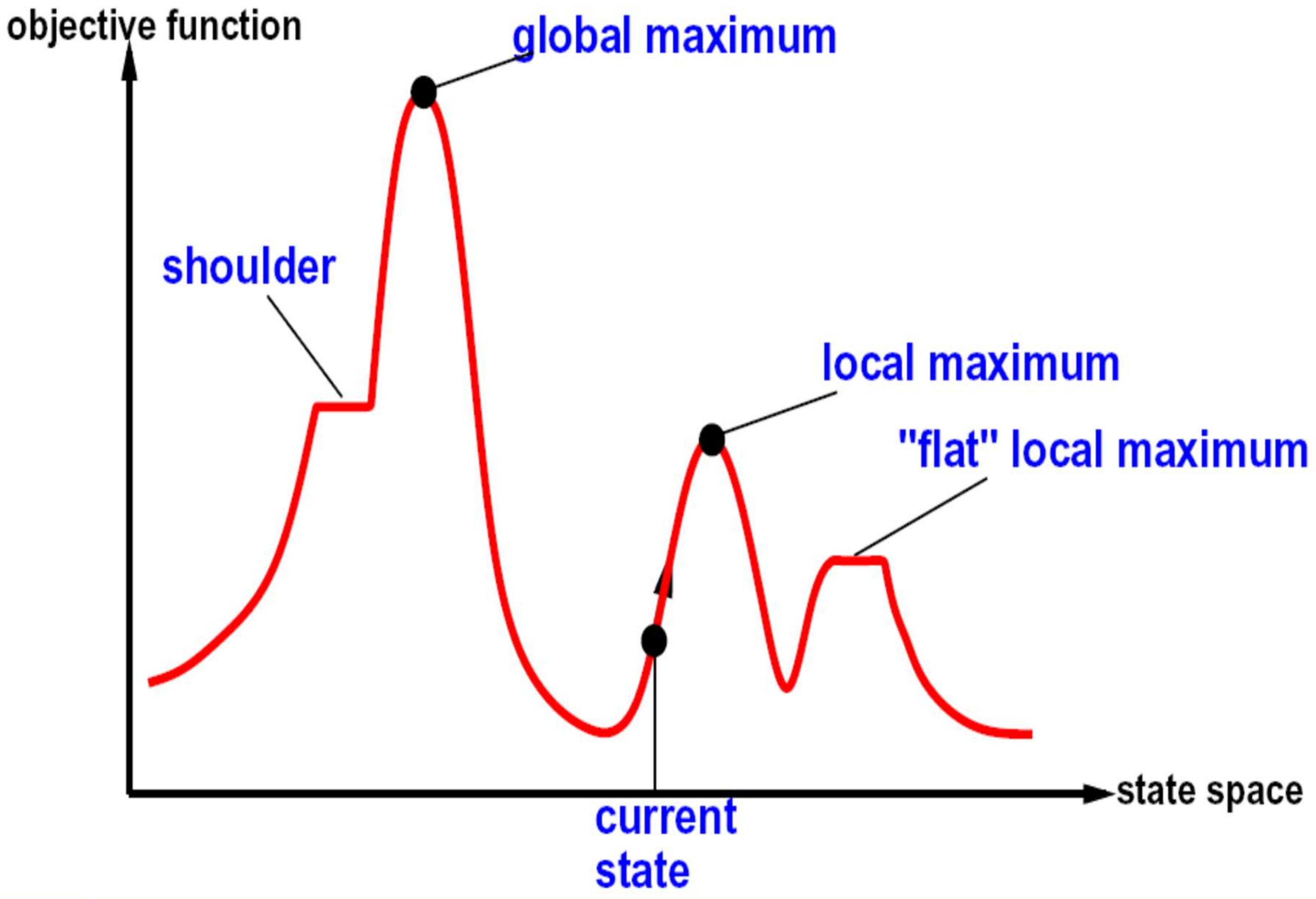
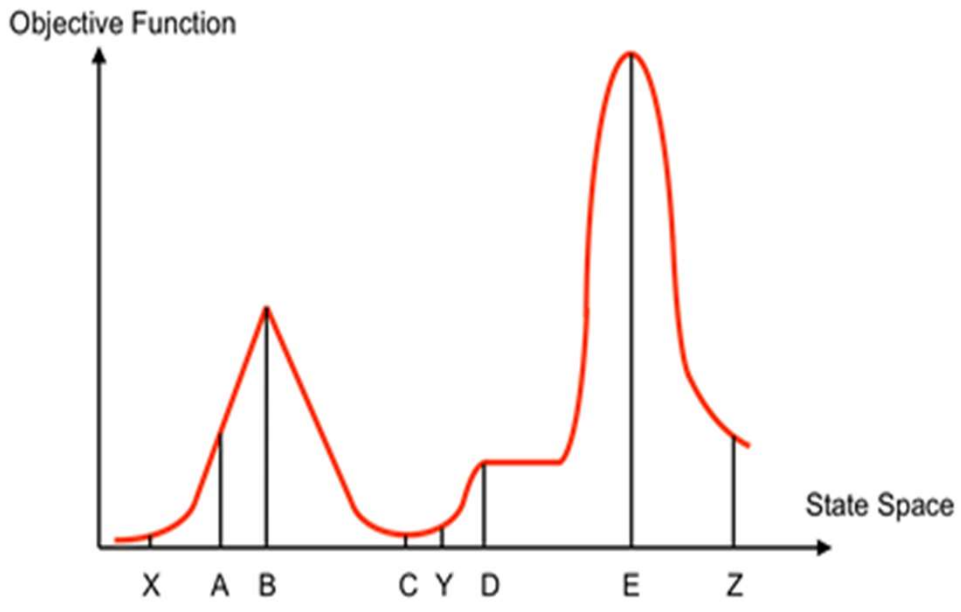
CS 4810 Artificial Intelligence
CS 6810 Topics in Artificial Intelligence

Hill Climbing Diagram



Hill Climbing Quiz

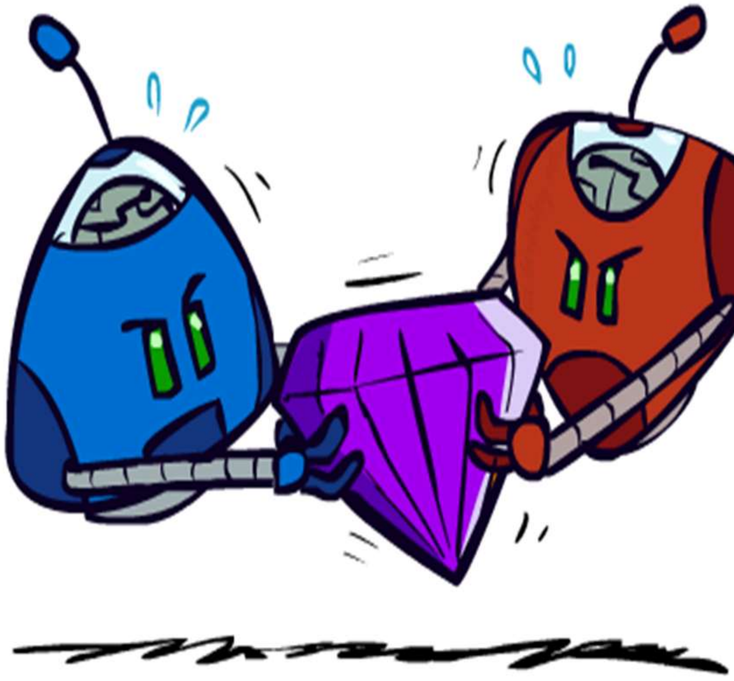


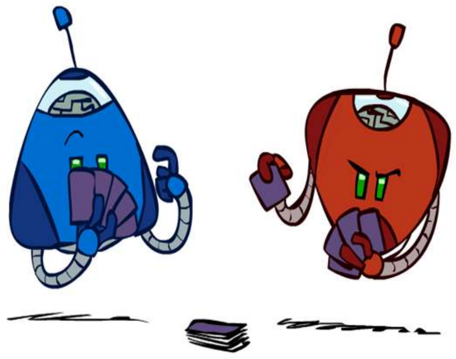
Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Adversarial Games

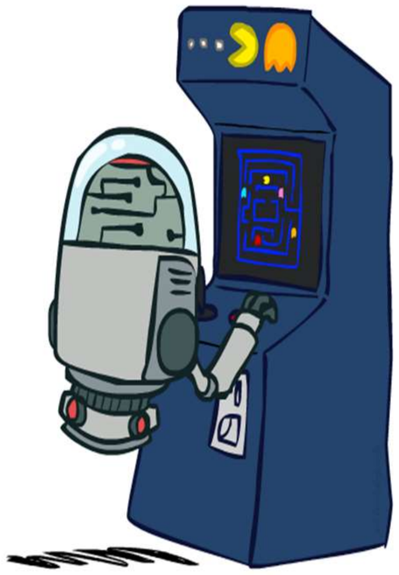




Types of Games

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move from each state

Deterministic Games

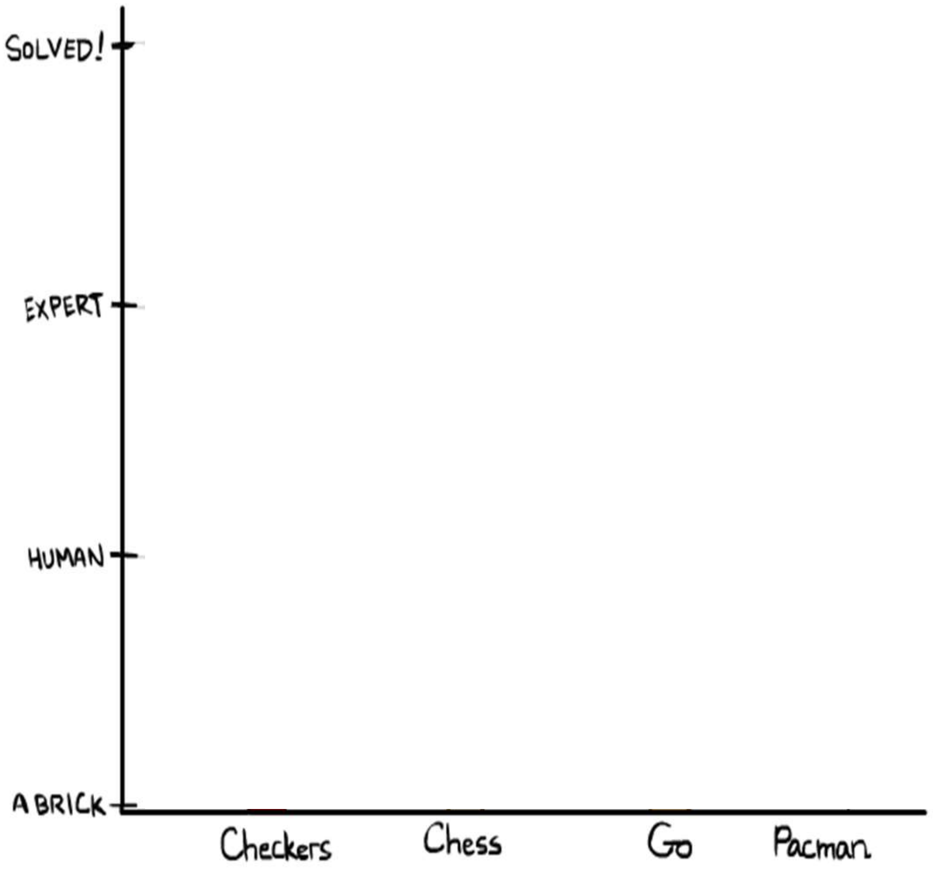


- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: $SxA \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $SxP \rightarrow R$

