Multi-agent systems and The Game Theory

Games in Normal Form, Games in Extensive Form

Ing. Tomas Borovicka

Department of Theoretical Computer Science (KTI), Faculty of Information Technology (FIT)
Czech Technical University in Prague (CVUT)

BIE-ZUM, LS 2013/14, 10. lecture

https://edux.fit.cvut.cz/courses/BIE-ZUM/

Summary of Previous Lecture

- Data Mining Algorithms:
  - Apriori
  - Nearest Neighbor Classification
  - Naive Bayes
  - Decision Tree
  - k-means

Multi-agent System

Multi-agent system is a collection of semi-autonomous sub-agents in shared environment, such that each agent

- **perceives** the environment,
- **acts** flexibly to reach its **objectives**,
- **interacts** with other agents
  - **cooperates** or **competes**.

Agent Definition

Russel & Norwig

An intelligent agent perceives its environment via sensors and acts rationally upon that environment with its effectors.

Maes

Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

Hayes-Roth

Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.

Agent Definition

Wooldridge & Jennings

An agent is an entity which is: Situated in some environment.

- Autonomous, in the sense that it can act without direct intervention from humans or other software processes, and controls over its own actions and internal state.
- Flexible which means:
  - Responsive (reactive): agents should perceive their environment and respond to changes that occur in it;
  - Proactive: agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behavior and take the initiative when appropriate;
  - Social: agents should be able to interact with humans or other artificial agents.

Agent Features

autonomous control over its own actions,

goal-oriented realize a set of goals,

reactive reacts on changes in the environment,

proactive initiative goal-directed behavior,

communicative communicates with other agents, perhaps including humans,

learning changes its behavior based on its previous experience,

mobile able to move to another place / machine,

Agent Function

Agent function

Behavior of an agent is describe by **agent function**, which maps any given percept sequence to an action:

$$f : \mathcal{P} \to \mathcal{A}.$$

The agent function is implemented by an **agent program**.

- The agent function is an abstract mathematical description.
- Agent program is concrete implementation running within some physical system.

Basic Types of Agent programs

Rozlišujeme 4 úrovně agentů z hlediska komplexnosti:

1. **Simple reflex agent**
   - select actions on the basis of the current percept, ignoring the rest of the percept history,
   - fully observable environment,
   - controlled by condition–action / **if-then** rules,
2. **Model-based reflex agent**
   - internal state hat depends on the percept history,
   - partially observable environment,
   - model of the environment, that helps to determine the current state of a partially observable environment
3. **Goal-based agent**
   - information that describes situations that are desirable (goals),
   - uses search and planning to find action sequences that achieve the goals,
4. **Utility-based agent**
   - uses **utility function**
     - ★ a mapping from states of the world to real numbers,
     - ★ indicating the agent's level of happiness with that state of the world.

Utility Function

- Utility-based agent acts rationally if prefers actions that maximize its utility.
- An utility is a numeric value representing how 'good' the state is.

Utility function

An **utility function** is a function which associates a real value with every environment state:

$$u\colon S \rightarrow \mathbb{R}$$

such that $u(s_1) \geq u(s_2)$ iff the agent prefers s_1 to s_2, i.e. $s_1 \succeq s_2$.

Lottery

An agent may not know the outcomes of his actions, but may instead only have a probability distribution over the outcomes.

Lottery

A **lottery** is a probability distribution over outcomes:

$$[p_1 : o_1, p_2 : o_2, \ldots, p_k : o_k],$$

where o_i are outcomes and $p_i > 0$ are probabilities such that

$$\sum_{i=1}^{k} p_i = 1.$$

- The lottery specifies that outcome o_i occurs with probability p_i.
- We will consider lotteries to be outcomes.

Agent's Rationality

- Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of agents such that each agent selects actions which outcome is given by lottery $\ell \in \mathcal{L}$.
- Consider that each agent A_i has an utility function u_i such that $u_i(o_j)$ is an utility of outcome o_j for an agent A_i.

Agent's Rationality

Self-interested rational agent

Self-interested rational agent is an agent A_i that selects the action that maximize its individual utility, i.e. executing the lottery ℓ^* that maximize the expected utility.

$$\ell^* \in \arg\max_{\ell \in \mathcal{L}} \sum_{(p_j :\ o_j) \in \ell} p_j \cdot u_i(o_j)$$

Cooperative rational agent

Cooperative rational agent is an agent A_i, that selects the action that maximize collective utility of all agents $A_i \in \mathcal{A}$, i.e. executing the lottery ℓ^* that maximize the expected utility of all agents:

$$\ell^* \in \arg\max_{\ell \in \mathcal{L}} \sum_{A_k \in \mathcal{A} \setminus \{A_i\}} \sum_{(p_j :\ o_j) \in \ell} p_j \cdot u_k(o_j) + \sum_{(p_j :\ o_j) \in \ell} p_j \cdot u_i(o_j).$$

Game Theory

- Mathematical study of interaction between rational,
  - Formal description, analyzing and choosing optimal strategy...

  self-interested agents.
- Basic categories:
  - **Cooperative games** – modeling unit is a team, where agents have the same interest.
  - **Non-cooperative games** – modeling unit is an individual that pursue their own interests.
- Theoretical description in:
  - **normal form** – the game is represented by **matrix**,
  - **extensive form** – the game is represented by **game tree**

