Game Playing

Why do AI researchers study game playing?

1. It’s a good reasoning problem, formal and nontrivial.

2. Direct comparison with humans and other computer programs is easy.

What Kinds of Games?

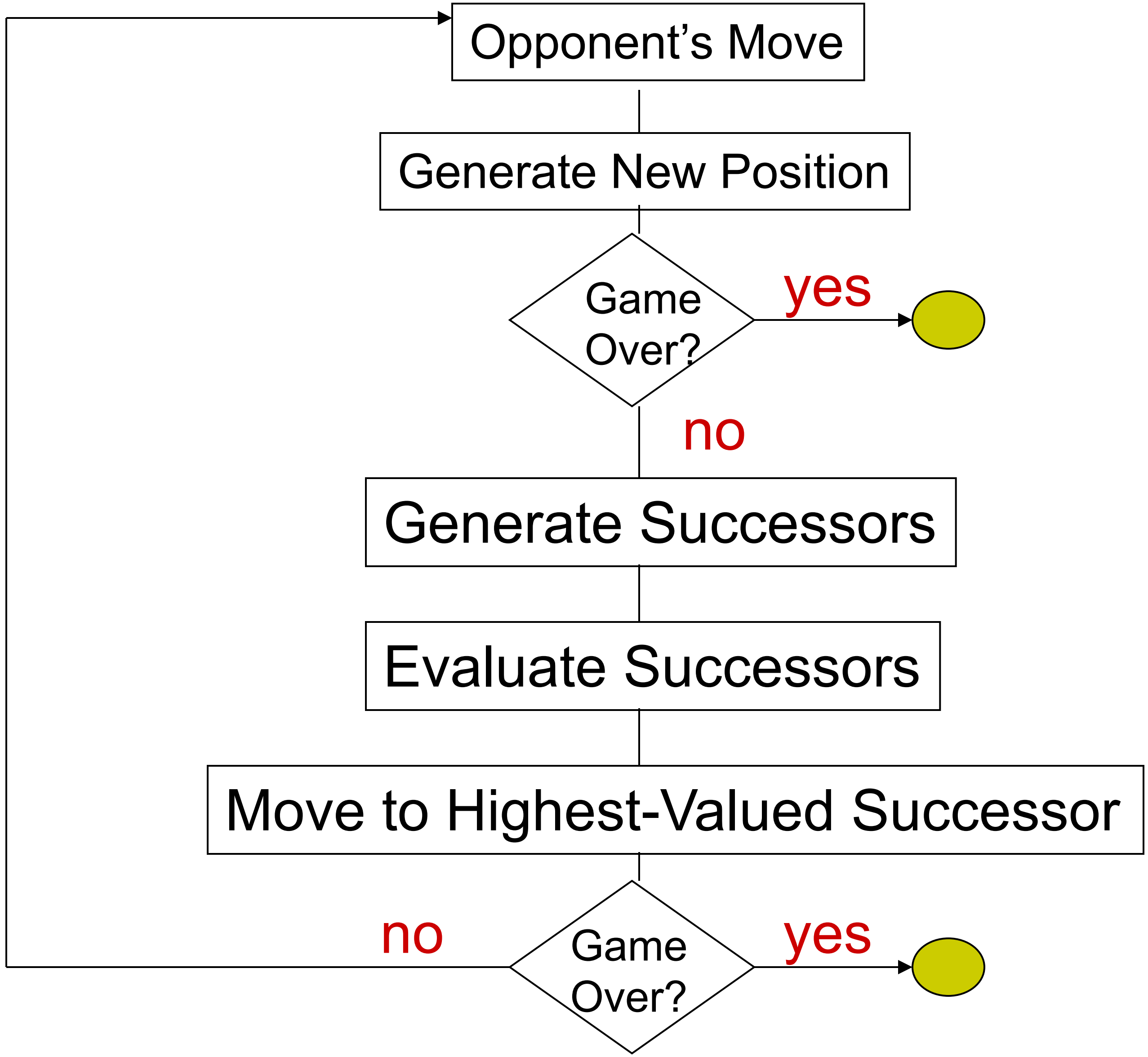
Mainly games of strategy with the following characteristics:

1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a reward for each move
4. Objective is to maximize your reward

Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Two-Player Game



Solving Problems involving Game

- Think about **what's happening during a game playing** like chess.
  - How do we compute heuristic and take advantage of it?
  - Once you made a move, what happens next?
  - Solving problems involving game need different strategies (like Game Theory).
- **What is Game Theory?**
  - "The study of mathematical models of conflict and cooperation between intelligent **rational decision-makers**."
  - **Originally**, started with **zero-sum games** involving two persons (von Neumann).
  - Today, game theory applies to a **wide range** of behavioral relations, and is now an umbrella term for the science of logical decision making in humans, animals, and computers.

Game Types

- Zero-sum game/Non-zero-sum game
  - A **zero-sum game** is a mathematical representation of a situation in which each participant's gain or loss of utility is exactly ***balanced*** by the losses or gains of the utility of the other participants.
  - **Non-zero-sum** describes a situation in which the interacting parties' aggregate gains and losses can be less than or more than zero. Example: prisoner's dilemma
- **Many other types** such as Cooperative/Non-cooperative, Symmetric/Asymmetric, Simultaneous/Sequential, etc.

Zero-Sum Game in a Chess

- **Things to consider when playing a chess**
  - Know the rules first and come up with winning strategies.
  - The game involves at least two players and alternate turns.
    - Try to play a chess game with your friend.
  - How can we use a game strategy under the environment of taking turns?
    - Need take into account for the actions of the opponent.
- **General winning strategy**
  - Maximize my advantage and Minimize opponent's advantage whenever possible (zero-sum game).
    - **Maximizing/Minimizing advantage doesn't necessarily mean we want MAX/MIN** score all the time.

Mini-Max Algorithm (Adversarial Search)

- ***Assumption***: Your opponent uses the ***same knowledge*** of the state space as you use and applies that knowledge in a consistent effort to win the game.
- **Algorithm sketch** (based on **BFS with bound**) to make a decision
  1. Create a game graph by the rules of the game and strategies.
  2. Label each level of the game graph, alternating MIN and MAX.
     - **Decide** either **MAX** or **MIN** at the root node based on your **heuristic** that **measures** your **advantage**.
     - **Note:** You always want to maximize your advantage.
  3. For each leaf node, apply a heuristic function.
  4. Propagates heuristic values upward the graph through successive parent nodes according to the following rules:
     - If the parent state is a **MAX** node, give it the maximum value from its children.
     - If the parent state is a **MIN** node, give it the minimum value from its children.
  5. Choose the path that returns the value to the root as your next move.

