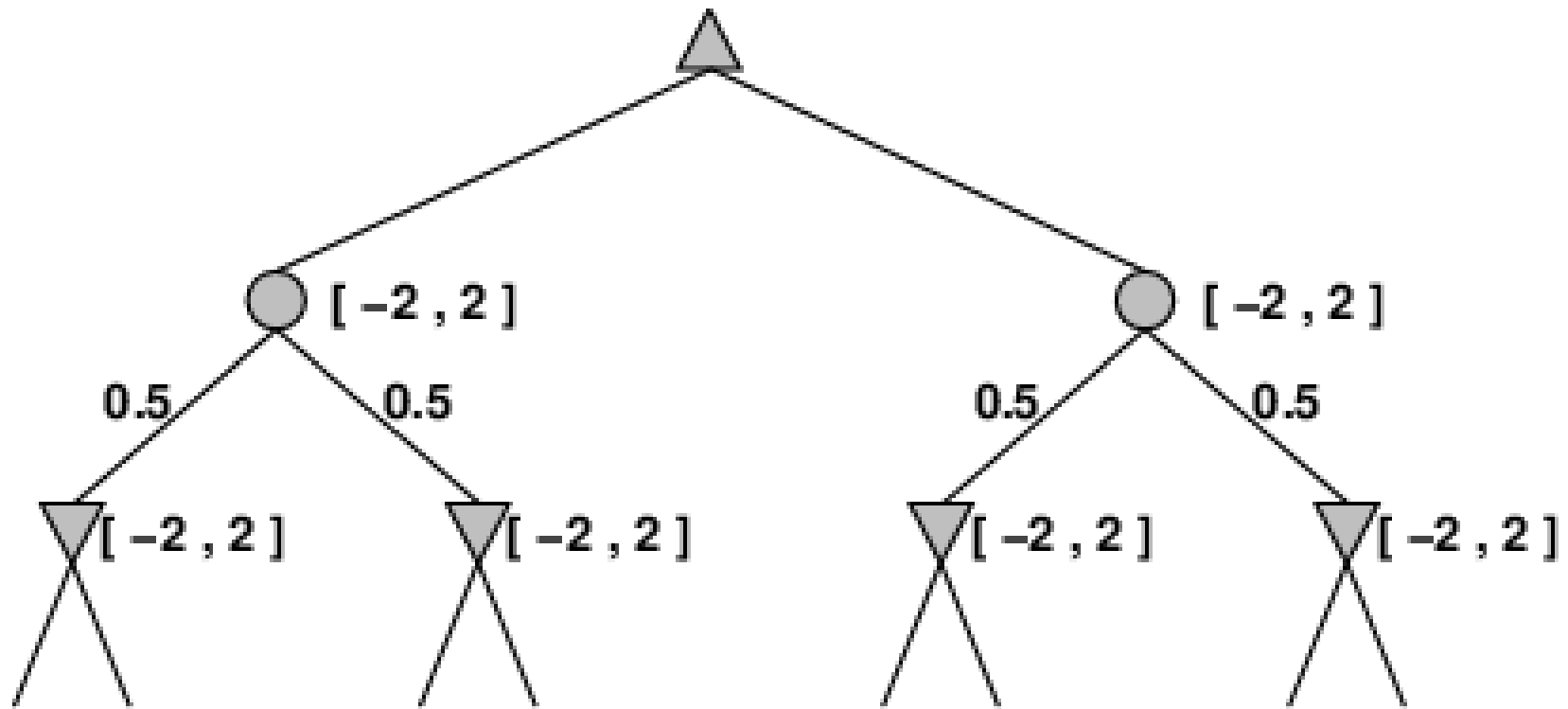
# Pruning in Nondeterministic Game Trees

Terminate, since right path will be worth on average.