Games in Normal Form

Finite game in normal form

Finite game in normal form for n players is a triplet $(\mathcal{N}, \mathcal{A}, u)$, where

- $\mathcal{N} = \{N_1, \ldots, N_n\}$ is a set of **players**,
- $\mathcal{A} = A_1 \times \ldots \times A_n$, where A_i is the **action set** for player N_i,
  - $\mathbf{a} \in \mathcal{A}$ is an **action profile**, and so A is the space of action profiles,
- $u = (u_1, \ldots, u_n)$ is a utility function, where u_i denotes utility function for player N_i.
  - $u_i \colon \mathcal{A} \to \mathbb{R}$,
  - $u_i(\mathbf{a})$ denotes utility of player N_i with action profile $\mathbf{a} \in \mathcal{A}$

Rock-paper-scissors

N_1 \ N_2	R	P	S
R	**0, 0**	**-1, 1**	**1, -1**
P	**1, -1**	**0, 0**	**-1, 1**
S	**-1, 1**	**1, -1**	**0, 0**

$$\mathcal{N} = \{N_1, N_2\}$$

$$\begin{aligned} \mathcal{A} = \{R,P,S\} \times \{R,P,S\} = \{ & (R,R),(R,P),(R,S), \\ & (P,R),(P,P),(P,S)\ \} \\ & (S,R),(S,P),(S,S), \end{aligned}$$

$$\begin{array}{llll} u_1: & (R,R) \mapsto 0, & (R,P) \mapsto -1, & (R,S) \mapsto 1, \\ & (S,R) \mapsto -1, & (S,P) \mapsto 1, & (S,S) \mapsto 0, \\ & (P,R) \mapsto 1, & (P,P) \mapsto 0 & (P,S) \mapsto -1, \end{array}$$

$$\begin{array}{llll} u_2: & (R,R) \mapsto 0, & (R,P) \mapsto 1, & (R,S) \mapsto -1, \\ & (S,R) \mapsto 1, & (S,P) \mapsto -1, & (S,S) \mapsto 0, \\ & (P,R) \mapsto -1, & (P,P) \mapsto 0 & (P,S) \mapsto 1, \end{array}$$

Coordination Games

Common-payoff

Common-payoff game

Let $G = (\mathcal{N}, \mathcal{A}, u)$ be a game in normal form, then G is a **common-payoff** game iff

$$\forall \mathbf{a} \in \mathcal{A}\colon\ u_1(\mathbf{a}) = u_2(\mathbf{a}) = \ldots = u_n(\mathbf{a}).$$

Example:

- Choosing sides

N_1 \ N_2	L	R
L	1, 1	0, 0
R	0, 0	1, 1

Competition games

Constant-sum

Constant-sum game

Let $G = (\mathcal{N}, \mathcal{A}, u)$ be a game in normal form, then G is a **constant-sum** game iff

$$\exists c \in \mathbb{R}\colon \forall \mathbf{a} \in \mathcal{A}\colon \sum_{i=1}^{n} u_i(\mathbf{a}) = c.$$

Special case in which $c = 0$ is called **zero-sum** game.

Example:

- Matching pennies

N_1 \ N_2	H	T
H	1, -1	-1, 1
T	-1, 1	1, -1

Which action profiles are interesting?

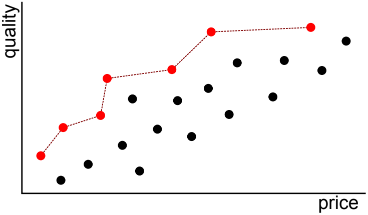
- **Pareto Optimality**
  - There is no other action profile that would increase utility of any player without reducing utility of at least one player.
- **Nash Equilibrium**
  - Any player can not increase its utility by changing action profile.
  - Players are in equilibrium, a change by any player would lead to decrease in its utility.

Pareto Optimality

- Multi-criteria optimization.
  - More than one objective function to be optimized simultaneously.
- Trade-offs between two or more conflicting objectives.
  - e.g. minimizing cost while maximizing quality.
    - cheap products are usually of poor quality,
    - good quality products are expensive...

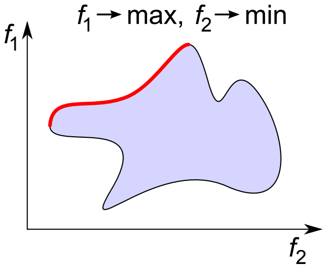
Pareto Optimality: Example

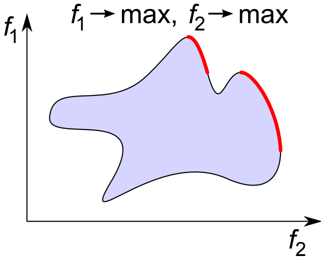
- Consider category of products on the market that you can get in various quality and price.