- Solution for a player is a policy: $S \rightarrow A$

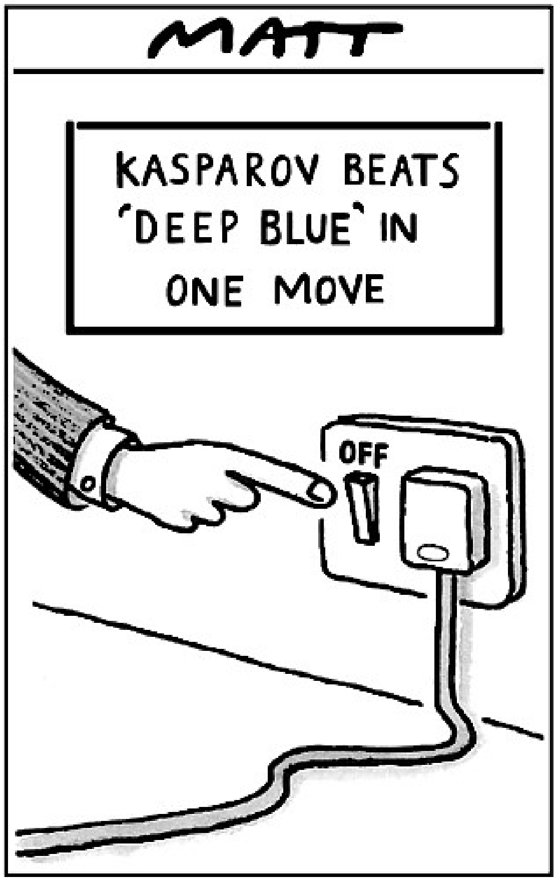
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go:**, b > 300! Classic programs use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.
- **Pacman**



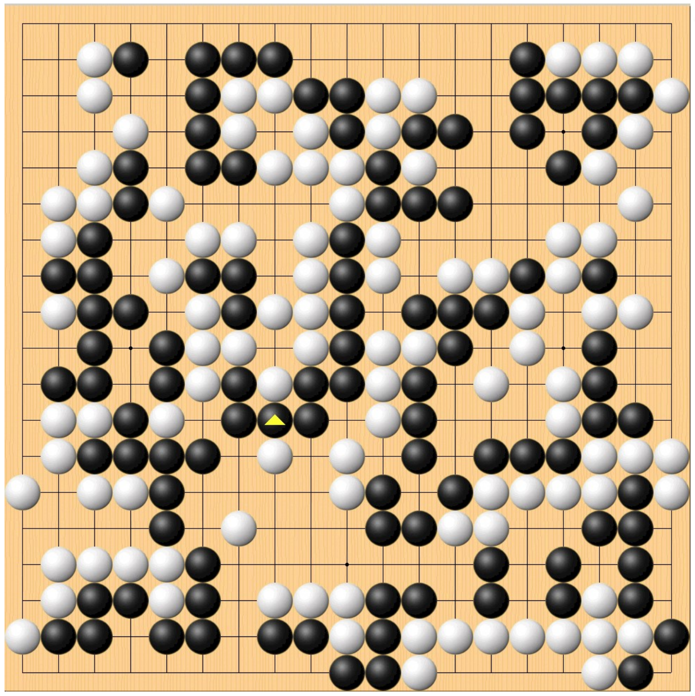
Games and adversarial search (Chapter 5)

World Champion chess player Garry Kasparov is defeated by IBM's Deep Blue chess-playing computer in a six-game match in May, 1997 (link)



© Telegraph Group Unlimited 1997

AlphaGo vs. Ke Jie, 3:0



Game 3, May 27, 2017

Ke Jie was ranked 1st among all human players worldwide under Rémi Coulom's ranking system, and had held that position since late 2014.
Ke Jie was also ranked number one in the world under Korea Baduk Association's, Japan Go Association's, and Chinese Weiqi Association's ranking systems.

Go vs. Chess

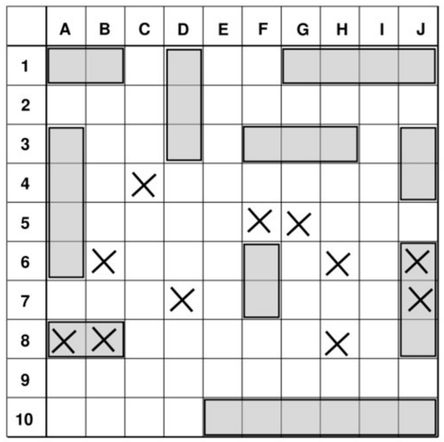
	Board Size	Branching Factor
Go	19*19=361	≈250
Chess	8*8=64	≈35

Why study games?

- Games are a traditional hallmark of intelligence
- Games are easy to formalize
- Games can be a good model of real-world competitive or cooperative activities
  - Military confrontations, negotiation, auctions, etc.

Types of game environments

	Deterministic	Stochastic
Perfect information (fully observable)	Chess, checkers, go	Backgammon, monopoly
Imperfect information (partially observable)	Battleships	Scrabble, poker, bridge



Battleships

Backgammon

Bridge

Alternating two-player zero-sum games

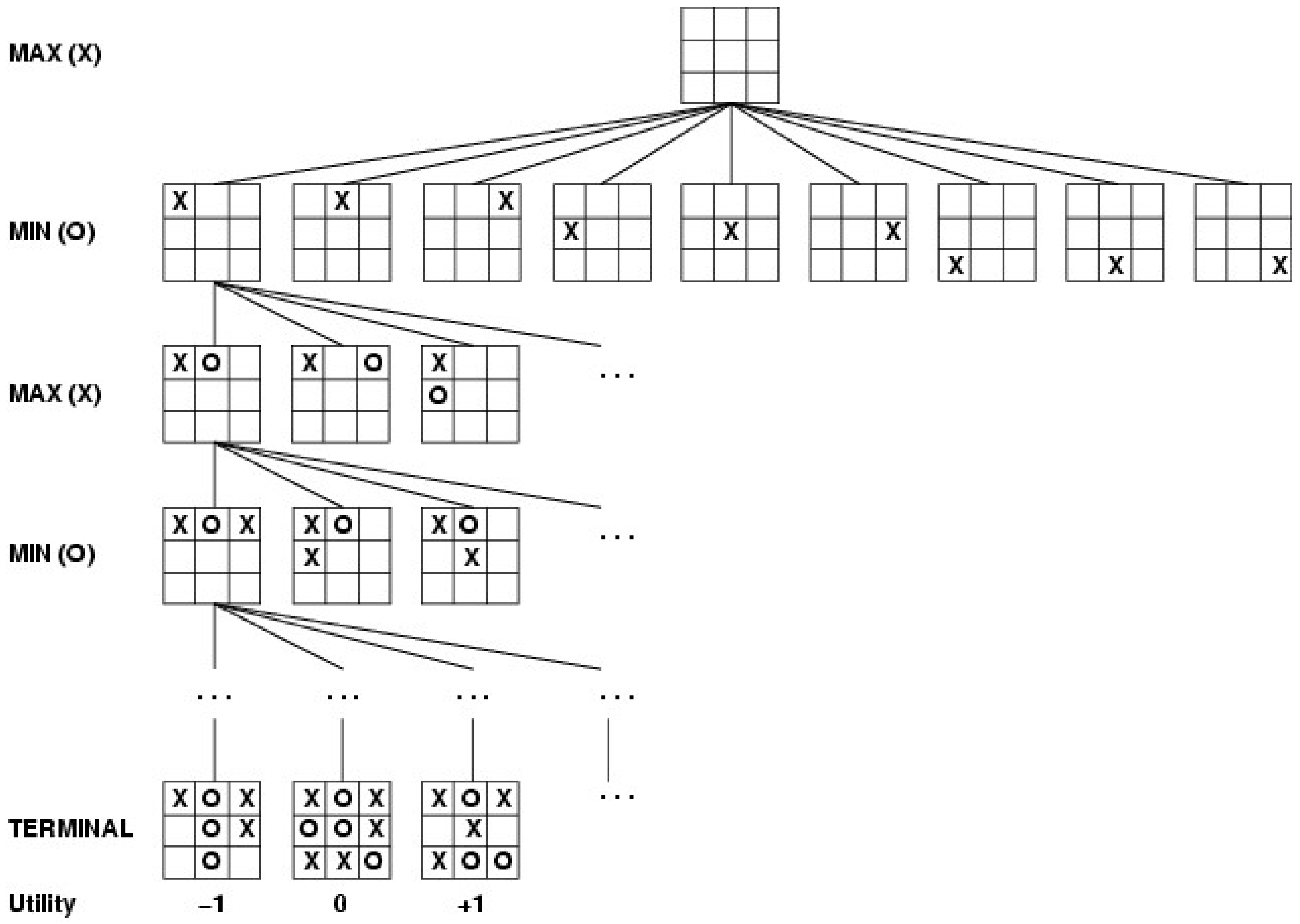
- Players take turns
- Each game outcome or **terminal state** has a **utility** for each player (e.g., 1 for win, 0 for loss)
- The sum of both players' utilities is a constant

Games vs. single-agent search

- We don’t know how the opponent will act
    - The solution is not a fixed sequence of actions from start state to goal state, but a ***strategy*** or ***policy*** (a mapping from state to best move in that state)
- Efficiency is critical to playing well
    - The time to make a move is limited
    - The branching factor, search depth, and number of terminal configurations are huge
        - In chess, branching factor ≈ 35 and depth ≈ 100, giving a search tree of 10^{154} nodes
            - Number of atoms in the observable universe ≈ 10^{80}
    - This rules out searching all the way to the end of the game

Game tree

- A game of tic-tac-toe between two players, “max” and “min”