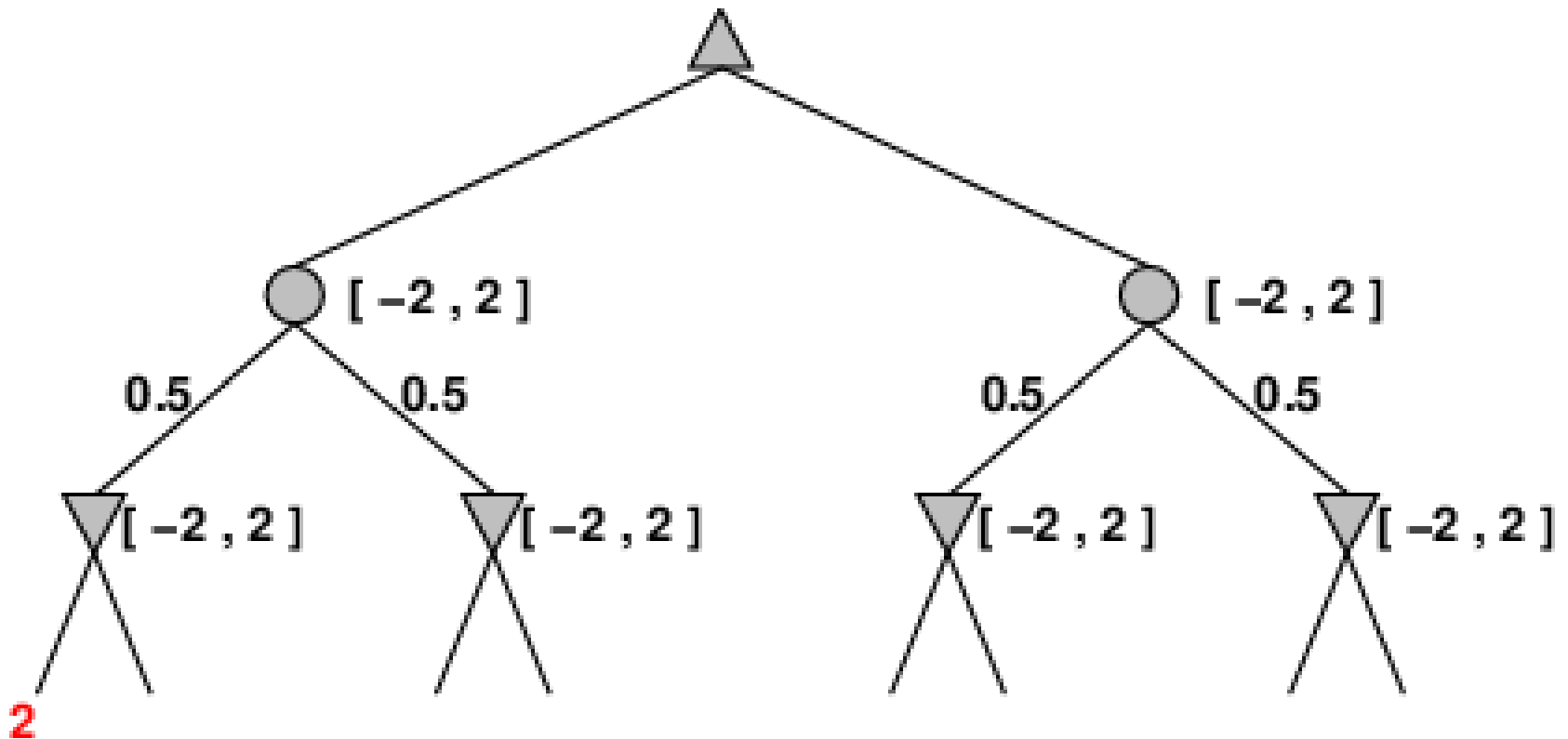
# Pruning with Bounds

More pruning occurs if we can bound the leaf values (0,1,2)