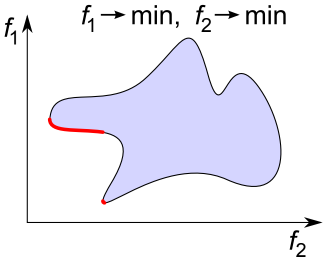


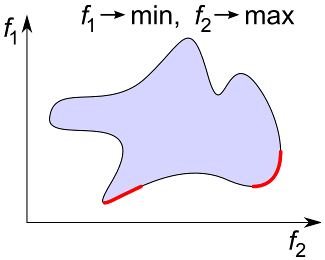
- Products labeled with red color **are** Pareto-optimal, because:
  - there is no product at lower or equal price with higher quality,
  - there is no product with higher or equal quality at a lower price.
- Products labeled with black color **are not** Pareto-optimal, because:
  - there is a higher quality product at a lower or equal price (or)
  - there is a product at lower price with equal or higher quality.

Pareto Optimal Borders









Pareto Optimality in Game Theory

Pareto-dominance

Consider a game in normal form $(\mathcal{N}, \mathcal{A}, u)$. We say that action profile $\mathbf{a}' = (a'_{N_1}, \ldots, a'_{N_n}) \in \mathcal{A}$ **Pareto-dominates** action profile $\mathbf{a} = (a_{N_1}, \ldots, a_{N_n}) \in \mathcal{A}$ iff:

1. $\forall i \in \{1, \ldots, n\}: u_i(\mathbf{a}') \geq u_i(\mathbf{a})$,
2. $\exists i \in \{1, \ldots, n\}: u_i(\mathbf{a}') > u_i(\mathbf{a})$.

Pareto-optimality

Let $(\mathcal{N}, \mathcal{A}, u)$ be a game in normal form. Action profile $\mathbf{a}^* \in \mathcal{A}$ is Pareto-optimal, if there is no action profile $\mathbf{a}' \in \mathcal{A}$ that Pareto-dominates it.

Example: Pareto-optimal Action profiles

- Consider a game in normal form with following game matrix.
- Pareto-optimal action profiles are labeled by green color.

N_1 \ N_2	A	B	C	D
E	6, 3	8, 2	8, 3	7, 1
F	3, 2	4, 5	6, 4	6, 5
G	4, 5	0, 8	5, 7	6, 1

Nash equilibrium

Best response

Games in normal form assume **limited observability**

- Players selects actions independently to each other.
- If the player would knew what everyone else was going to do, it would be easy to pick an action.

Best Response

Consider a game in normal form $(\mathcal{N}, \mathcal{A}, u)$ action profile $\mathbf{a} = (a_{N_1}, a_{N_2}, \ldots, a_{N_n})$ of player $N_i \in \mathcal{N}$ and its utility function u_i.

Let

$$\mathbf{a}_{-i} = (a_{N_1}, \ldots, a_{N_{i-1}}, a_{N_{i+1}}, \ldots, a_{N_n})$$

be an action profile with actions of all players without N_i.

Then the **best response** is

$$BR(\mathbf{a}_{-i}) = \arg\max_{\hat{a}_{N_i} \in A_i} u_i((a_{N_1}, \ldots, a_{N_{i-1}}, \hat{a}_{N_i}, a_{N_{i-1}}, \ldots, a_{N_n}))$$

Nash equilibrium

Nash equilibrium

Consider a game in normal form $(\mathcal{N}, \mathcal{A}, u)$ and action profile $\mathbf{a} = (a_{N_1}, a_{N_2}, \ldots, a_{N_n})$. We say that **a** is **Nash equilibrium** iff

$$\forall i \in \{1, \ldots, n\} \colon a_{N_i} \in BR(\mathbf{a}_{-i}).$$

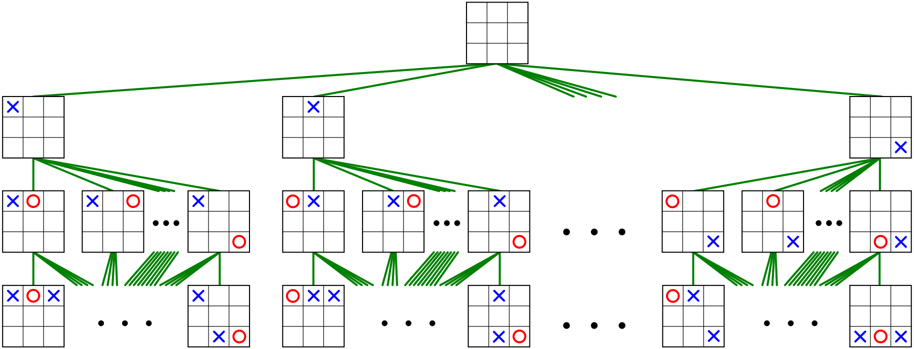
- Nash equilibrium is an action profile where action of each player is the best response.
  - Knowing the actions of the others all players are ”happy” with the action they selected.
  - Players are in equilibrium means that no player wants to change the action.

Example

- Consider a game in normal form with following game matrix.
- Nash equilibrium are labeled by gold color.

$N_1 \backslash N_2$	A	B	C	D
E	6, 3	8, 2	8, 3	**9, 8**
F	**7, 9**	4, 5	6, 4	6, 5
G	4, 5	**9, 9**	5, 7	6, 1

Games in Extensive Form



Examples

- tic-tac-toe,
- chess,
- checkers,
- reversi,
- go,
- ...