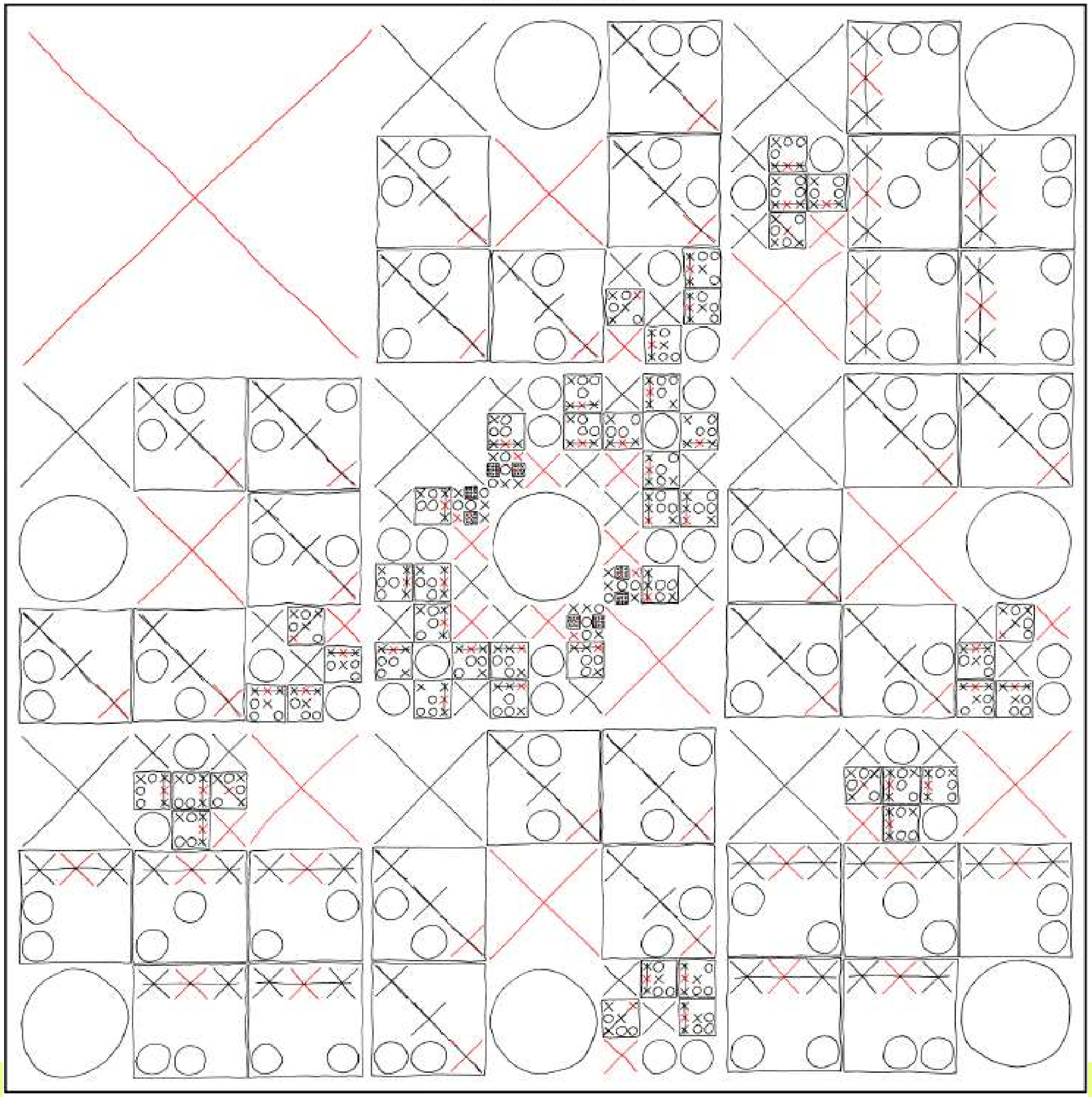


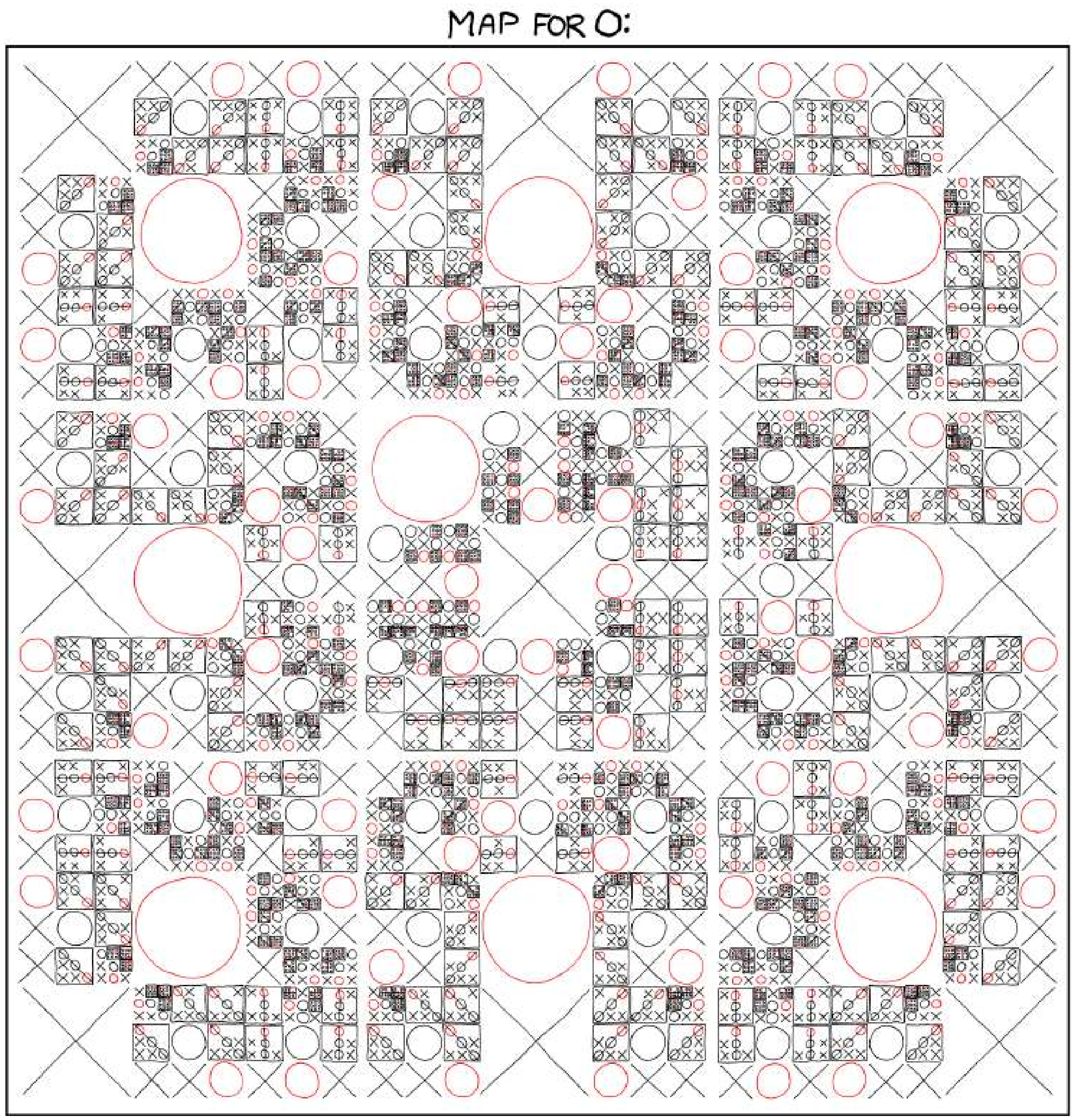
COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

http://xkcd.com/832/

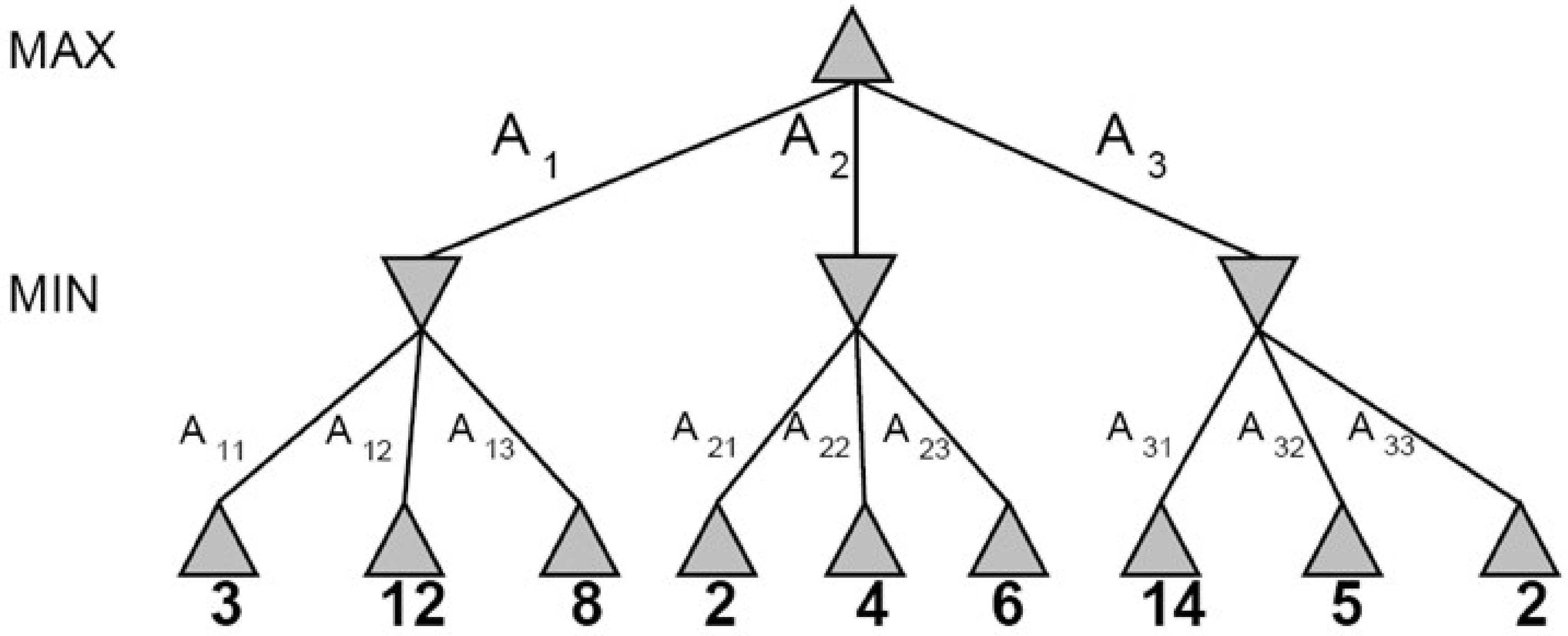
MAP FOR X:





http://xkcd.com/832/

A more abstract game tree



Terminal utilities (for MAX)