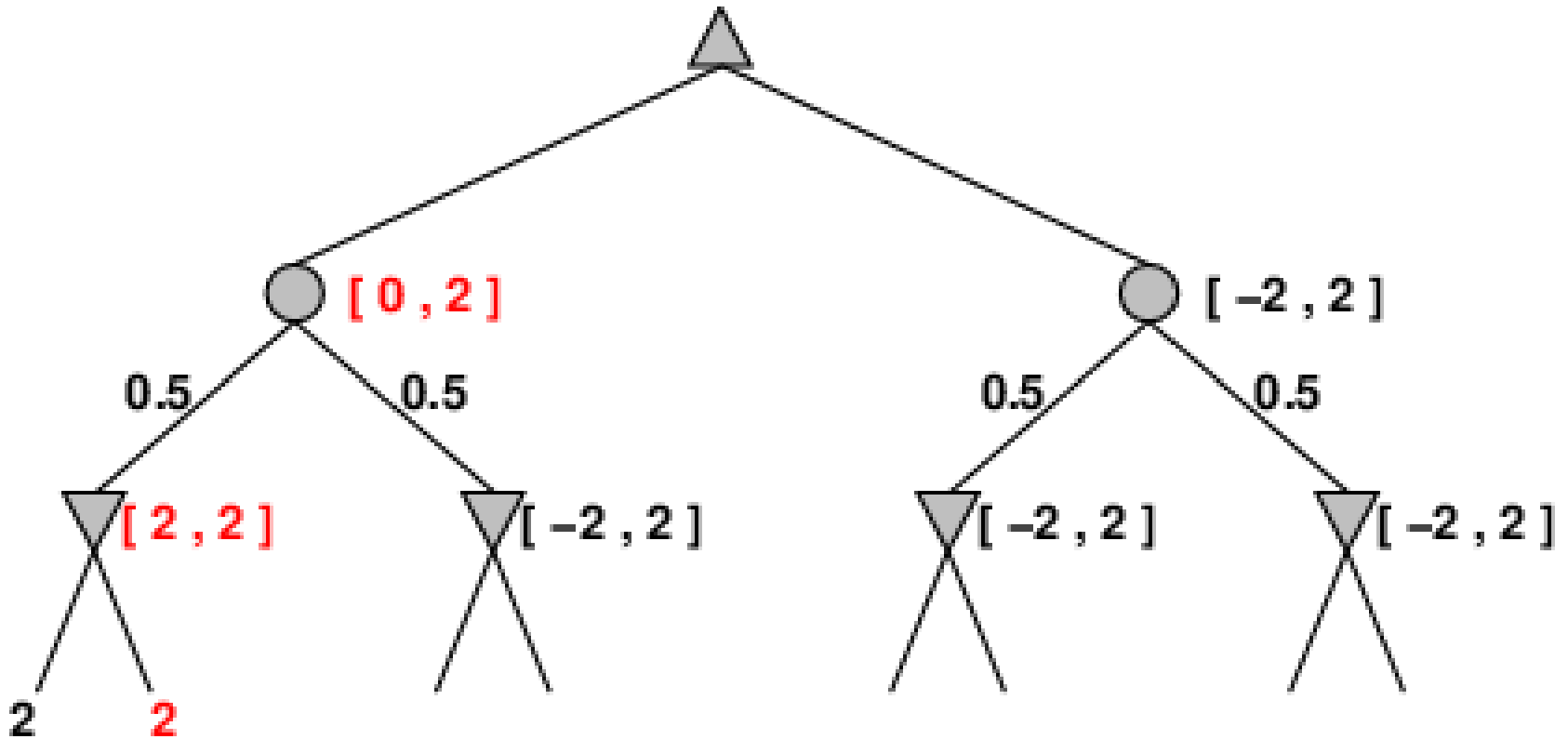
# Pruning with Bounds

More pruning occurs if we can bound the leaf values (0,1,2)