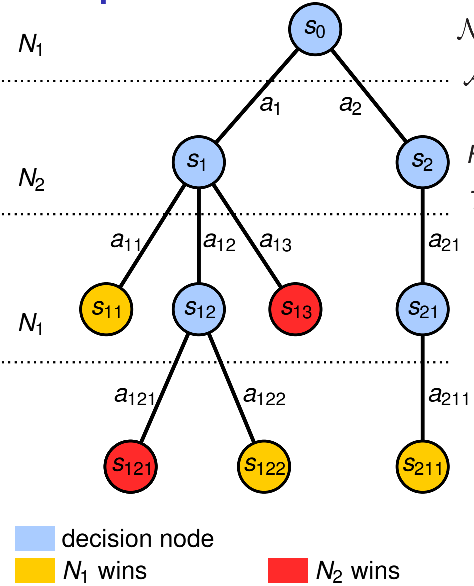
Game in Extensive Form

Finite Game in Extensive Form

A finite game in extensive form for n players is a tuple $(\mathcal{N}, \mathcal{A}, H, T, \chi, \rho, \sigma, u)$

- $\mathcal{N} = \{N_1, \ldots, N_n\}$ is a set of players,
- $\mathcal{A}$ is a set of actions,
- H is a set of decision nodes,
- $\chi \colon H \to 2^{\mathcal{A}}$ assigns a set of possible actions for each node,
- $\rho \colon H \to \mathcal{N}$ assigns to each non-terminal node a player whose turn,
- T is a set of terminal nodes, $T \cap H = \{\}$,
- σ is a **successor function** $\sigma \colon H \times \mathcal{A} \to H \cup T$
  - $\forall h_1, h_2 \in H \,\forall a_1, a_2 \in \mathcal{A} \colon \sigma(h_1, a_1) = \sigma(h_2, a_2) \Rightarrow (h_1 = h_2 \land a_1 = a_2)$,
  - Decision nodes form a game tree.
- $u = (u_1, \ldots, u_n)$, where $u_i \colon T \to \mathbb{R}$ is a utility function of player N_i in the terminal nodes.

Example of Trivial Game in Extensive Form



$$\mathcal{N} = \{N_1, N_2\}$$

$$\mathcal{A} = \{a_1, a_2, a_{11}, a_{12}, a_{13}, a_{21}, a_{121}, a_{122}, a_{211}\}$$

$$H = \{s_0, s_1, s_2, s_{12}, s_{21}\}$$

$$T = \{s_{11}, s_{13}, s_{121}, s_{122}, s_{211}\}$$

$$\chi: s_0 \mapsto \{a_1, a_2\}, s_1 \mapsto \{a_{11}, a_{12}, a_{13}\}, \ldots, s_{21} \mapsto \{a_{211}\}$$

$$\rho: s_0, s_{12}, s_{21} \mapsto N_1, \quad s_1, s_2 \mapsto N_2$$

$$\sigma: (s_0, a_1) \mapsto s_1, (s_0, a_2) \mapsto s_2, (s_1, a_{11}) \mapsto s_{11}, (s_1, a_{12}) \mapsto s_{12}, \ldots, (s_{21}, a_{211}) \mapsto s_{211}$$

$$u_1: s_{11}, s_{122}, s_{211} \mapsto 1, s_{121}, s_{13} \mapsto -1$$

$$u_2: s_{11}, s_{122}, s_{211} \mapsto -1, s_{121}, s_{13} \mapsto 1$$

Two Player Zero-sum Games

- Two player, zero sum games have a prominent position in game theory.

Two player zero-sum game

Two player zero-sum game in extensive form is a game $(\mathcal{N}, \mathcal{A}, H, T, \chi, \rho, \sigma, u)$, where:

1. $|\mathcal{N}| = 2$,
2. $u = (u_1, u_2)$,
3. $\forall t \in T\colon u_1(t) + u_2(t) = 0$.

Motivation: chess, checkers, tac-tac-toe, …

- Game finishes in terminal node in one of the following states:
  - player N_1 wins, player N_2 loses $\rightsquigarrow u_1(t) = 1, u_2(t) = -1$,
  - player N_1 loses, player N_2 wins $\rightsquigarrow u_1(t) = -1, u_2(t) = 1$,
  - draw $\rightsquigarrow u_1(t) = 0, u_2(t) = 0$.

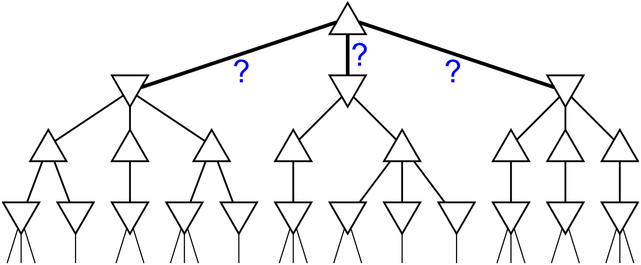
Size of a Game Tree

Game in extensive form induces a game tree, typically with huge number of nodes:

- **Tic-Tac-Toe**
  - trivial game,
  - 5478 valid configurations,
  - 255168 leafs in the game tree.
- **Checkers**
  - $\approx 10^{20}$ valid configurations,
  - $\approx 10^{40}$ leafs in the game tree.
- **Chess**
  - $\approx 10^{45}$ valid configurations,
  - $\approx 10^{123}$ leafs in the game tree.