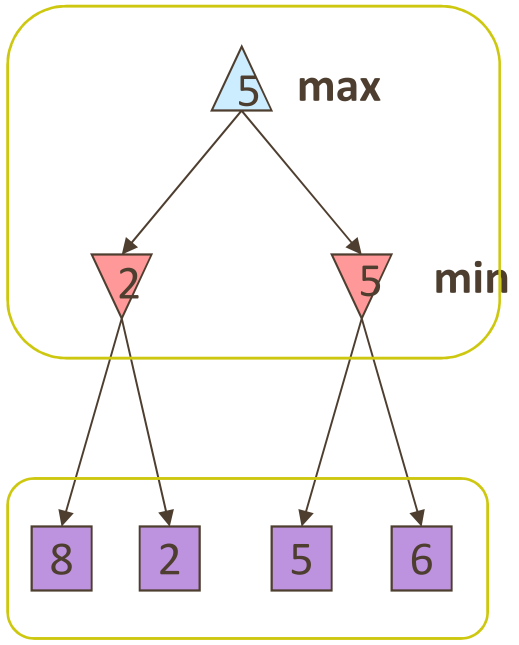
A *two-ply* game

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary

Minimax values: computed recursively



Terminal values: part of the game

Minimax Implementation

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

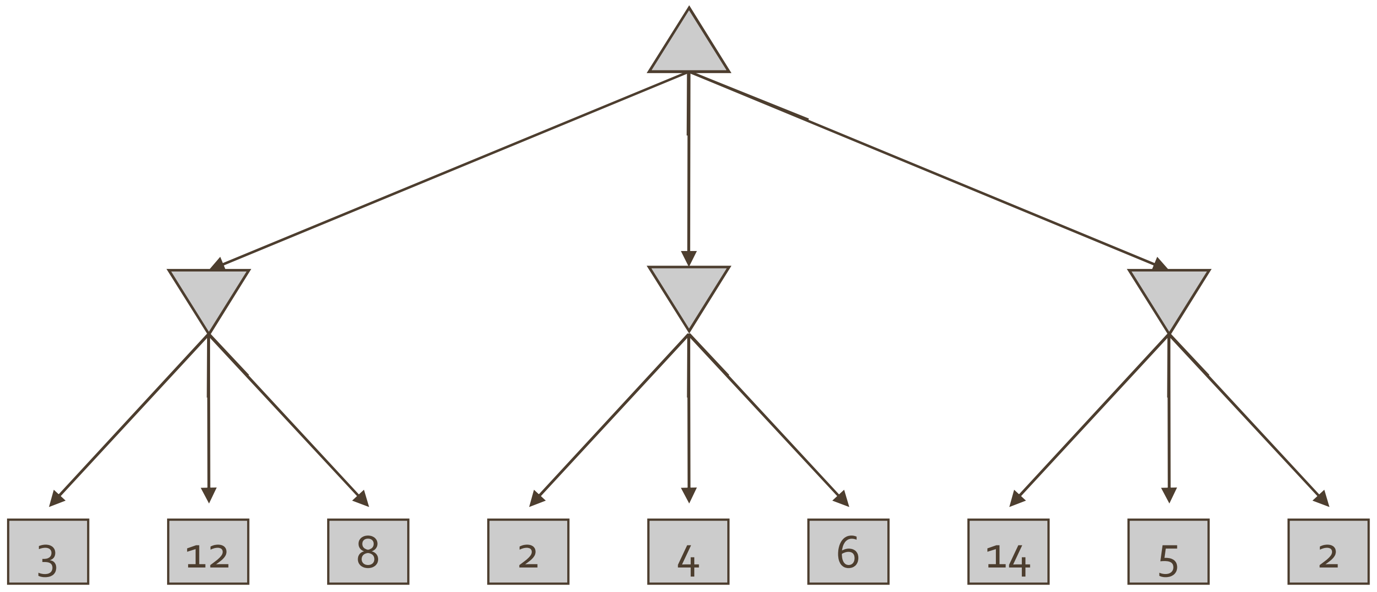
Minimax Implementation (Dispatch)

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

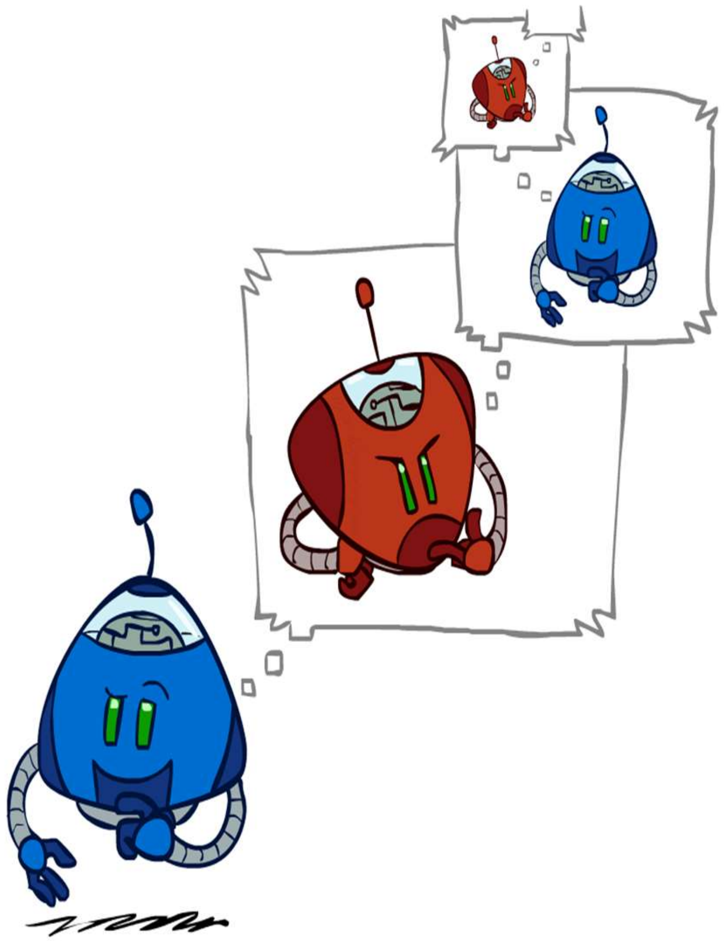
Minimax Example



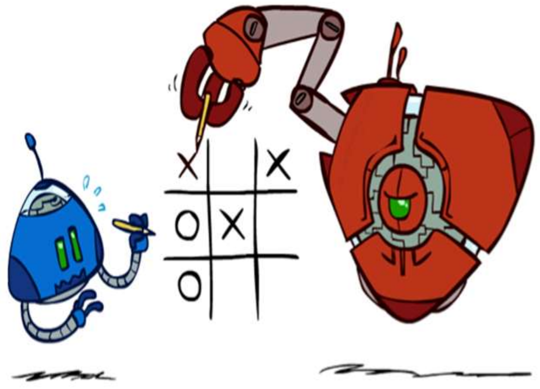
Minimax Efficiency

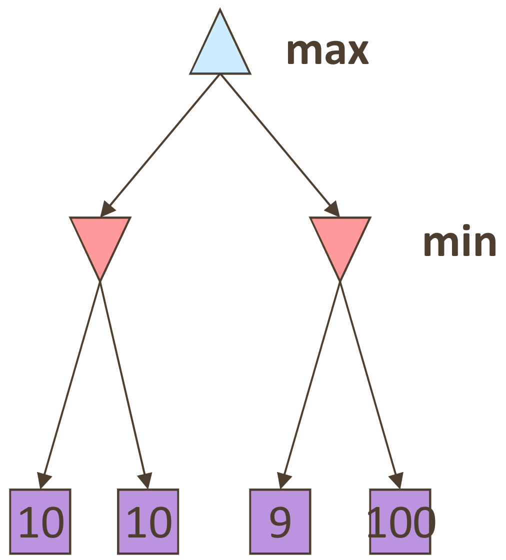
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

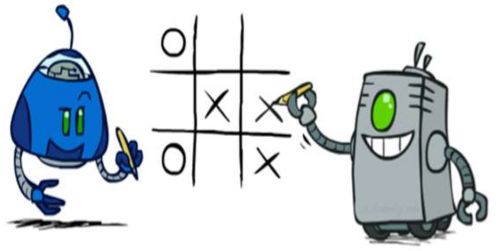
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Minimax Properties

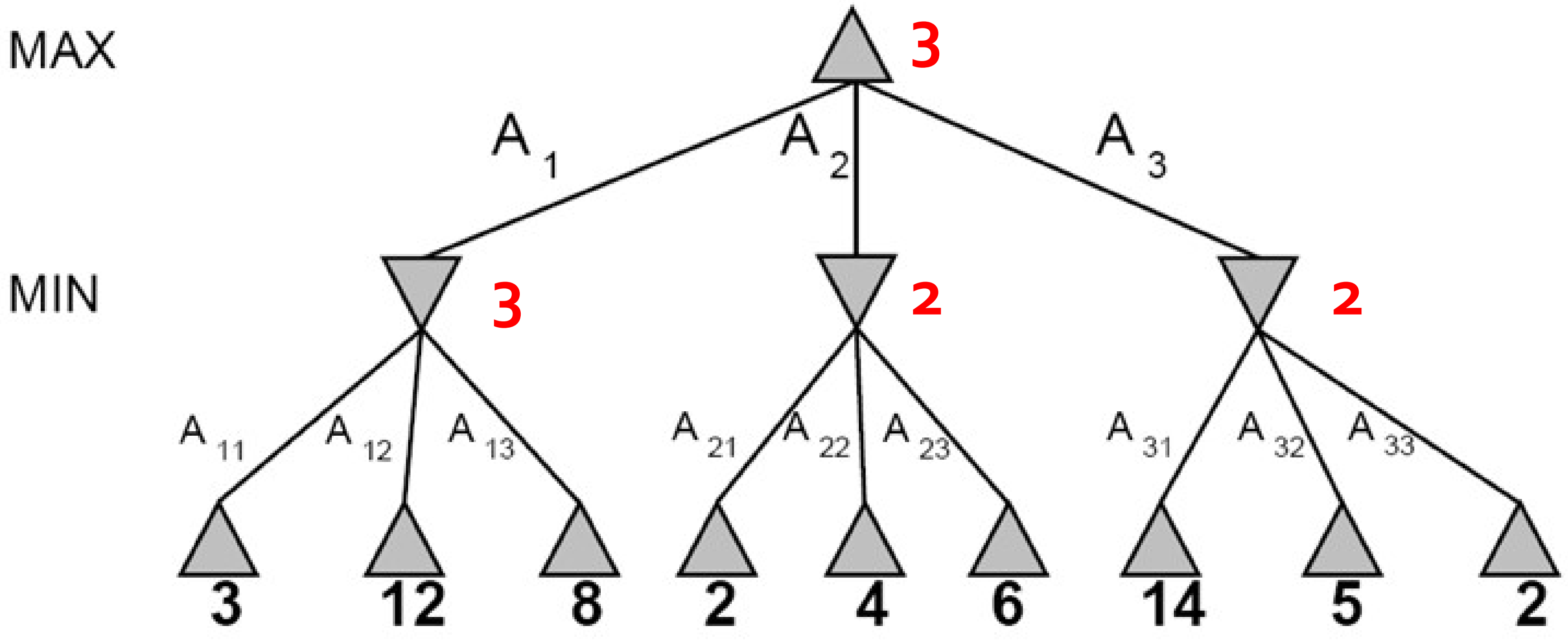






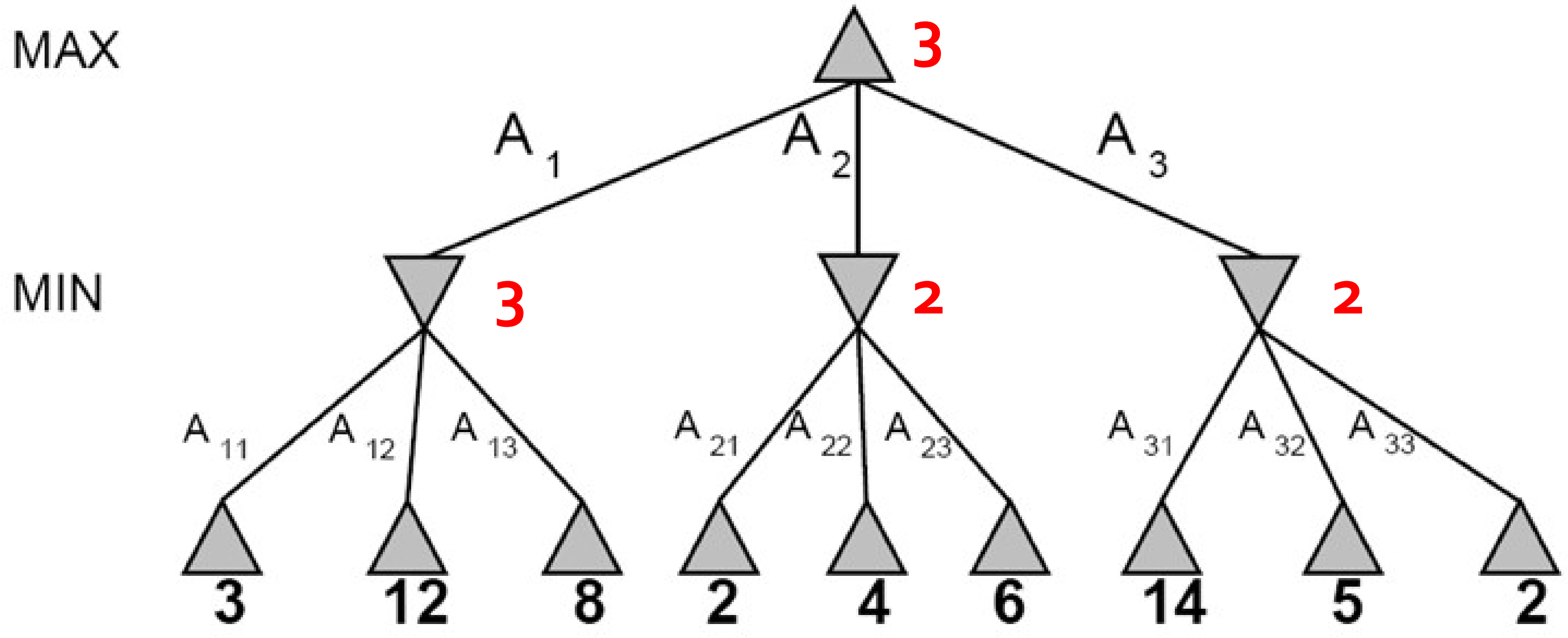
Optimal against a perfect player. Otherwise?