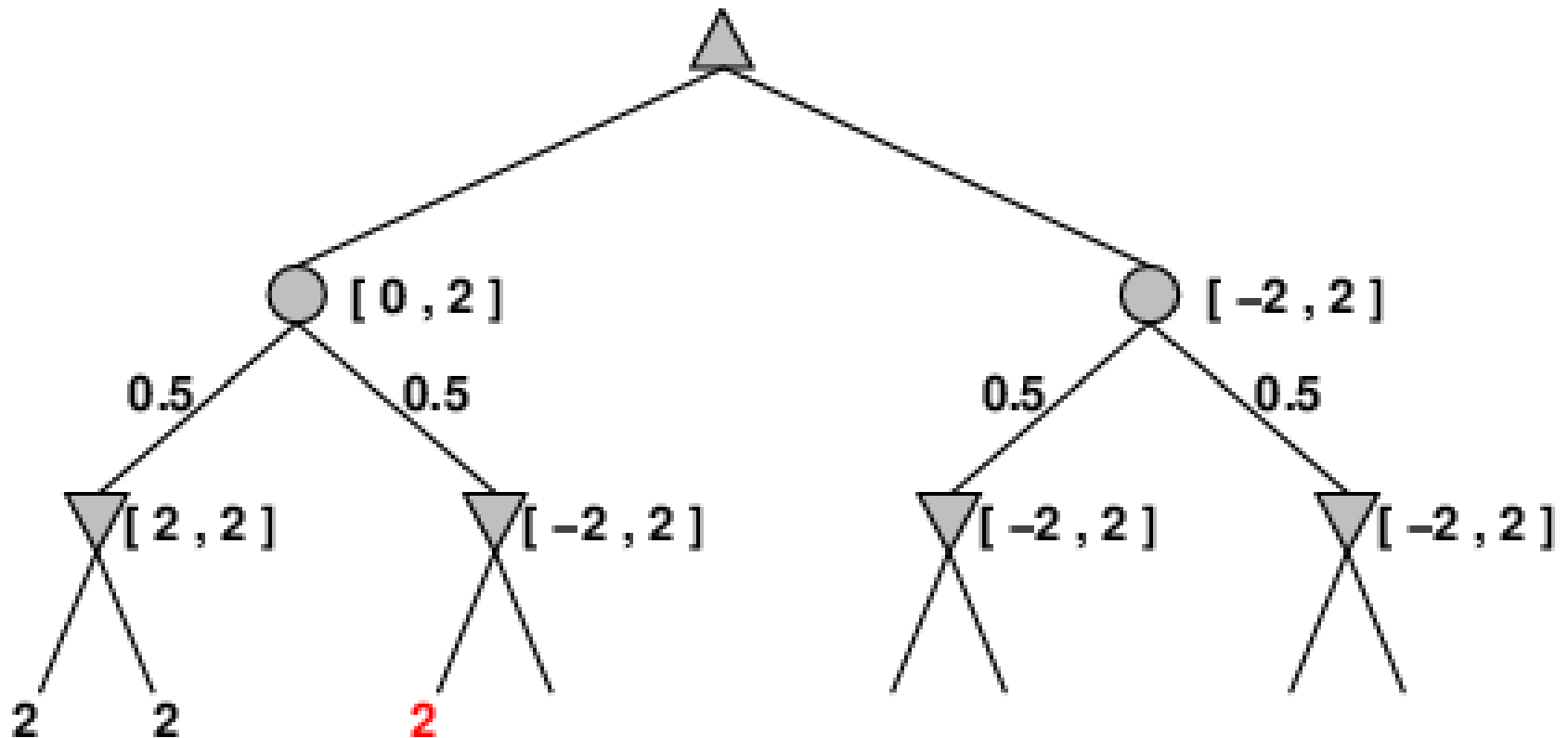
# Pruning with Bounds

More pruning occurs if we can bound the leaf values (0,1,2)