Players *MIN* a *MAX*

- If we consider two player zero-sum game we can substitute utility functions of both players by one function $u\colon T \to \mathbb{R}$,
  - ***MAX*** – decision nodes in a game tree are marked by $\triangle$
    - players whose turn,
    - maximizes u,
  - ***MIN*** – decision nodes in a game tree are marked by $\triangledown$
    - opponent,
    - minimizes u,



Optimal Play

- Player can make
  - non-optimal move
    - player *MAX* does not select an action maximizing the minimal utility,
    - player *MIN* does not select an action minimizing the maximal utility,
    - **example**: player could win but selects an action that allows the opponent to win.
  - optimal move
    - player selects optimal action and its position is not worse,
    - **example**: player can win and chooses an action that lead to win.
- Player play **optimal (perfect) play** if in each turn selects an optimal action.
  - To play optimally is very difficult – combinatorial explosion.
  - It is not feasible to consider all possible actions.
  - Heuristics, limited depth of the game tree.

Perfect play

≫ *The unlimited intellect assumed in the theory of games, on the other hand, never makes a mistake and a smallest winning advantage is as good as mate in one. A game between two such mental giants, Mr. A and Mr. B, would proceed as follows. They sit down at the chessboard, draw the colours, and then survey the pieces for a moment. Then either*

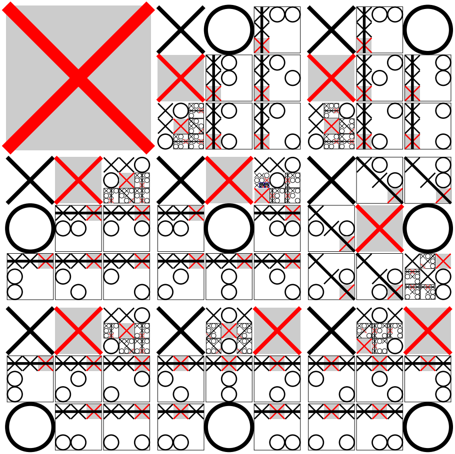
(1) Mr. A says, "I resign" or

(2) Mr. B says, "I resign" or

(3) Mr. A says, "I offer a draw," and Mr. B replies, "I accept." ≪

Claude E. Shannon, 1950

Perfect play for player $\times$ in Tic-Tac-Toe



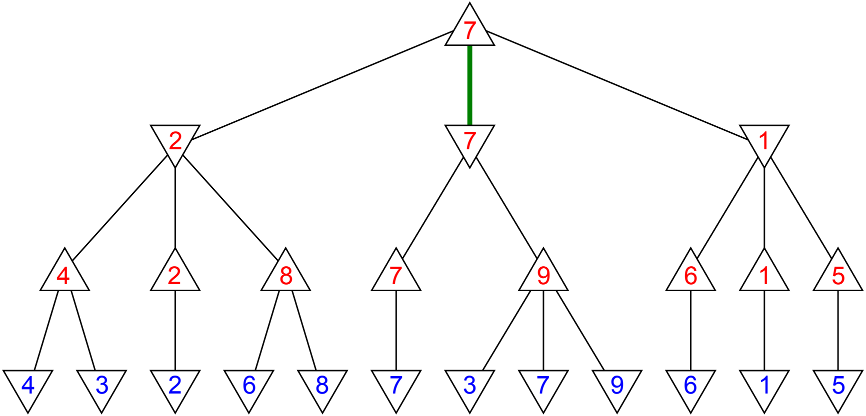
(Wikipedie)

Minimax Algorithm

- Selects the optimal action.
- Assumes that opponent plays optimally.

1. From the current decision generate complete game tree (or to depth equal to d) by in-order depth first traversing.
2. Evaluate each node:
   - $eval[x] \leftarrow u(x)$, if x is terminal or depth = d,
     - ★ $u(x)$ is either real utility, if x is terminal node, or heuristic if the expansion finished in depth d.
   - $eval[x] \leftarrow \max\limits_{a \in \chi(x)} eval[\sigma(x, a)]$, if x is *MAX* decision node,
   - $eval[x] \leftarrow \min\limits_{a \in \chi(x)} eval[\sigma(x, a)]$, if x is *MIN* decision node.
3. Return action $a \in \arg \max\limits_{a \in \chi(x_0)} eval[\sigma(x_0, a)]$.