Game tree search



- **Minimax value of a node**: the utility (for MAX) of being in the corresponding state, assuming perfect play on both sides
- **Minimax strategy:** Choose the move that gives the best worst-case payoff

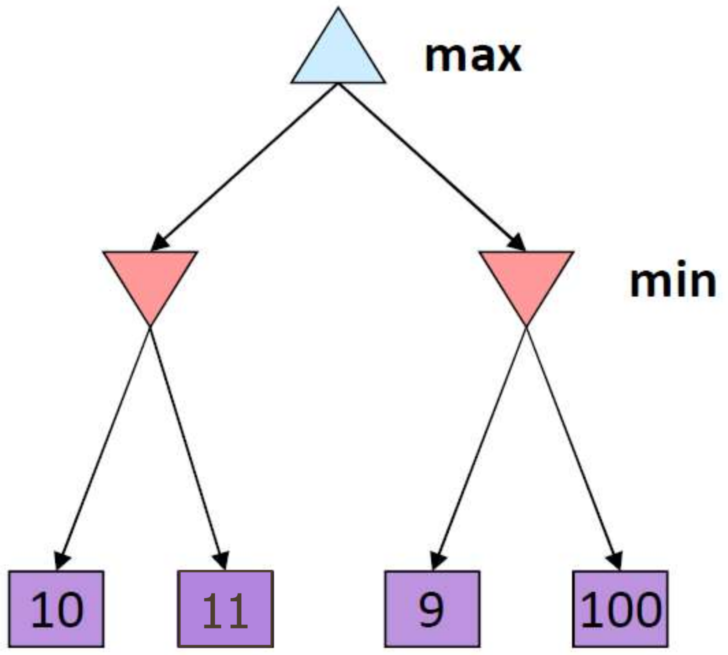
Computing the minimax value of a node



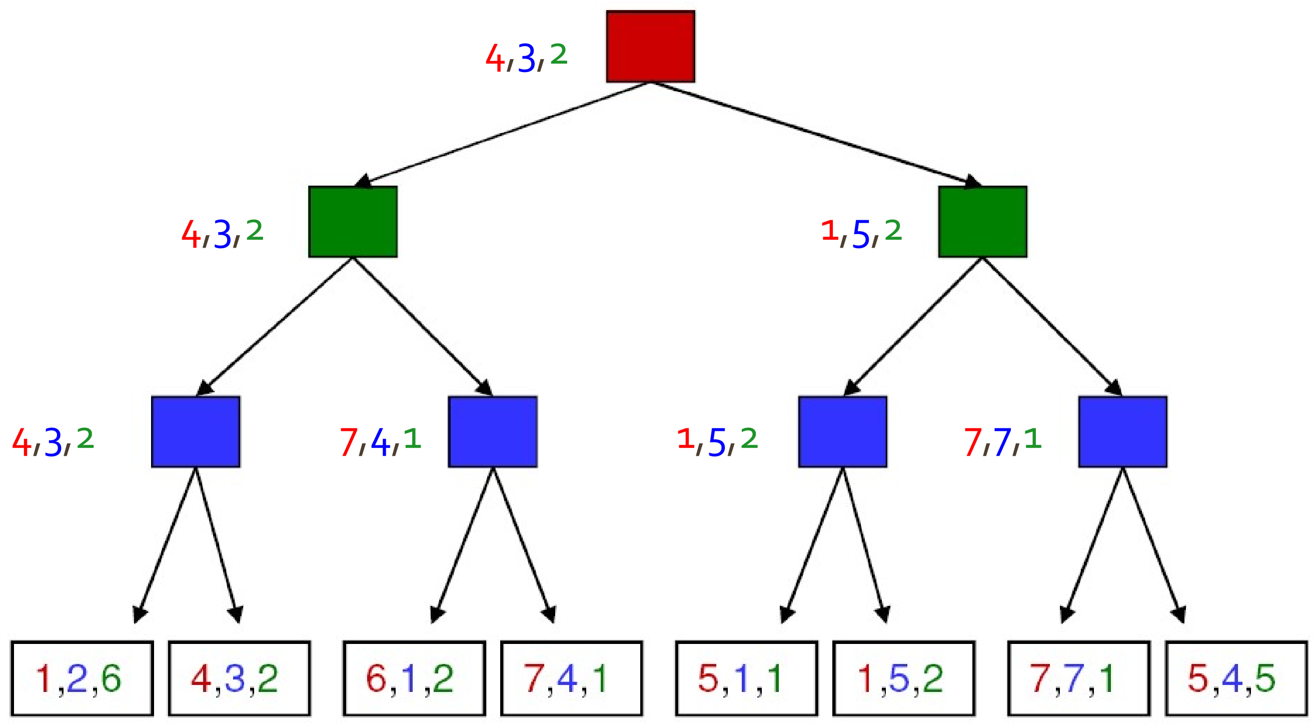
- **Minimax**(*node*) =
  - Utility(*node*) if *node* is terminal
  - $\max_{action}$ **Minimax**(Succ(*node*, *action*)) if *player* = MAX
  - $\min_{action}$ **Minimax**(Succ(*node*, *action*)) if *player* = MIN

Optimality of minimax

- The minimax strategy is optimal against an optimal opponent
- What if your opponent is suboptimal?
    - Your utility can only be higher than if you were playing an optimal opponent!
    - A different strategy may work better for a sub-optimal opponent, but it will necessarily be worse against an optimal opponent

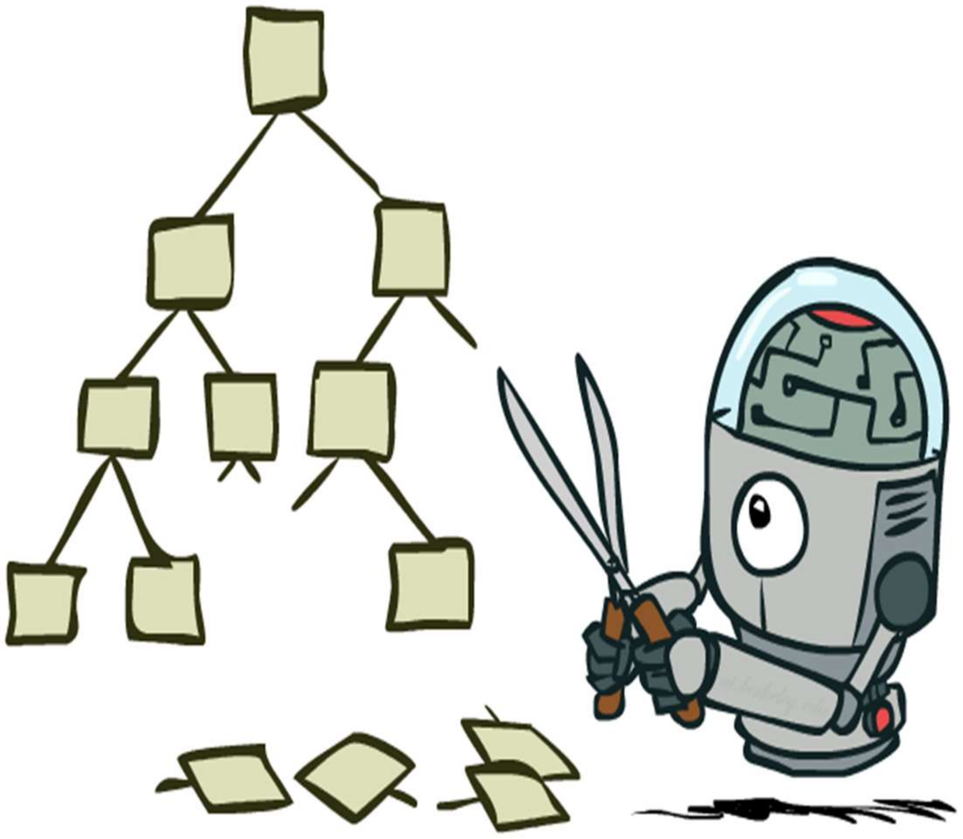


More general games



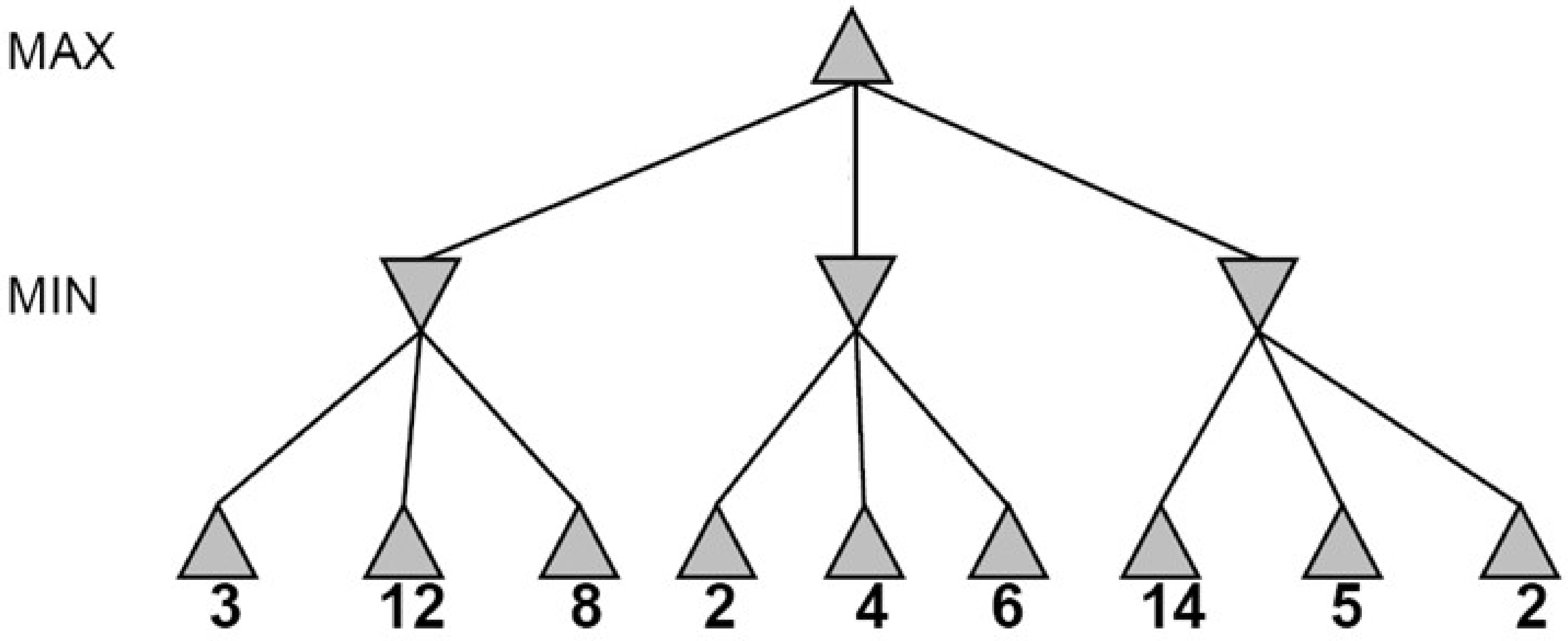
- More than two players, non-zero-sum
- Utilities are now tuples
- Each player maximizes their own utility at their node
- Utilities get propagated (*backed up*) from children to parents