# Pruning with Bounds

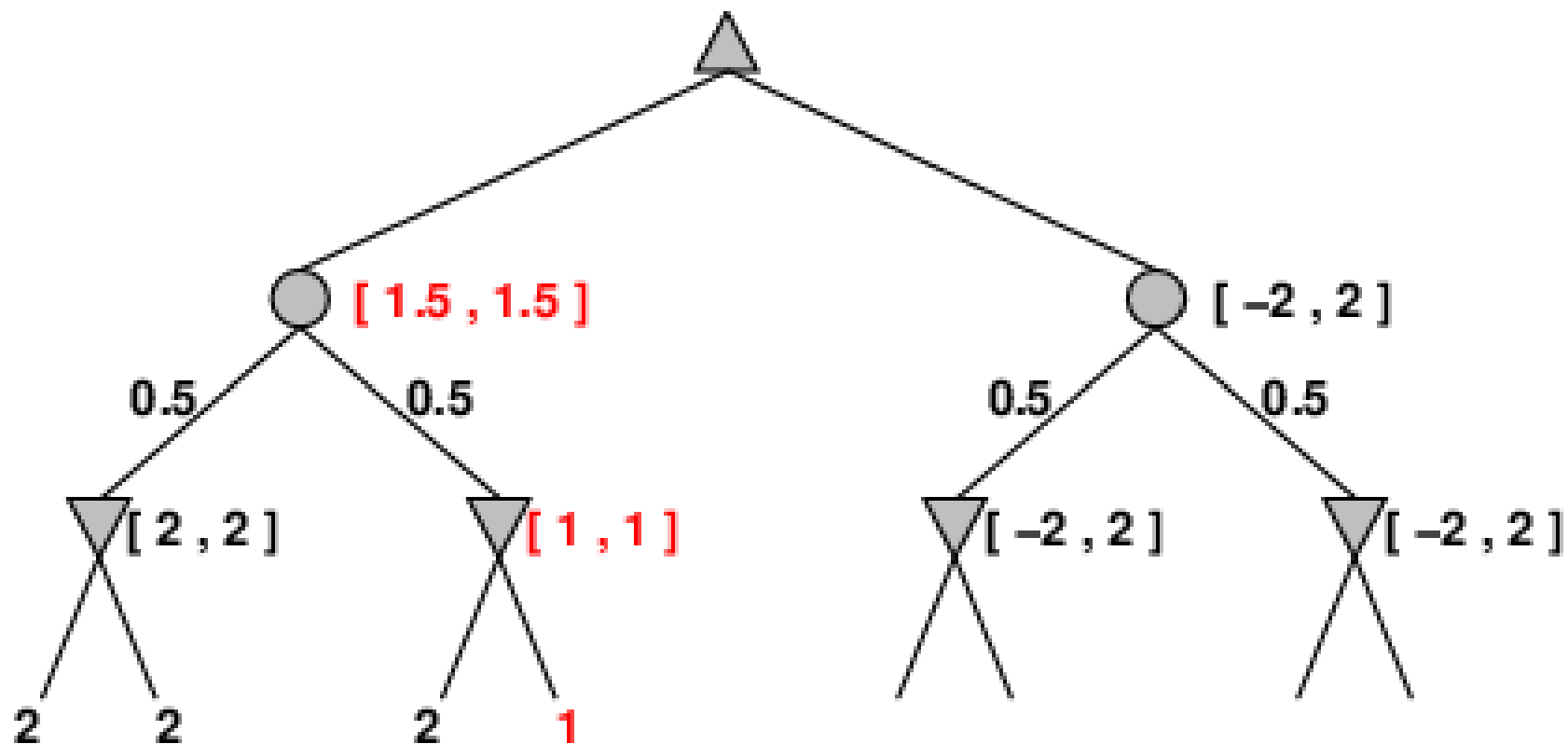
More pruning occurs if we can bound the leaf values (0,1,2)