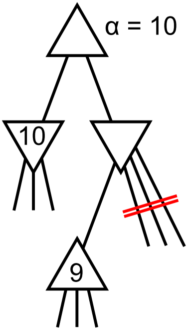
Minimax Example



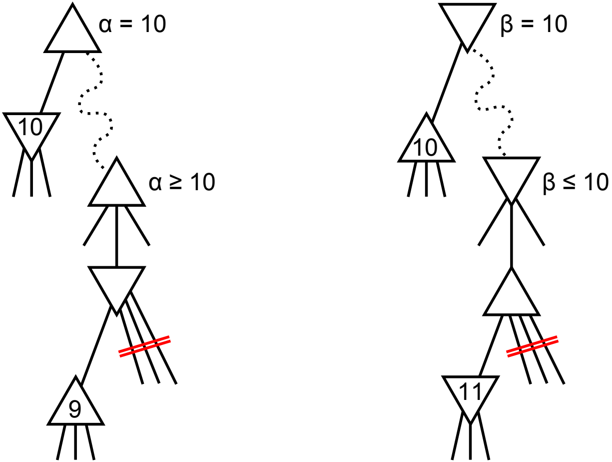
Alfa-beta Pruning

- Minimax needs to be optimized.
  - Searching the game tree is usually feasible only for small d.
  - for example average branching factor for chess is 35...
- With alfa-beta pruning algorithm keep two values for each expanded node
- α – the highest utility, that player *MIN* can not reduce if player *MAX* plays optimally...
- β – the lowest utility, that player *MAX* can not increase if player *MIN* plays optimally...

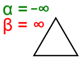
Alfa-beta Pruning



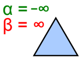
Alfa-beta Pruning



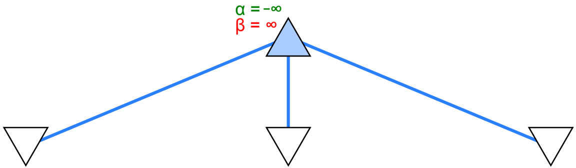
Minimax with alfa-beta prunning: Example



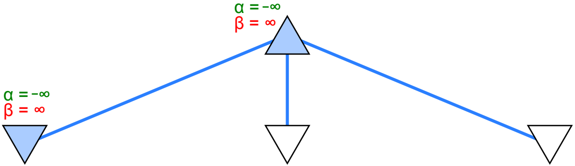
Minimax with alfa-beta prunning: Example



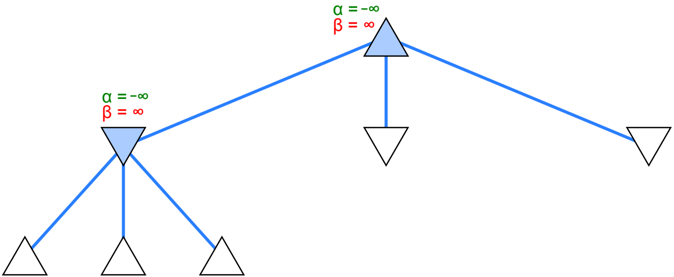
Minimax with alfa-beta prunning: Example



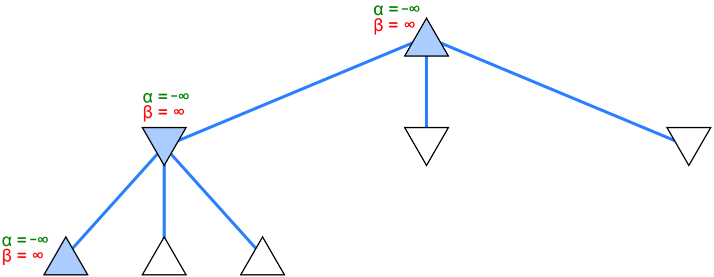
Minimax with alfa-beta prunning: Example



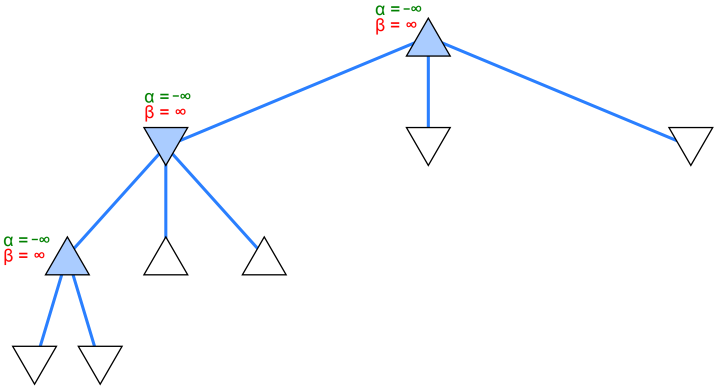
Minimax with alfa-beta prunning: Example



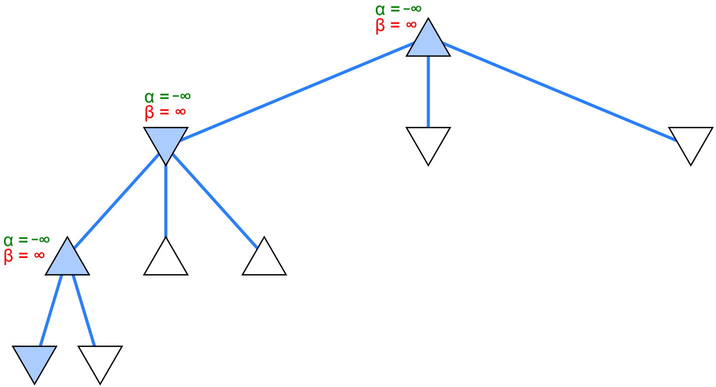
Minimax with alfa-beta prunning: Example