Game Tree Pruning



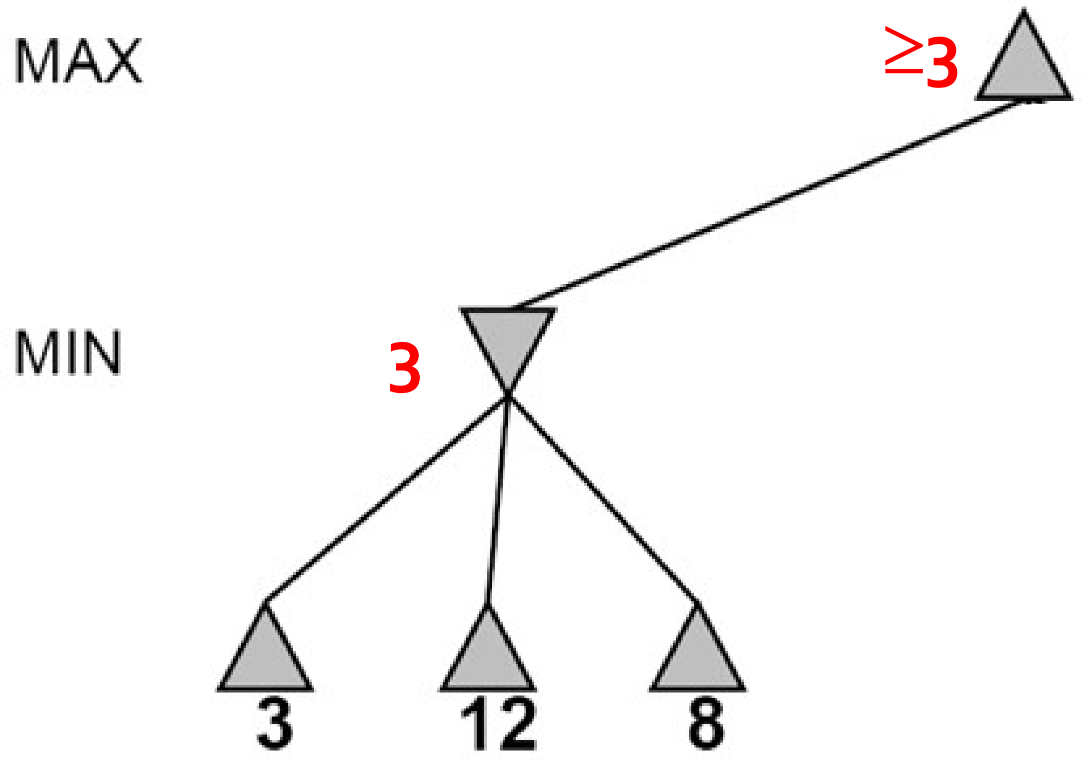
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



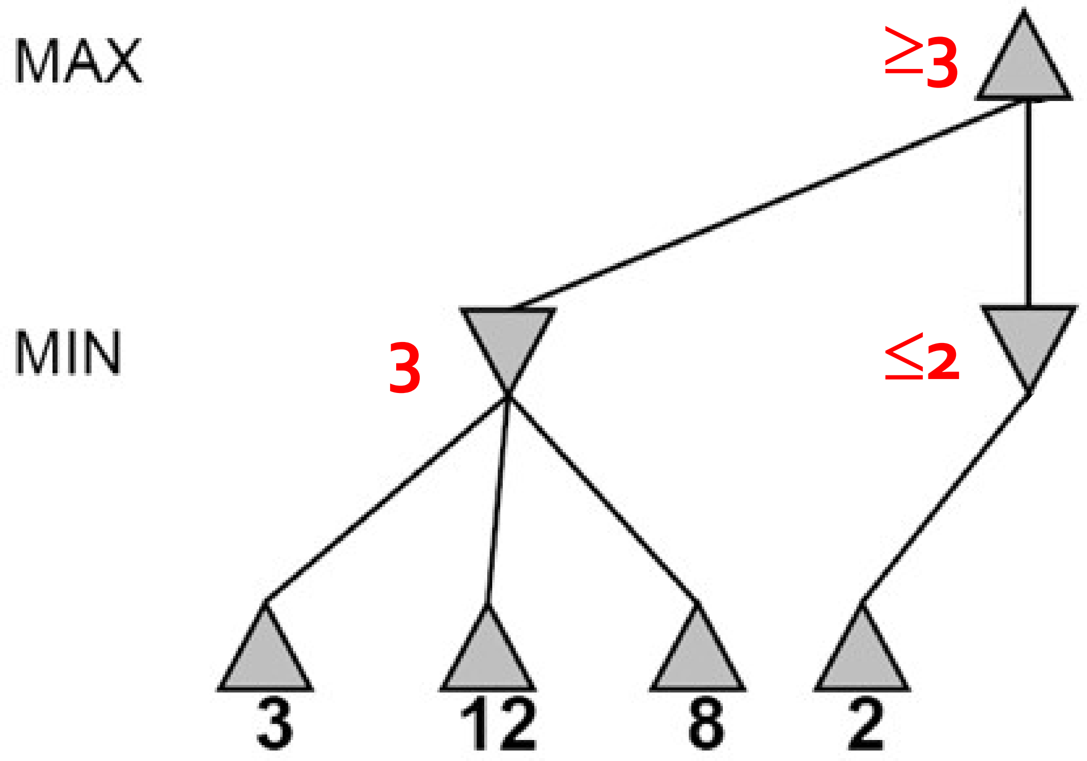
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



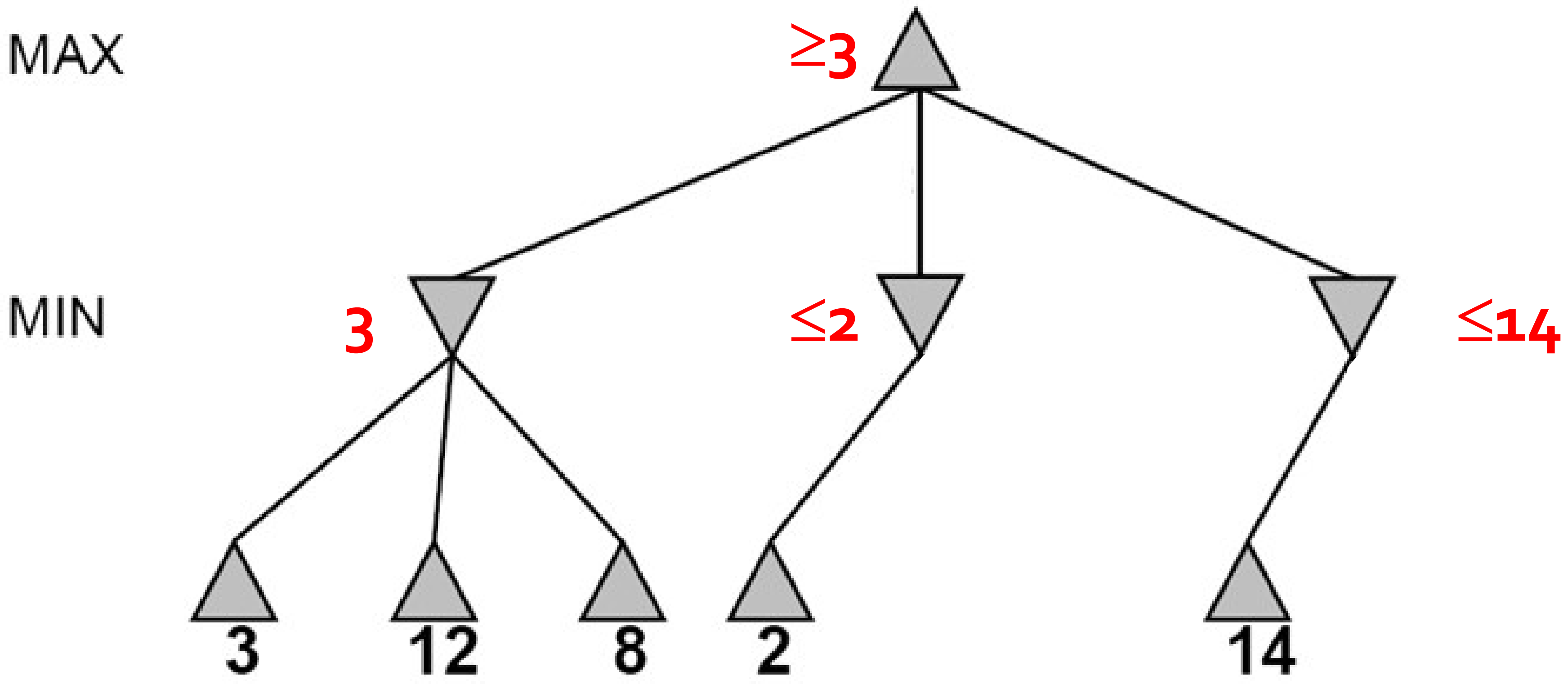
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