Mini-Max Algorithm

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

  $v \leftarrow$ 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*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

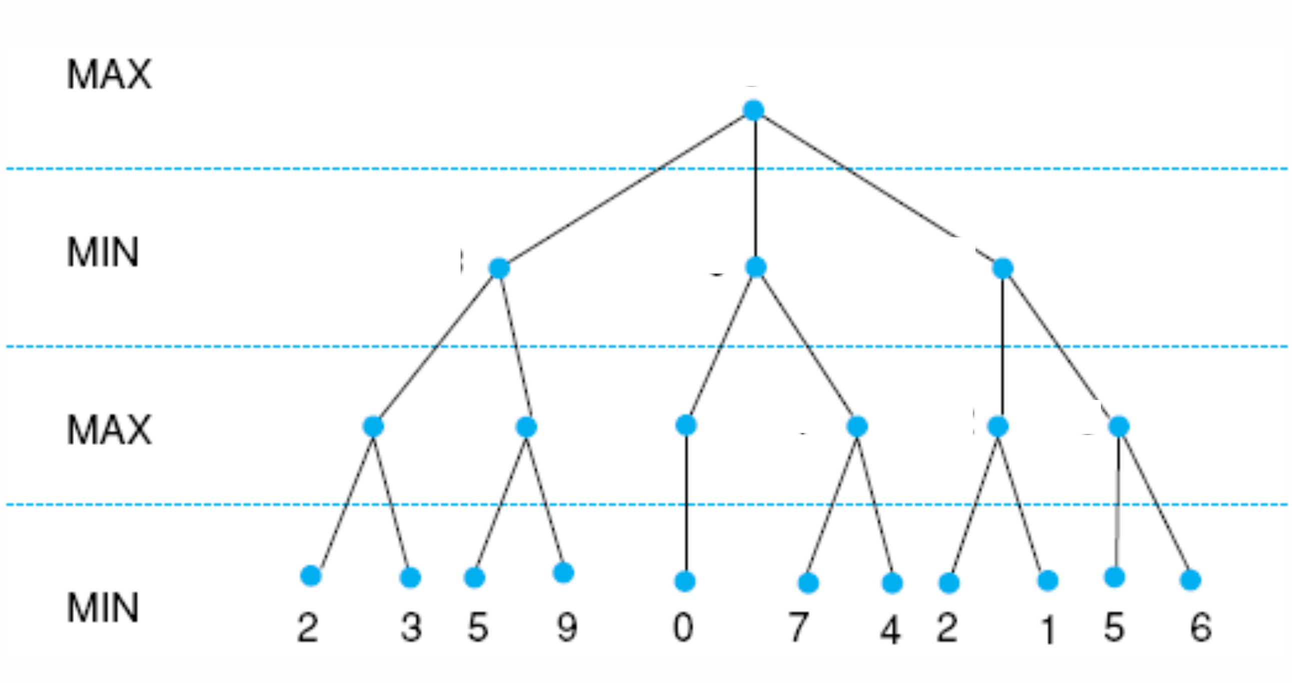
---

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**
    $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return v

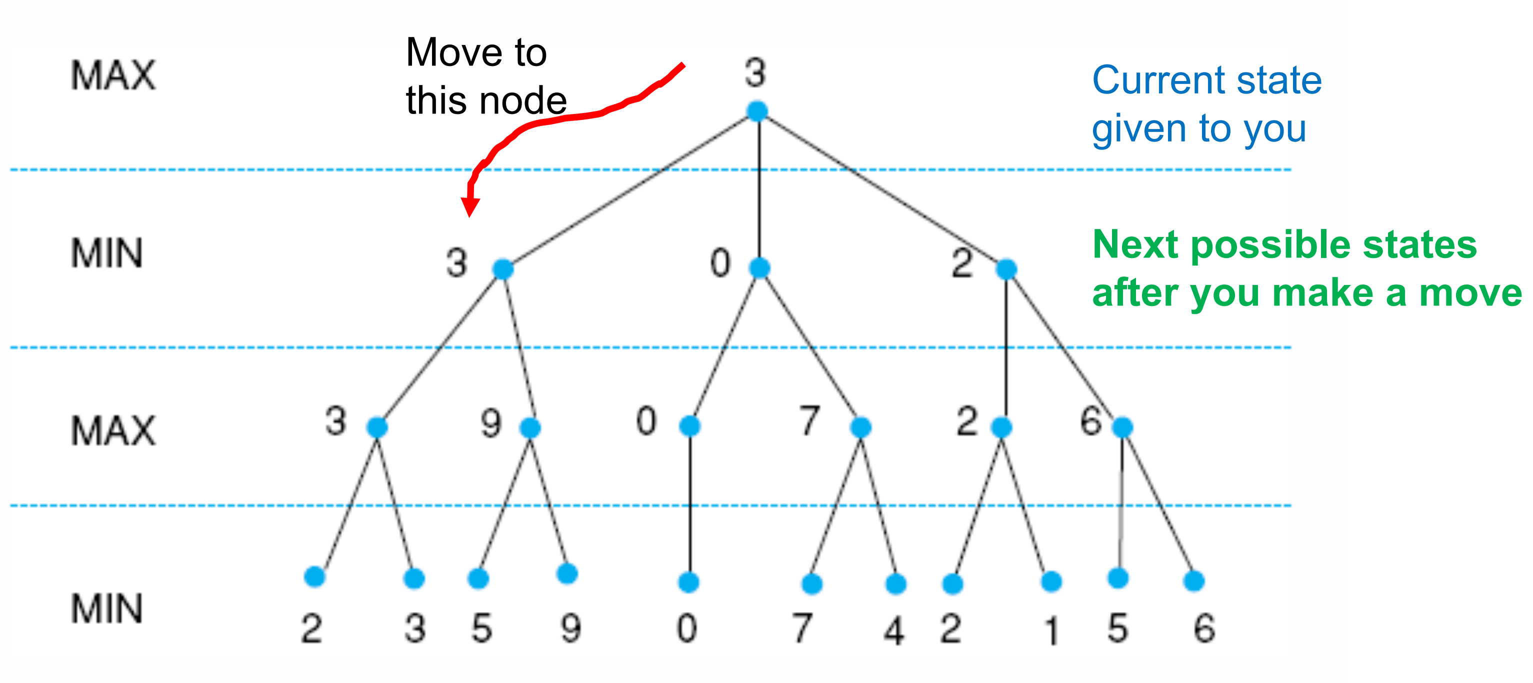
A Stage after Heuristics Applied to a Hypothetical Game Tree by <u>Fixed 3 Ply Mini-Max</u>

3-ply look ahead



Leaf nodes show heuristic values.