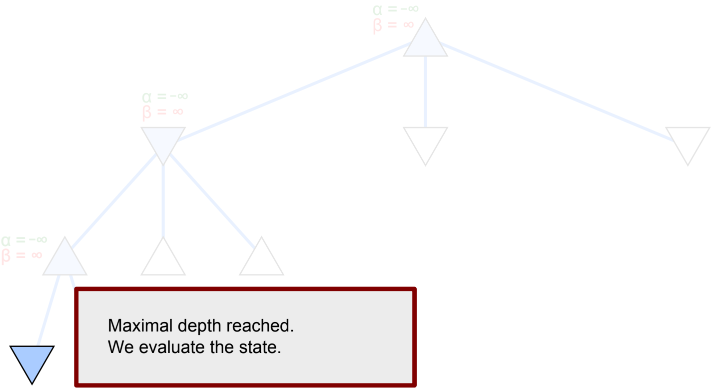
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example

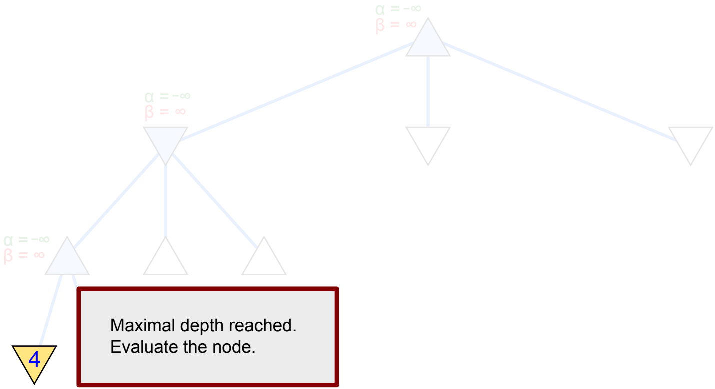


Minimax with alfa-beta prunning: Example



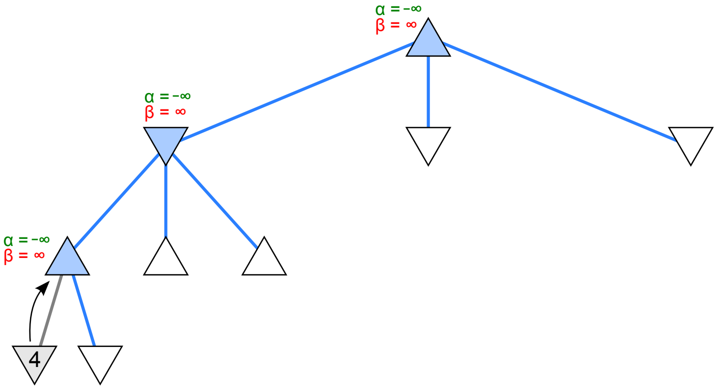
Maximal depth reached.
We evaluate the state.

Minimax with alfa-beta prunning: Example

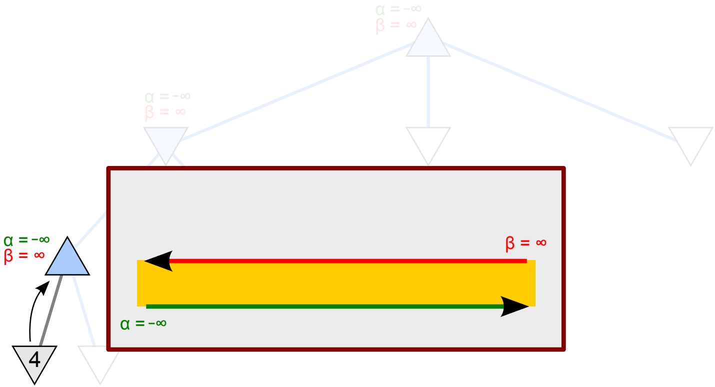


Maximal depth reached.
Evaluate the node.

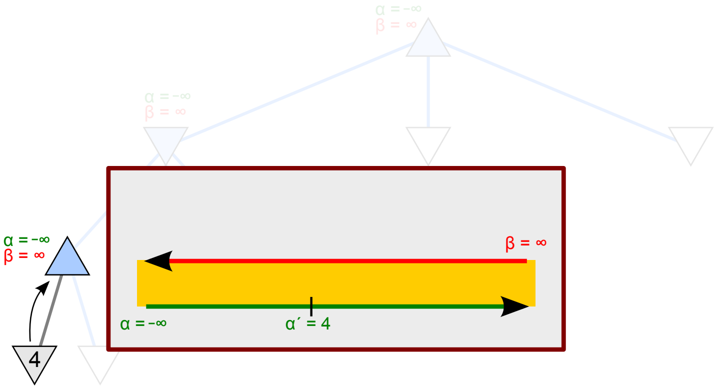
Minimax with alfa-beta prunning: Example



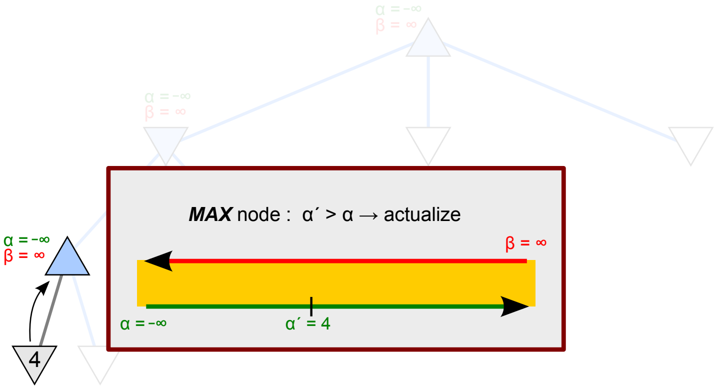
Minimax with alfa-beta prunning: Example