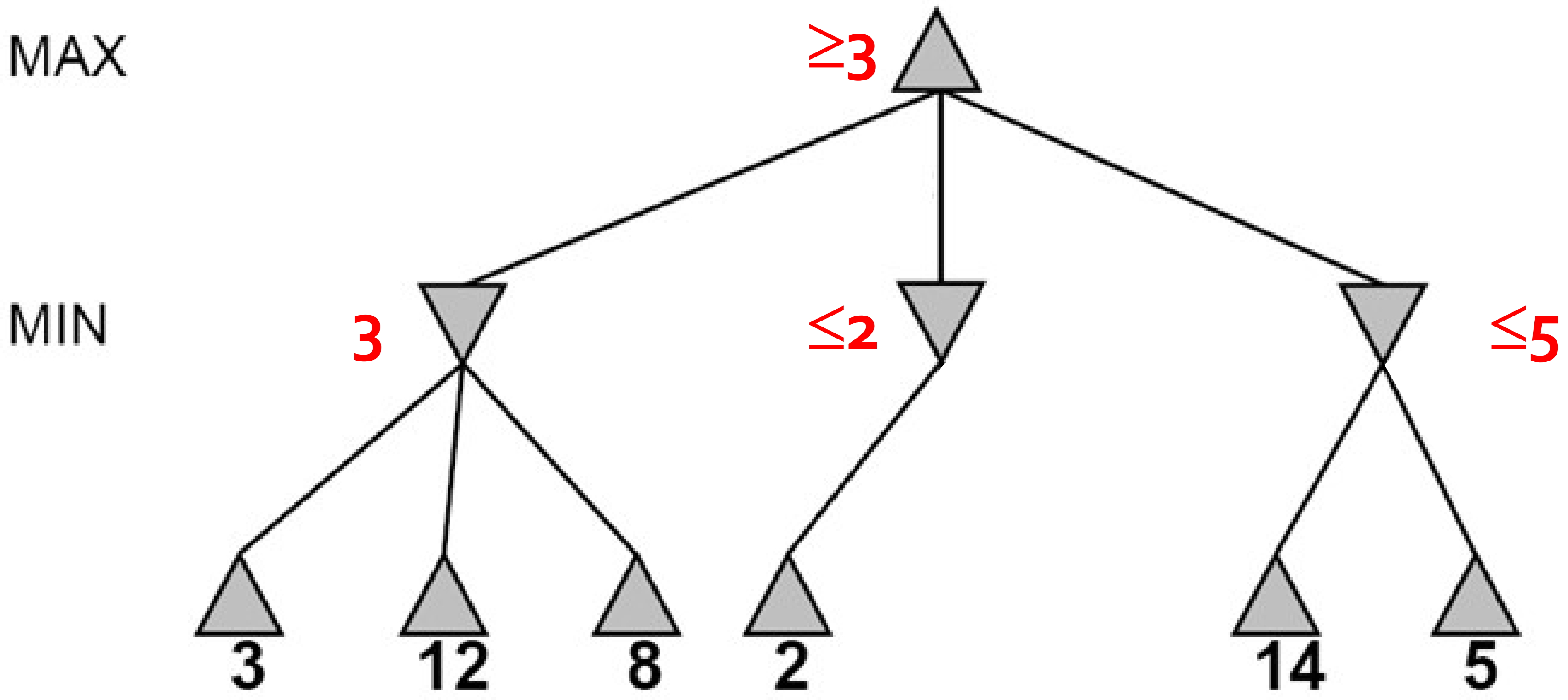
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



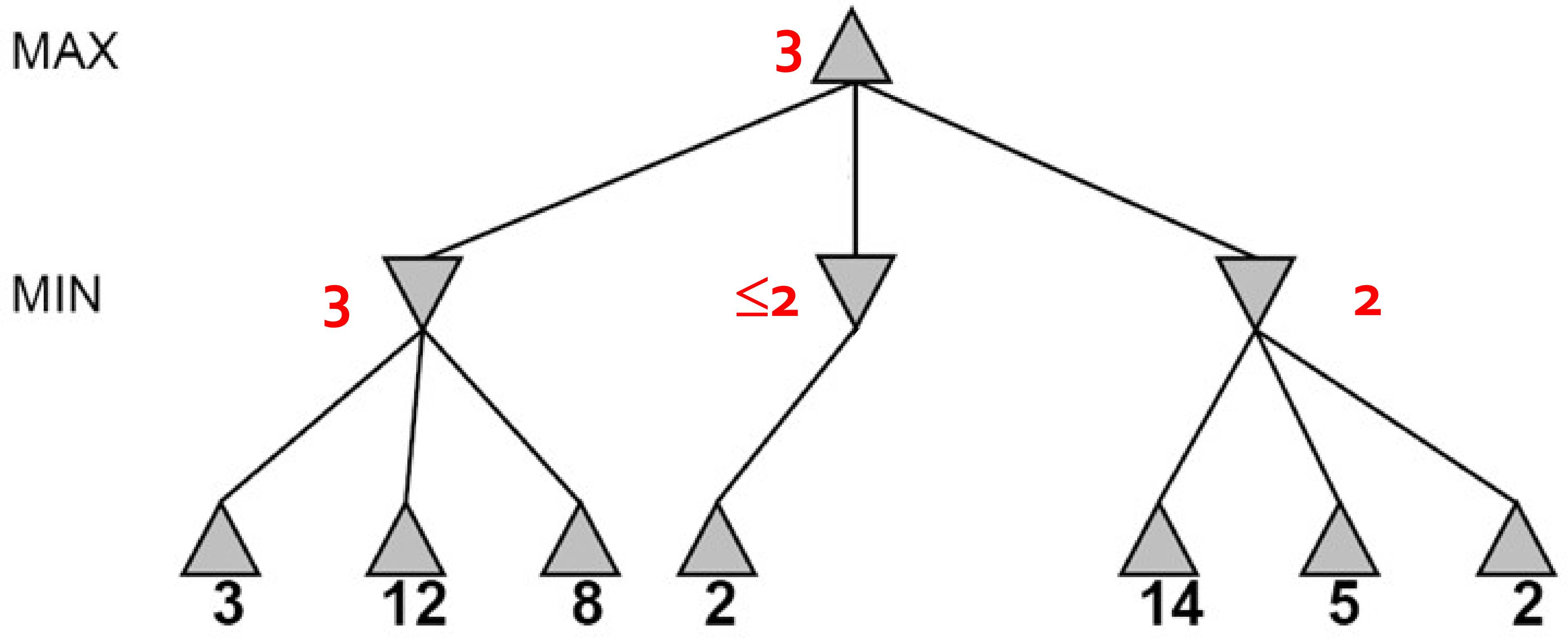
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning

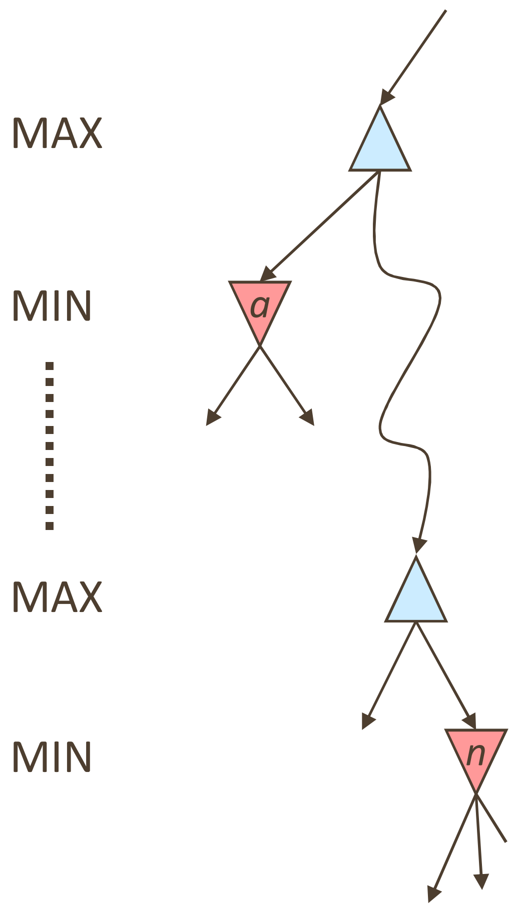
- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-Beta Pruning

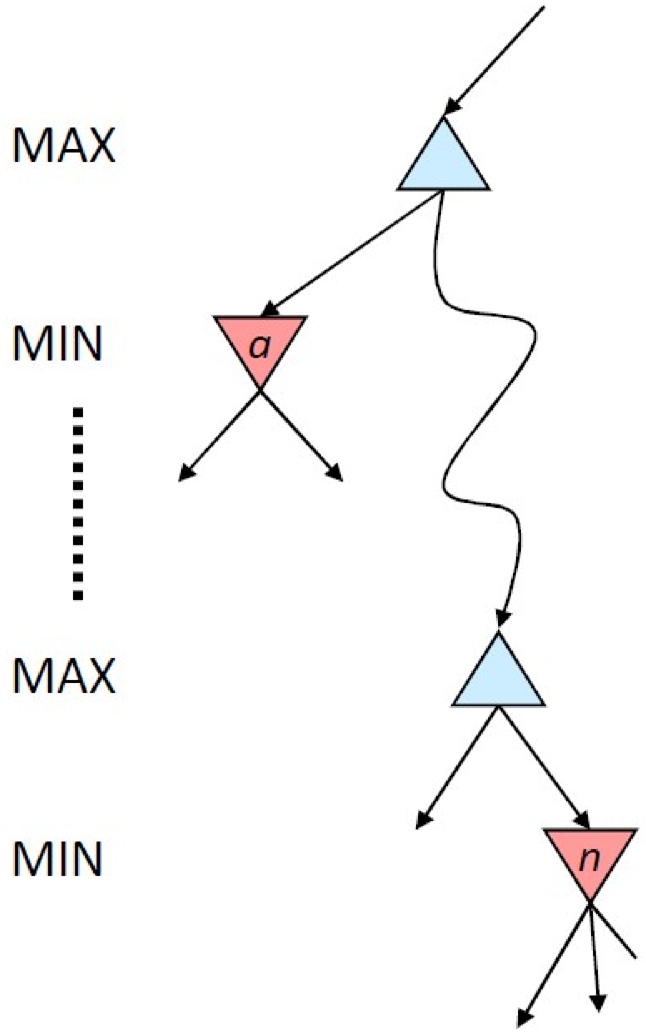
- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node *n*
  - We're looping over *n*'s children
  - *n*'s estimate of the childrens' min is dropping
  - Who cares about *n*'s value? MAX
  - Let *α* be the best value that MAX can get at any choice point along the current path from the root
  - If *n* becomes worse than *α*, MAX will avoid it, so we can stop considering *n*'s other children (it's already bad enough that it won't be played)

- MAX version is symmetric



Alpha-beta pruning

- **α** is the value of the best choice for the MAX player found so far at any choice point above node *n*
- We want to compute the MIN-value at *n*
- As we loop over *n*'s children, the MIN-value decreases
- If it drops below **α**, MAX will never choose *n*, so we can ignore *n*'s remaining children
- Analogously, **β** is the value of the lowest-utility choice found so far for the MIN player



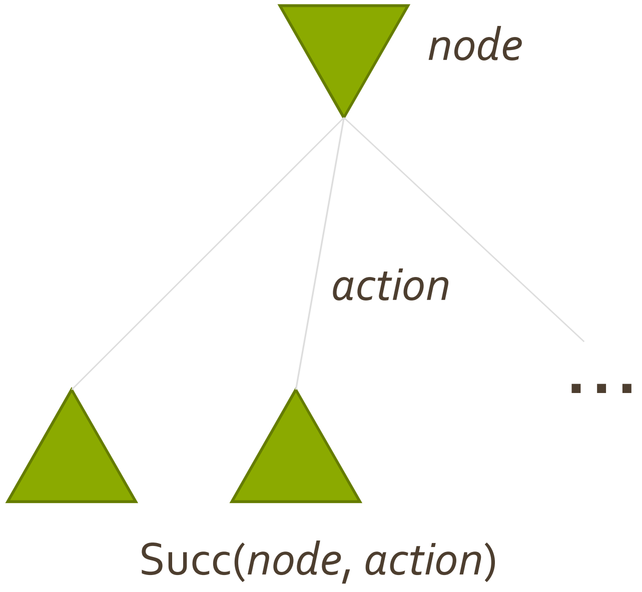
Alpha-beta pruning

```
Function action = Alpha-Beta-Search(node)
    v = Min-Value(node, −∞, ∞)
    return the action from node with value v
```