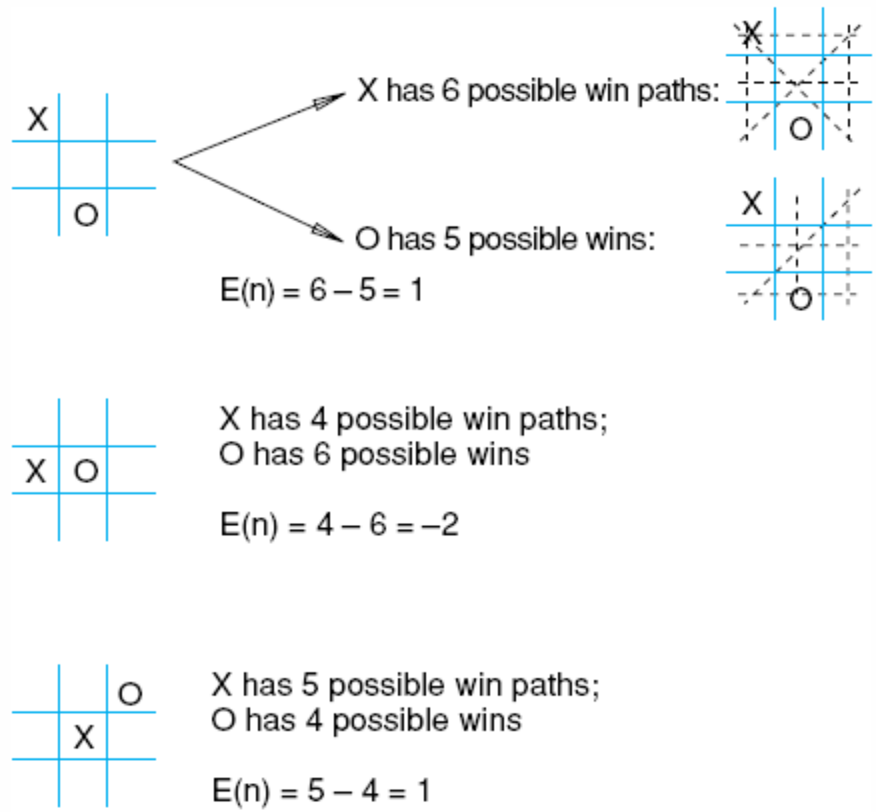
The Stage after Heuristic Values Propagated to a Hypothetical Game Tree



Leaf states show heuristic values; **Internal states** show backed-up values.

Heuristics Applied to States of Tic-Tac-Toe for Mini-Max Algorithm

X: my move
O: opponent's move



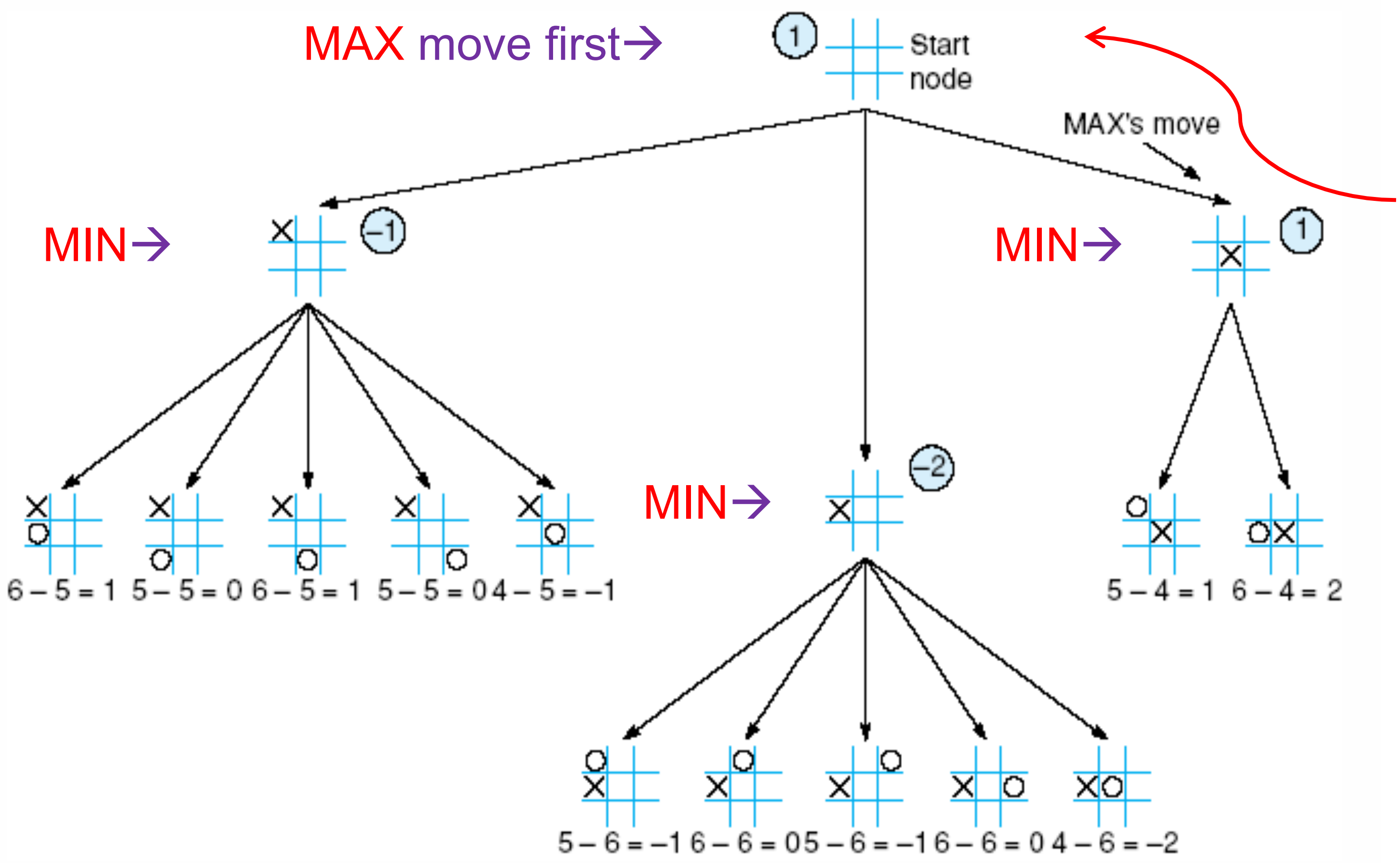
Heuristic is E(n) = M(n) – O(n)