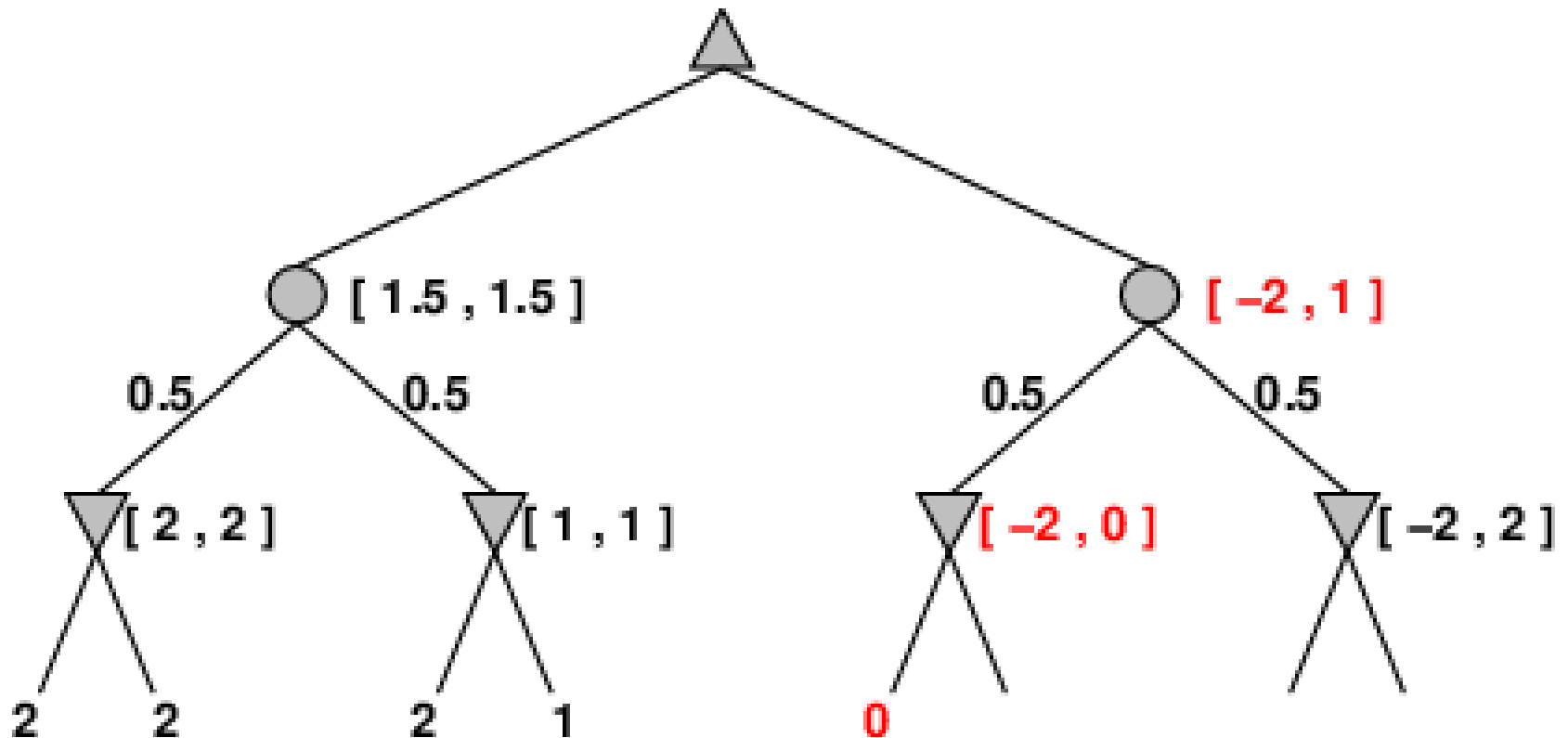
# Pruning with Bounds

More pruning occurs if we can bound the leaf values (0,1,2)



# Pruning with Bounds

More pruning occurs if we can bound the leaf values (0,1,2)