α: best alternative available to the Max player

β: best alternative available to the Min player

```
Function v = Min-Value(node, α, β)
    if Terminal(node) return Utility(node)
    v = +∞
    for each action from node
        v = Min(v, Max-Value(Succ(node, action), α, β))
        if v ≤ α return v
        β = Min(β, v)
    end for
    return v
```



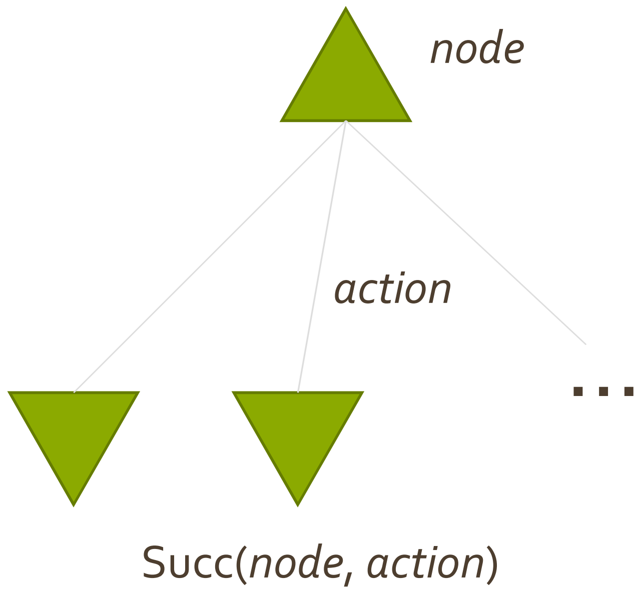
Alpha-beta pruning

```
Function action = Alpha-Beta-Search(node)
    v = Max-Value(node, −∞, ∞)
    return the action from node with value v
```

α: best alternative available to the Max player

β: best alternative available to the Min player

```
Function v = Max-Value(node, α, β)
    if Terminal(node) return Utility(node)
    v = −∞
    for each action from node
        v = Max(v, Min-Value(Succ(node, action), α, β))
        if v ≥ β return v
        α = Max(α, v)
    end for
    return v
```



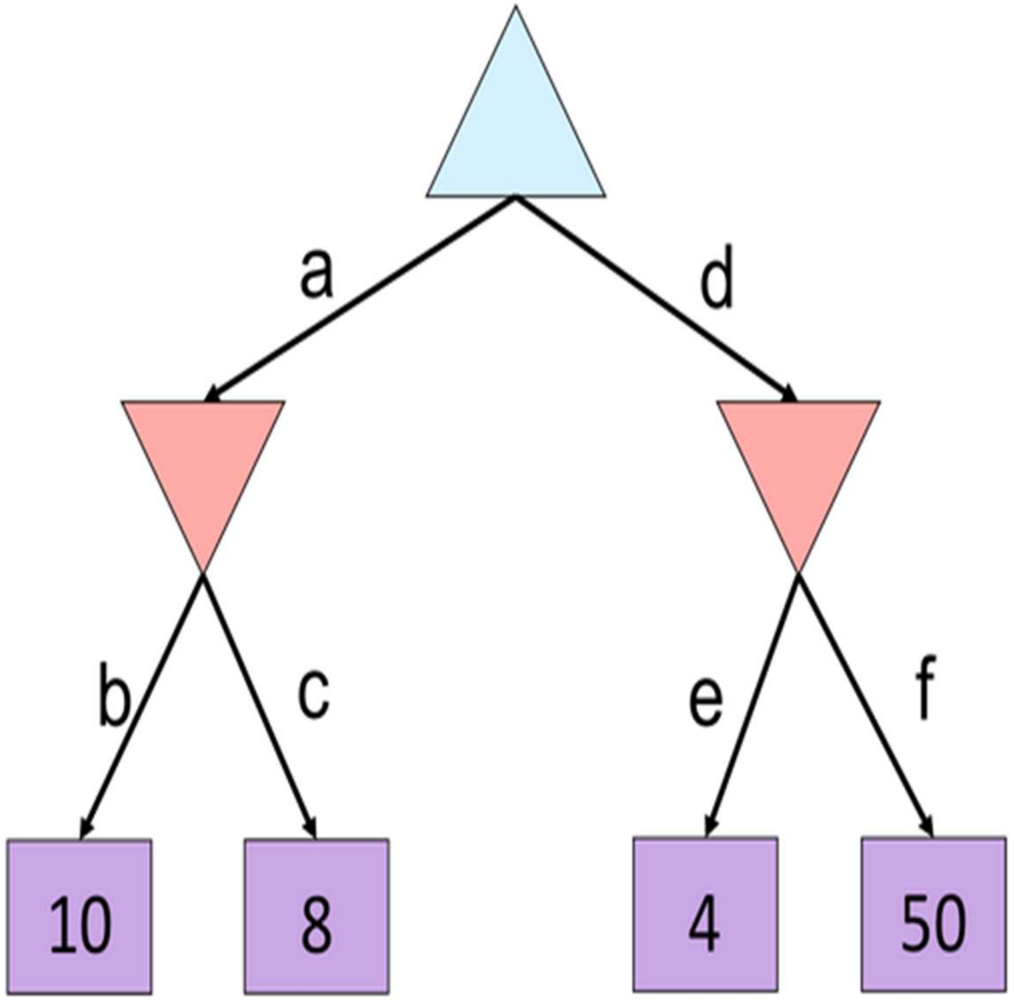
Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

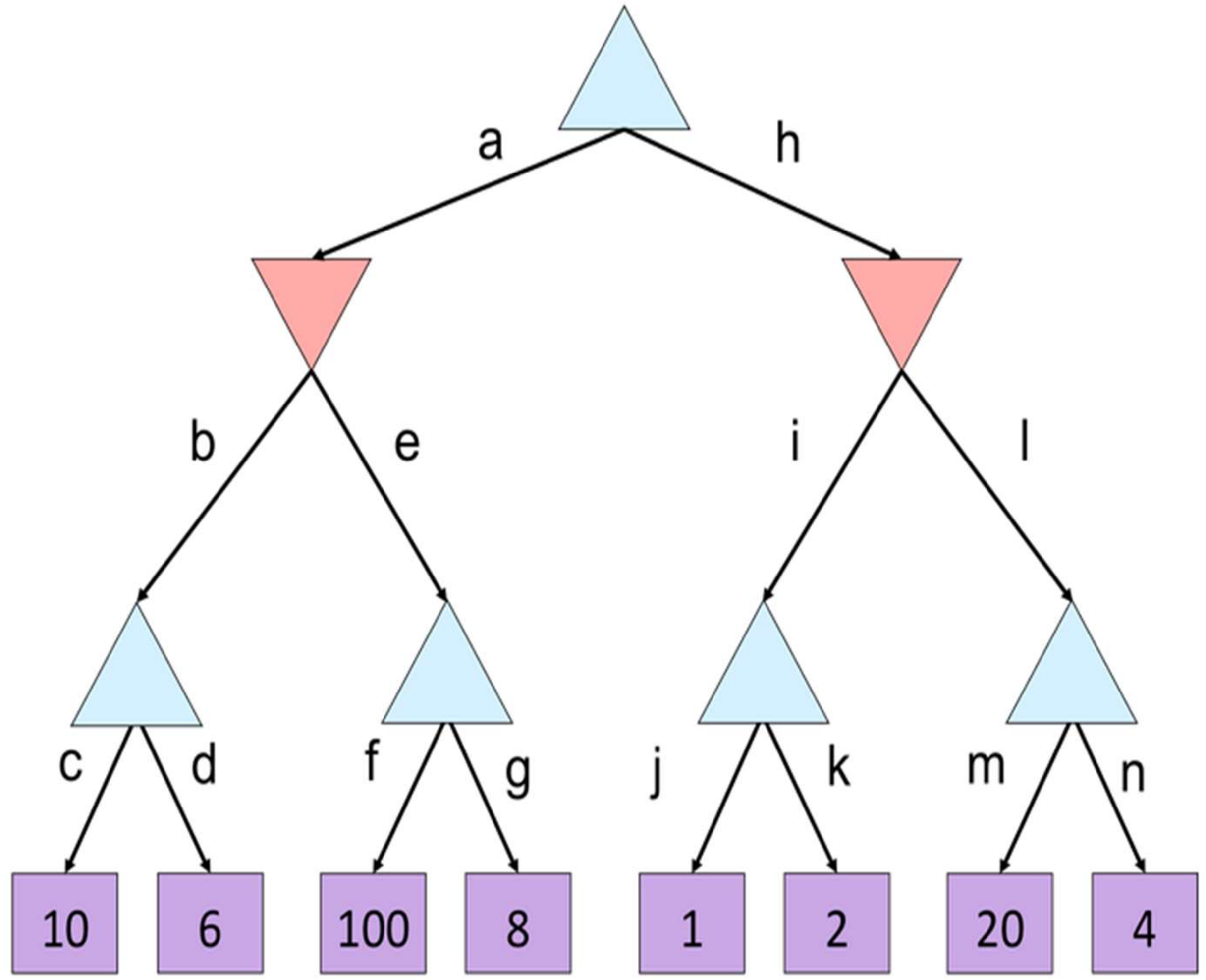
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Quiz



Alpha-Beta Quiz 2



Alpha-Beta Pruning Properties

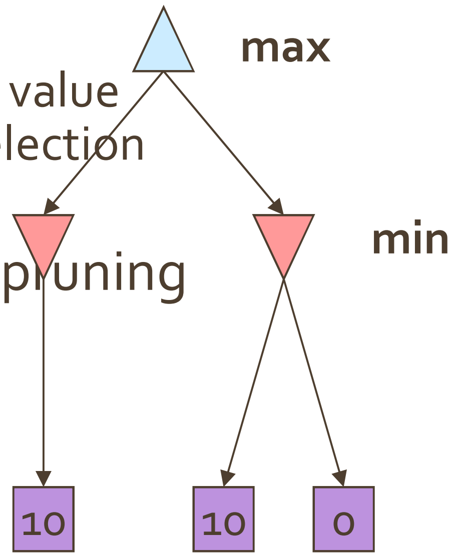
- This pruning has no effect on minimax value computed for the root!

- Values of intermediate nodes might be wrong
    - Important: children of the root may have the wrong value
    - So the most naïve version won't let you do action selection

- Good child ordering improves effectiveness of pruning

- With "perfect ordering":
    - Time complexity drops to $O(b^{m/2})$
    - Doubles solvable depth!
    - Full search of, e.g. chess, is still hopeless…

- This is a simple example of metareasoning (computing about what to compute)



Alpha-beta pruning

- Pruning does not affect final result
- Amount of pruning depends on move ordering
  - Should start with the "best" moves (highest-value for MAX or lowest-value for MIN)
  - For chess, can try captures first, then threats, then forward moves, then backward moves
  - Can also try to remember "killer moves" from other branches of the tree
- With perfect ordering, the time to find the best move is reduced to $O(b^{m/2})$ from $O(b^m)$
  - Depth of search is effectively doubled

Evaluation function

- Cut off search at a certain depth and compute the value of an **evaluation function** for a state instead of its minimax value
  - The evaluation function may be thought of as the probability of winning from a given state or the *expected value* of that state
- A common evaluation function is a weighted sum of *features*:

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

  - For chess, w_k may be the **material value** of a piece (pawn = 1, knight = 3, rook = 5, queen = 9) and $f_k(s)$ may be the advantage in terms of that piece
- Evaluation functions may be *learned* from game databases or by having the program play many games against itself