where M(n) is the total of My possible winning lines

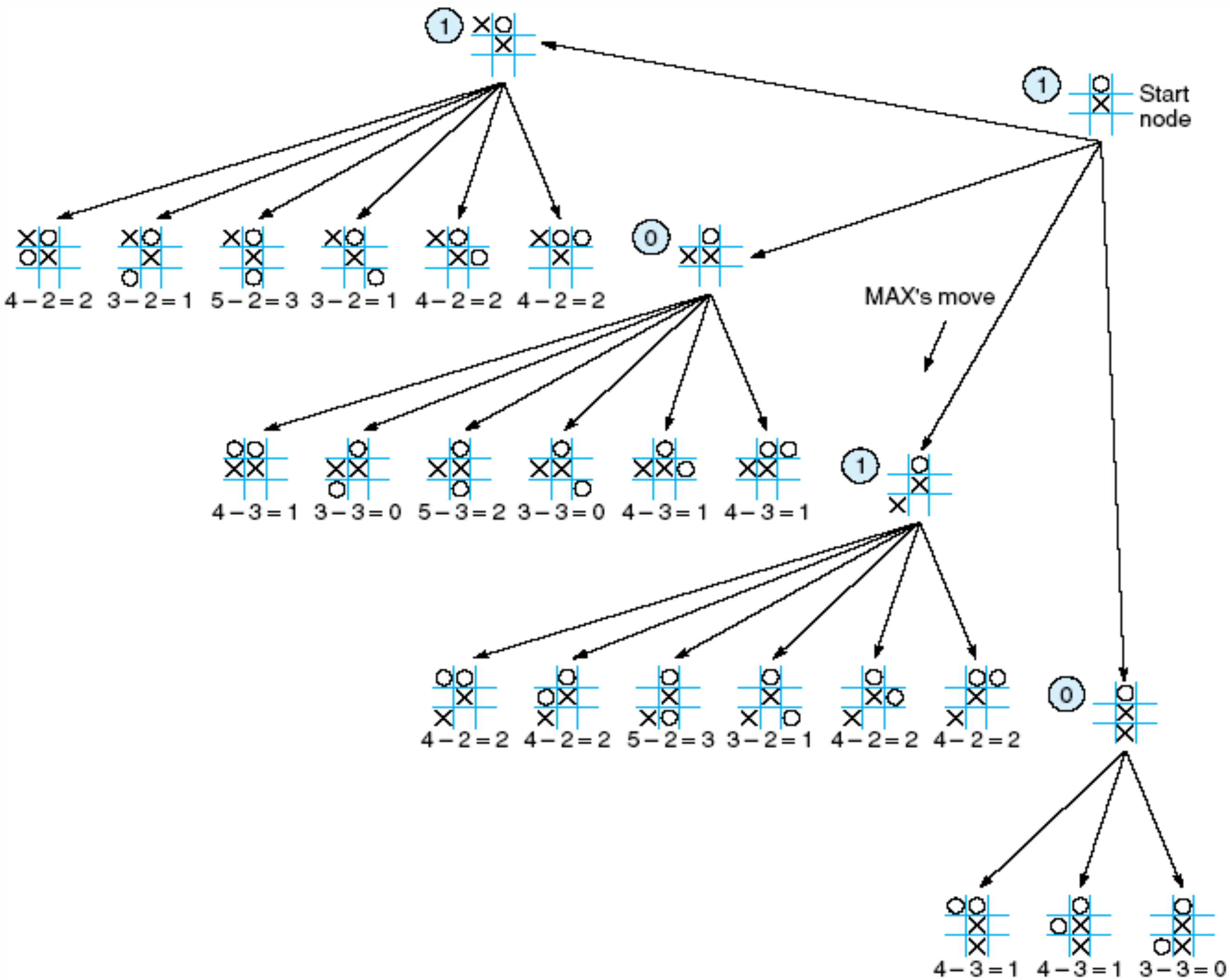
O(n) is total of Opponent's possible winning lines

E(n) is the total Evaluation for state n

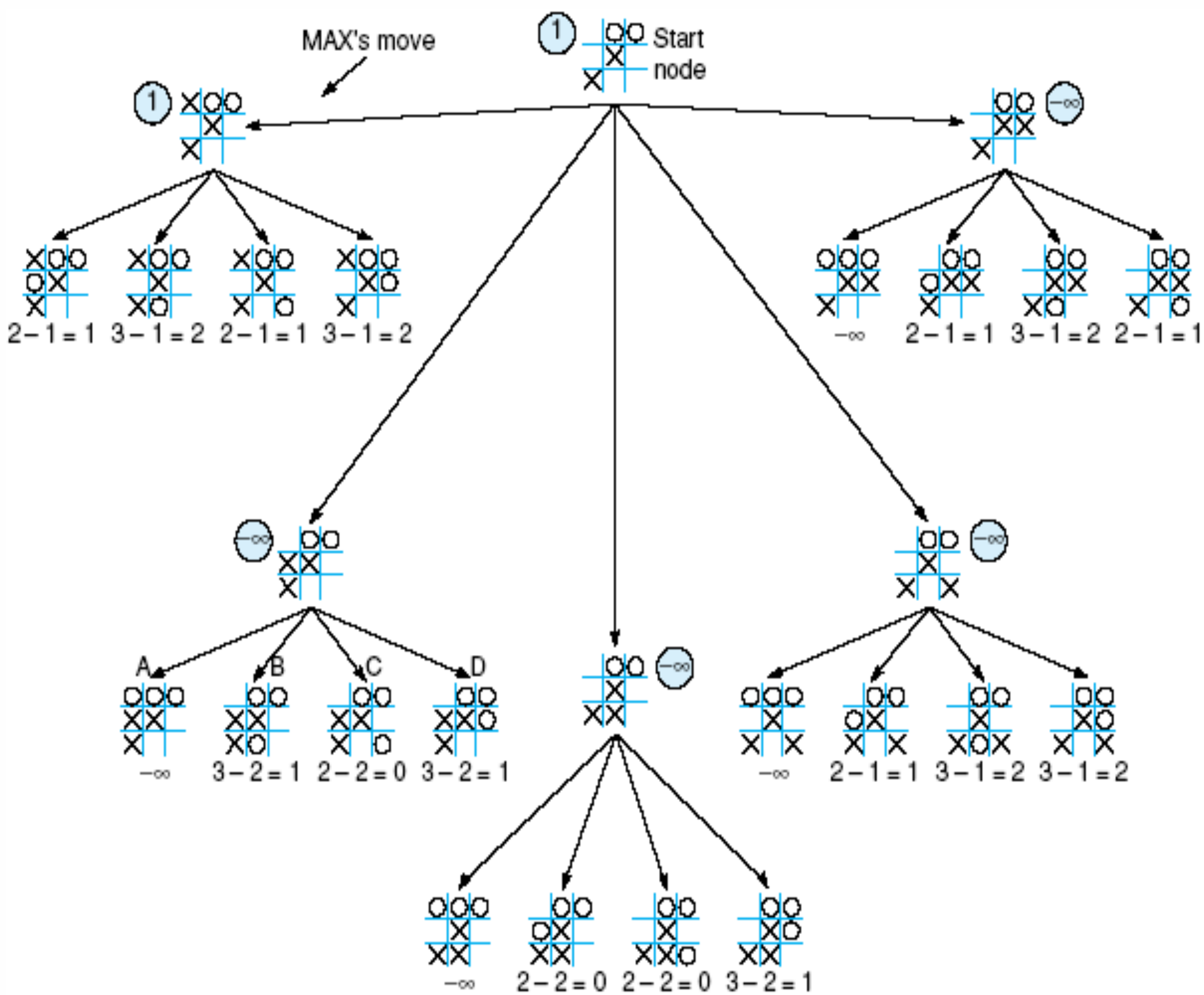
Two Ply Mini-max Applied to the Opening Move of Tic-Tac-Toe (from Nilsson, 1971)



Two Ply Mini-max, and One of Two Possible MAX's Second Moves



Two-ply Mini-max Applied to MAX's Move Near the End of the Game



+Can we use the Best-First Search when playing a game?