# Nondeterministic Games in Practice

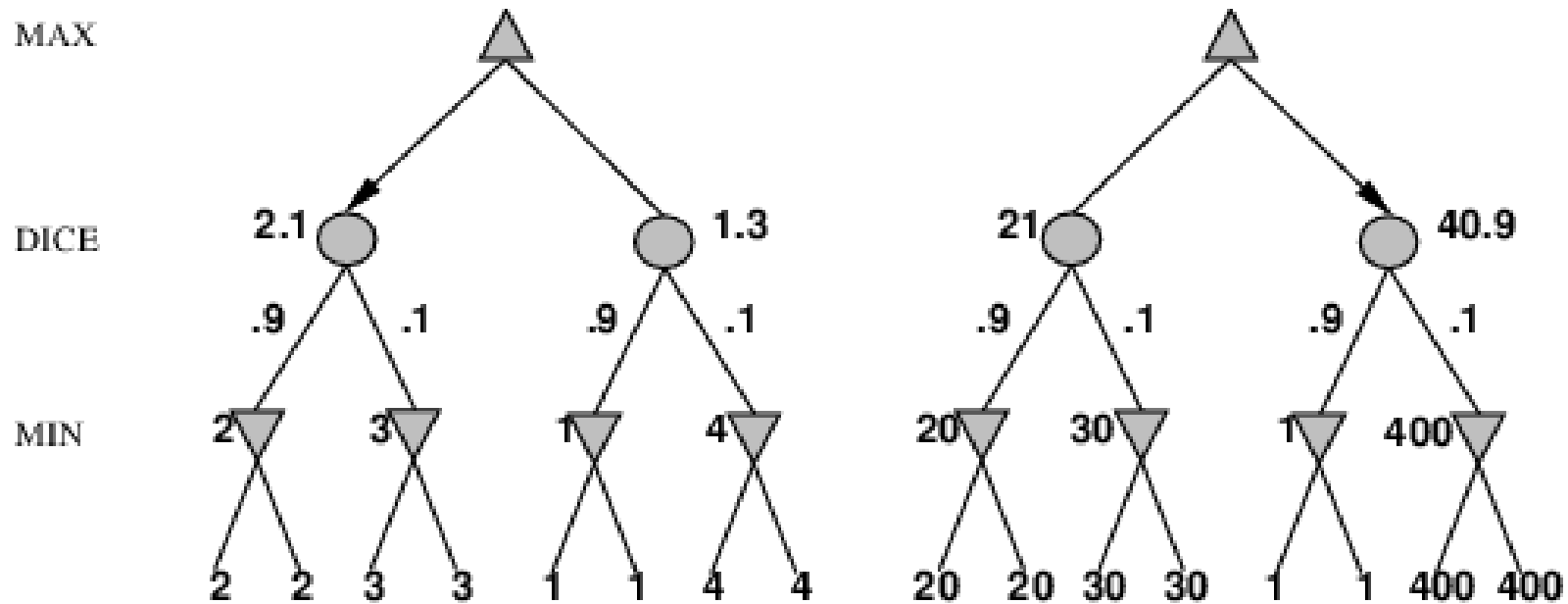


- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
- Backgammon  $\approx$  20 legal moves (can be 6,000 with 1-1 roll)

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

- As depth increases, probability of reaching a given node shrinks  
 $\Rightarrow$  value of lookahead is diminished
- $\alpha$ - $\beta$  pruning is much less effective
- 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

# imperfect information

# Games of Imperfect Information



- E.g., card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- **Idea:** compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
- Special case: if an action is optimal for all deals, it's optimal.

# Commonsense Counter-Example

- Road A leads to a small heap of gold pieces  
Road B leads to a fork:  
take the left fork and you'll find a mound of jewels;  
take the right fork and you'll be run over by a bus.■
- Road A leads to a small heap of gold pieces  
Road B leads to a fork:  
take the left fork and you'll be run over by a bus;  
take the right fork and you'll find a mound of jewels.■
- Road A leads to a small heap of gold pieces  
Road B leads to a fork:  
guess correctly and you'll find a mound of jewels;  
guess incorrectly and you'll be run over by a bus.

# Proper Analysis



- Intuition that the value of an action is the average of its values in all actual states is **WRONG**
- With partial observability, value of an action depends on the **information state** or **belief state** the agent is in
- Can generate and search a tree of information states
- Leads to rational behaviors such as
  - acting to obtain information
  - signalling to one's partner
  - acting randomly to minimize information disclosure



# Computer Poker

- Hard game
    - imperfect information — including bluffing and trapping
    - stochastic outcomes — cards drawn at random
    - partially observable — may never see other players hand when they fold
    - non-cooperative multi-player — possibility for coalitions
  - Few moves (fold, call, raise), but large number of stochastic states
  - Relative balance of deception plays very important  
also: when to bluff
- ⇒ There is no single best move
- Need to model other players (style, collusion, patterns)
  - Hard to evaluate (not just win/loss, different types of opponents)

# Summary



- Games are fun to work on
- They illustrate several important points about AI
  - perfection is unattainable  $\Rightarrow$  must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state
- Games are to AI as grand prix racing is to automobile design