+If the opponent makes a mistake, will the mini-max still work?

+Can a player who uses mini-max strategy be guaranteed to win a game against a player who doesn't?

Questions

- **How to decide MIN or MAX at the root node?** (basis of my advantage)
- **Why do we alternate MIN-MAX?**
- **When you begin with MAX, do MIN nodes try to choose the worst move?**
- **Why do we apply heuristic function to ONLY leaf nodes?**
- **If leaf nodes correspond to opponent's turn, do we have to choose always MIN?**
- **Do we reuse this same game tree to decide next move when you have your turn after the opponent's move?**

Alpha-beta Pruning for Mini-max

- **Problem of mini-max**
  - Pursues all branches in the space, including many that could be ignored or pruned by a more intelligent algorithm.
- **Main idea of alpha-beta pruning**
  - Rather than searching the entire space to the ply depth, it proceeds in a *depth-first* fashion. Two values, **alpha** for **MAX** and **beta** for **MIN** are determined during each search using more informed heuristics for **efficiency**.
    - Alpha can never decrease and Beta can never increase.

Algorithm sketch

- Descend to full ply depth in a depth-first fashion and apply the heuristic f(n) to a state and all its siblings.
- Values are backed up to parents using mini-max algorithm.
- **Use two rules below to terminate search** based on **alpha** and **beta** values:

+**Stop** the search below any MIN node if the alpha value of its ancestors (MAX node) ≥ **the** beta value of the MIN node.

+**Stop** the search below any MAX node if the beta value of any of its ancestors (MIN node) ≤ the alpha value of the MAX node.

The α-β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

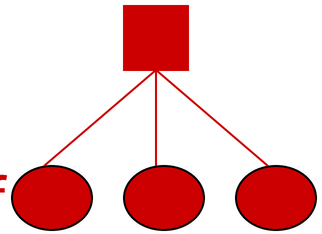
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, 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**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

ALPHA cutoff



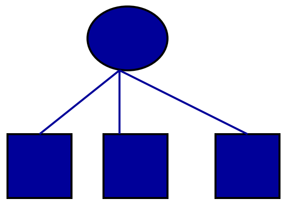
Note that: here, β is the successors' β. ν is current's state's temporary α

The α-β algorithm – cont.

```
function MIN-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for  MAX along the path to state
           β, the value of the best alternative for  MIN along the path to state

   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```

BETA cutoff



Note that: here, α is the successors' α. ν is current's state's temporary β

Alpha-Beta Procedure

- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

$v > \alpha$

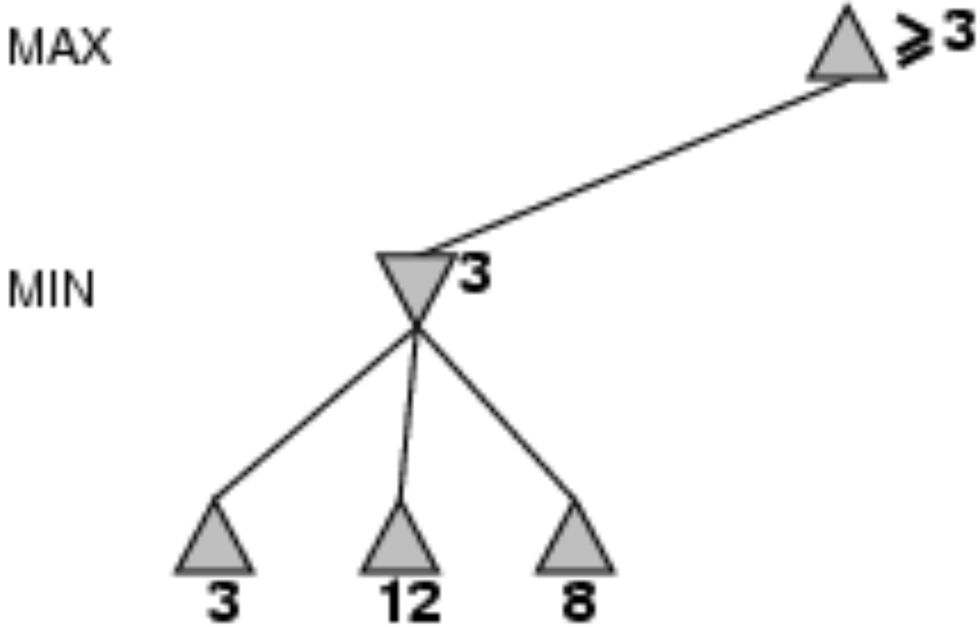
Note that: here, α is current state' α.
We seek for a v which is larger than α

- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

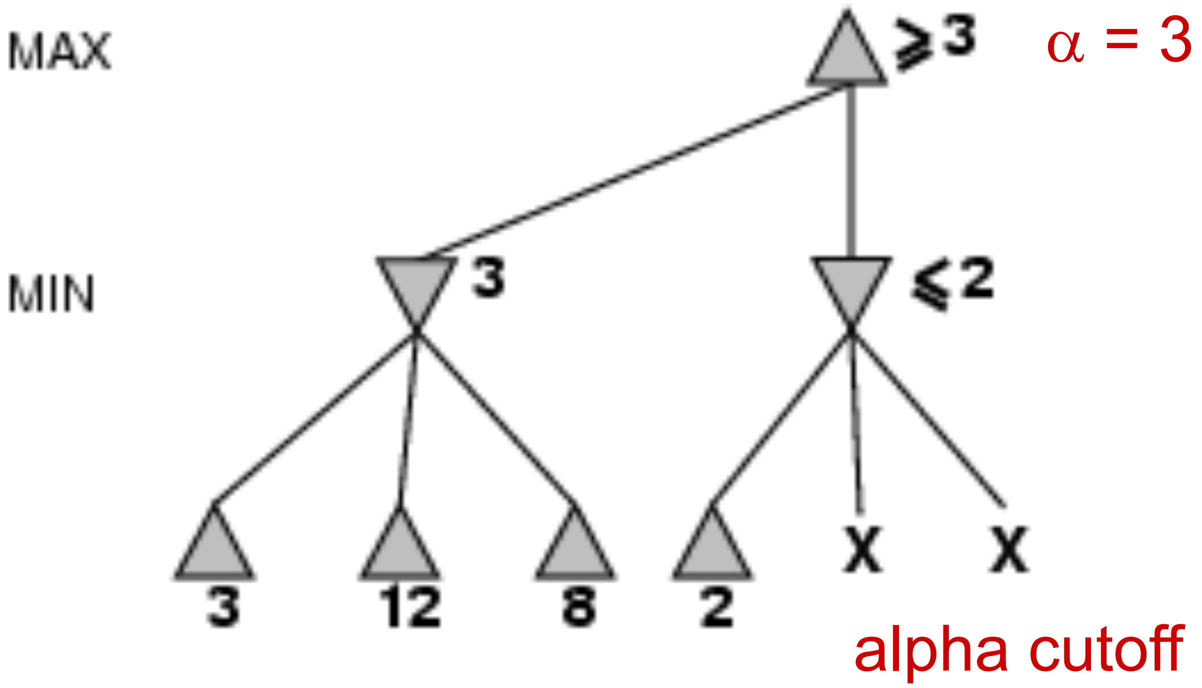
$v < \beta$

Note that: here, β is current state' β.
We seek for a v which is smaller than β

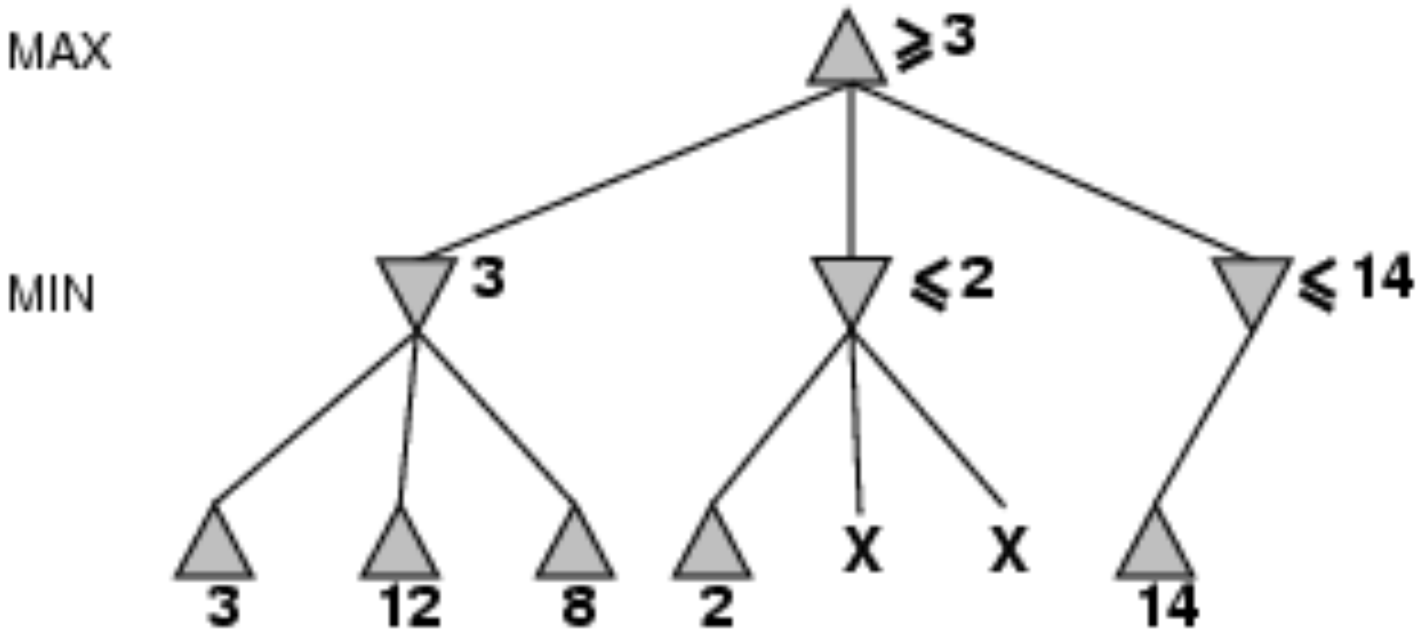
α-β pruning example



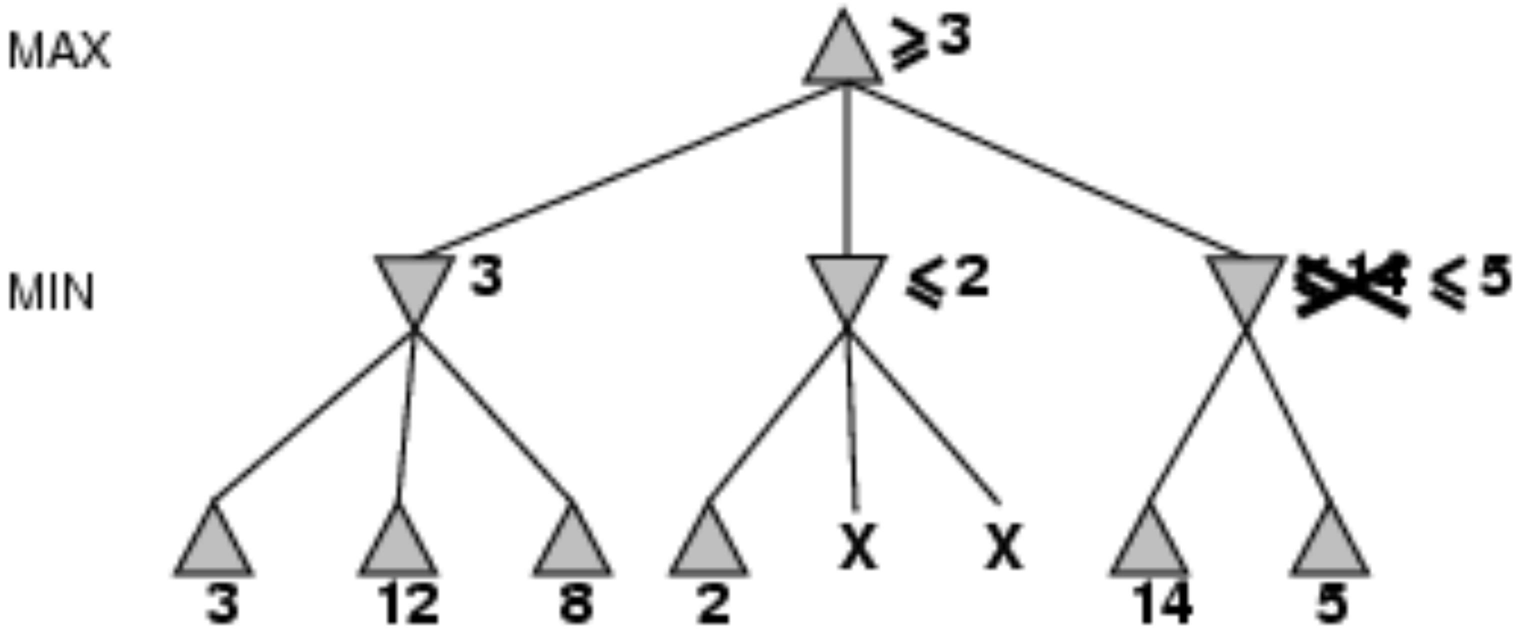
α-β pruning example



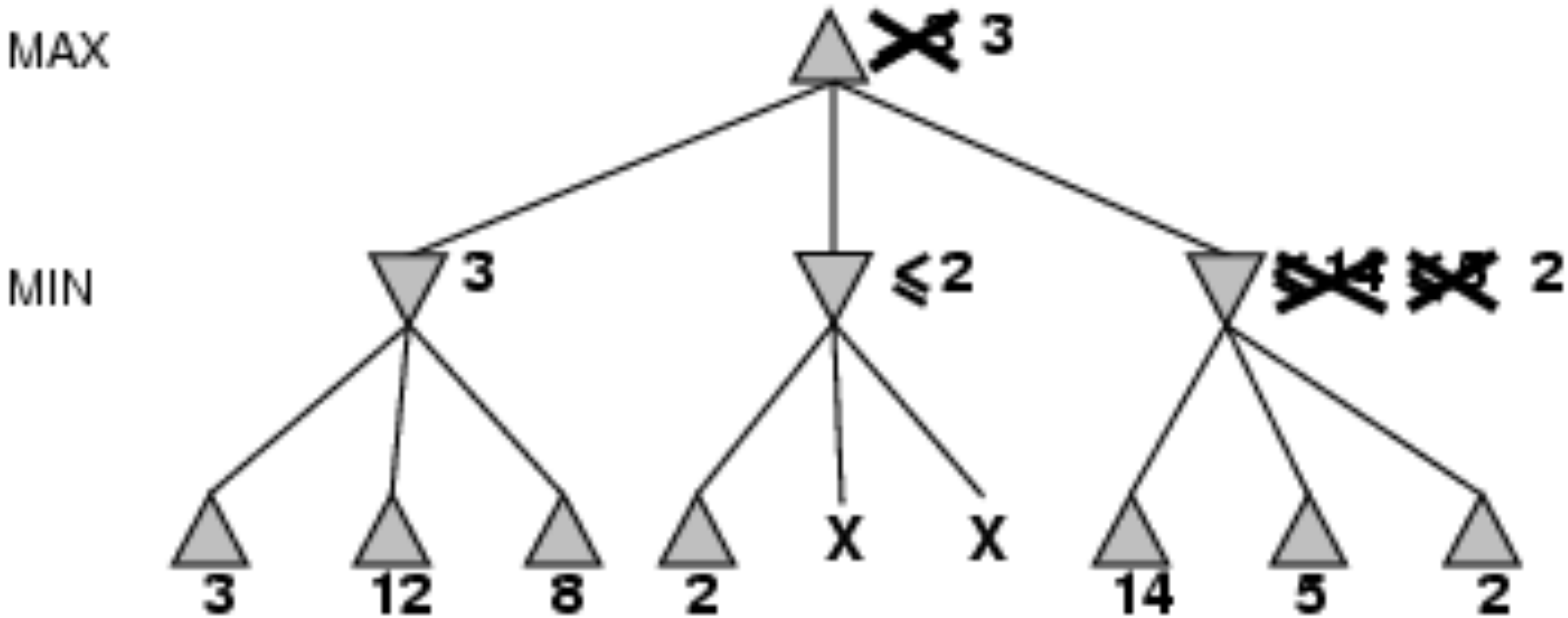
α-β pruning example



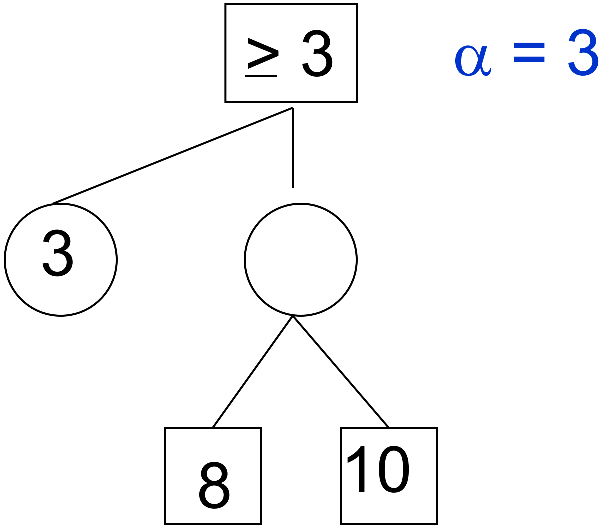
α-β pruning example



α-β pruning example

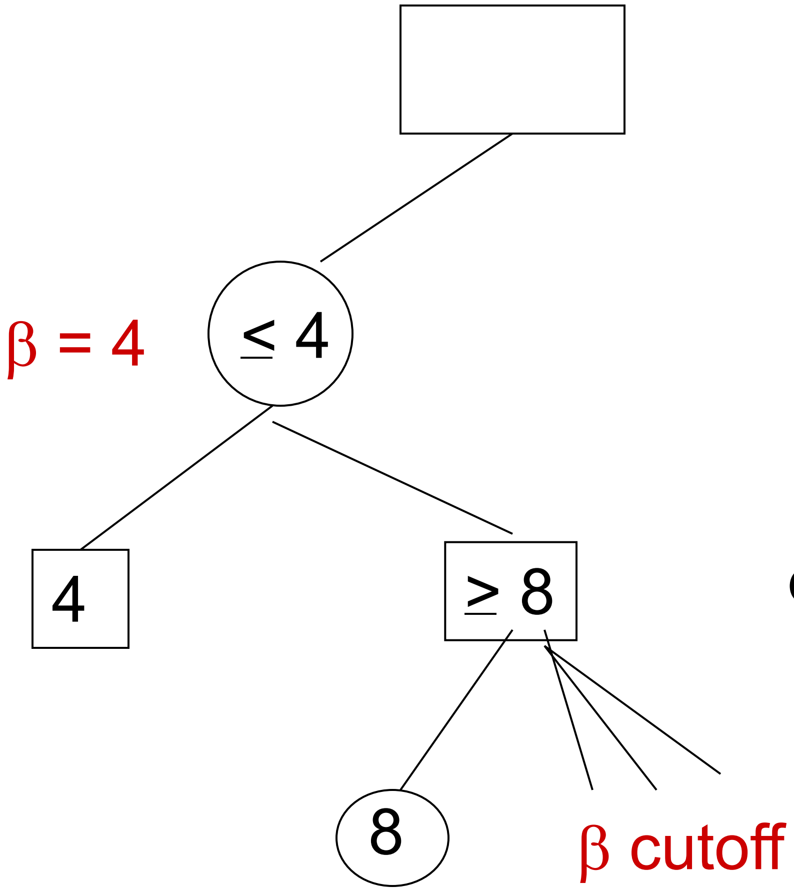


Alpha Cutoff



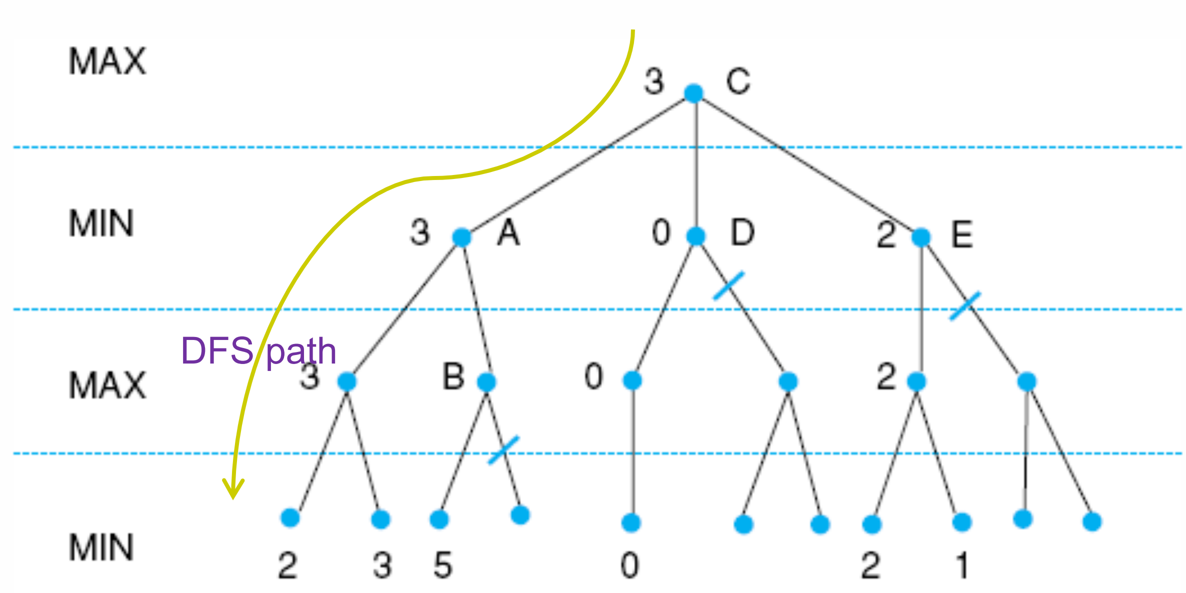
What happens here? Is there an alpha cutoff?

Beta Cutoff



Q: why is it not Alpha cutoff?

Alpha-beta Pruning Applied to a Hypothetical State Space Graph



Need to know **h()** values for both **current** and **parent** nodes.

Alpha-beta pruning NEVER create a complete game tree!

So Alpha-beta pruning NEVER prune any branch, instead NOT expand unnecessary branches. The quality of decision making will be the same if # of ply remains the same.

A has $\beta = 3$ (A will be no larger than 3)

B is β pruned, since $5 > 3$

C has $\alpha = 3$ (C will be no smaller than 3)

D is α pruned, since $0 < 3$

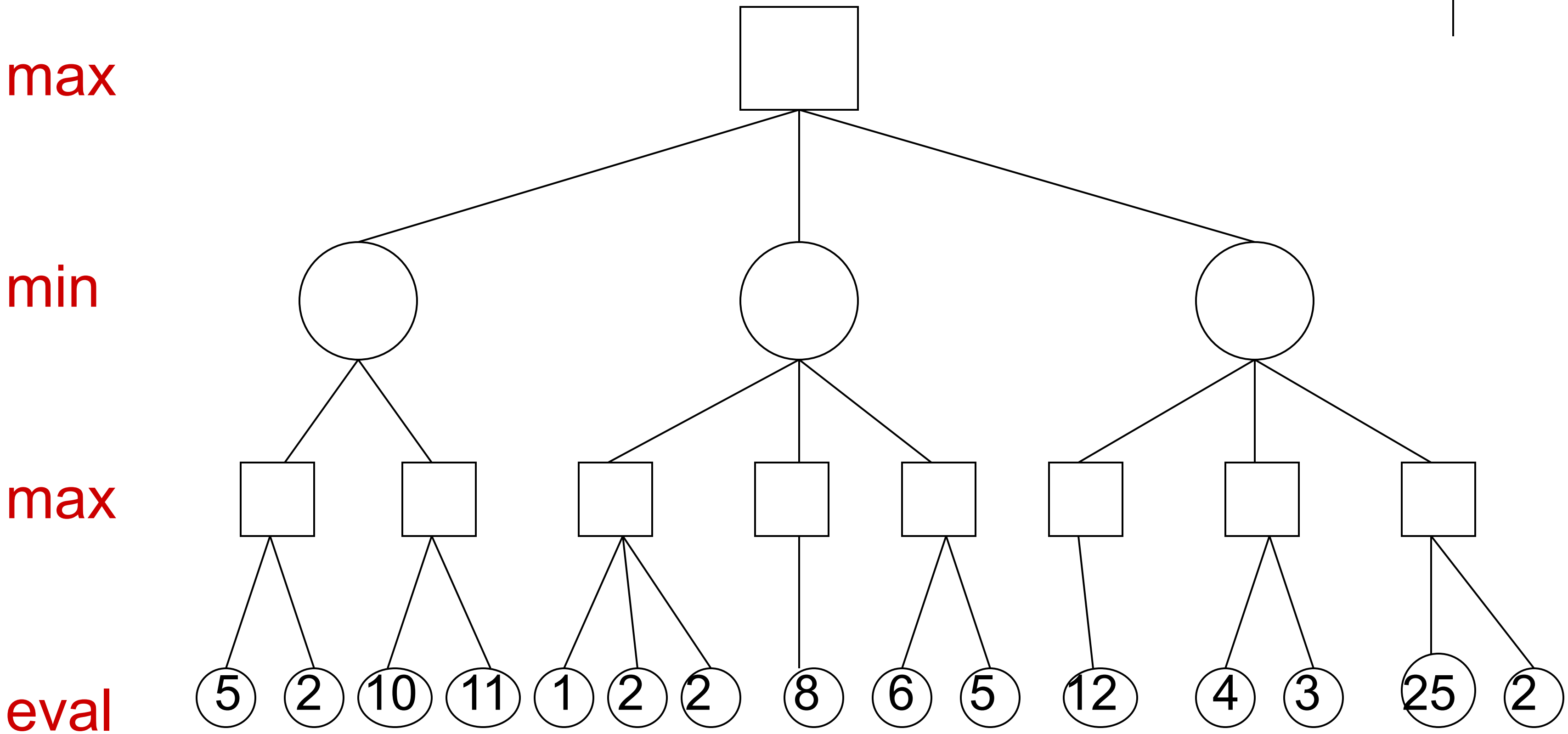
E is α pruned, since $2 < 3$

C is 3

States without numbers are not evaluated

+If we already implemented MINI-MAX algorithm correctly, how can we verify we correctly implemented Alpha-beta pruning?

Alpha-Beta Pruning Practice



References

- George Fluger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6^{th} edition, **Chapter 4**, Addison Wesley, 2009.
- Russel and Norvig, Artificial Intelligence: A Modern Approach, 3^{rd} edition, Prentice Hall, 2010.