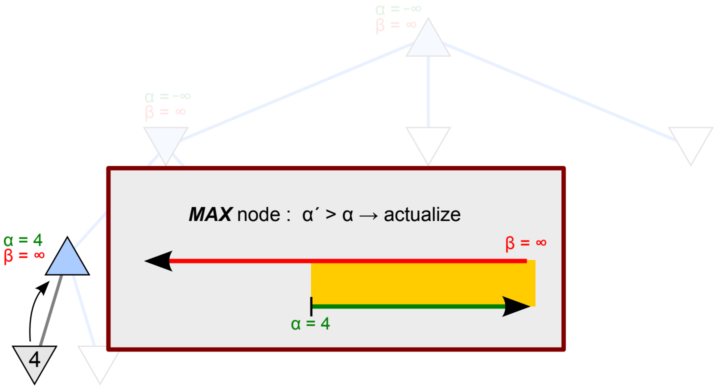
Minimax with alfa-beta prunning: Example



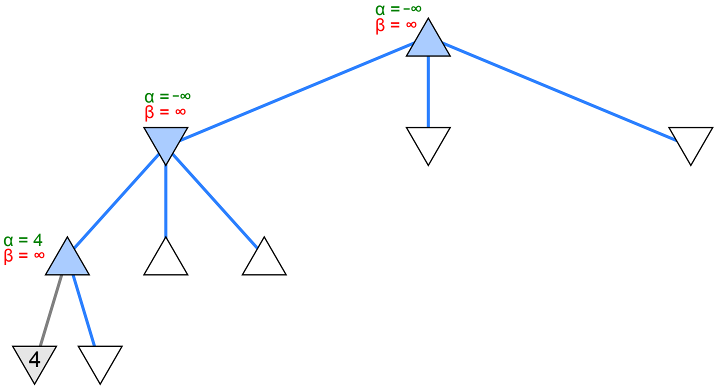
Minimax with alfa-beta prunning: Example



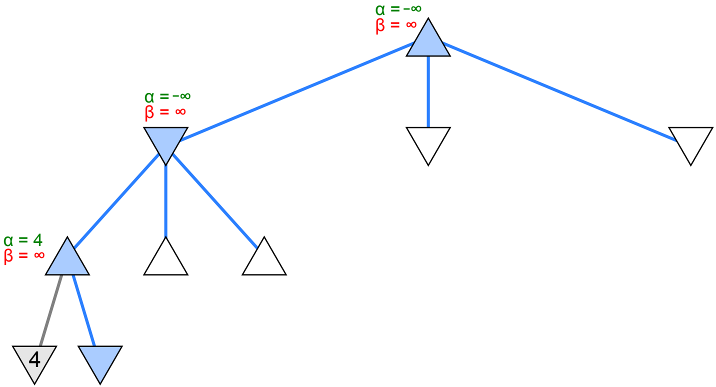
Minimax with alfa-beta prunning: Example



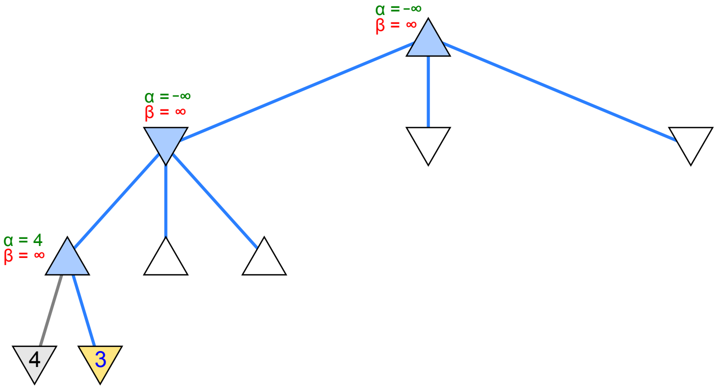
Minimax with alfa-beta prunning: Example



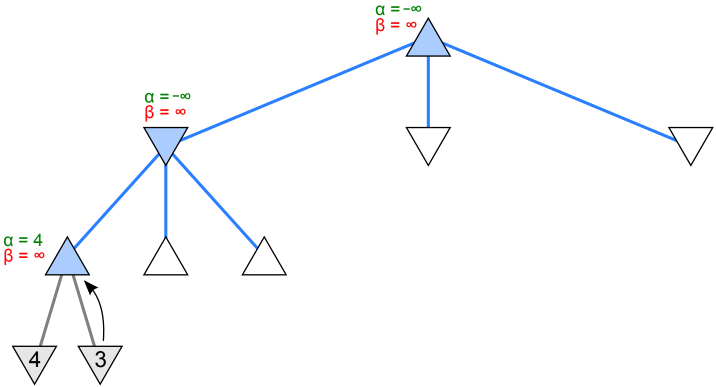
Minimax with alfa-beta prunning: Example



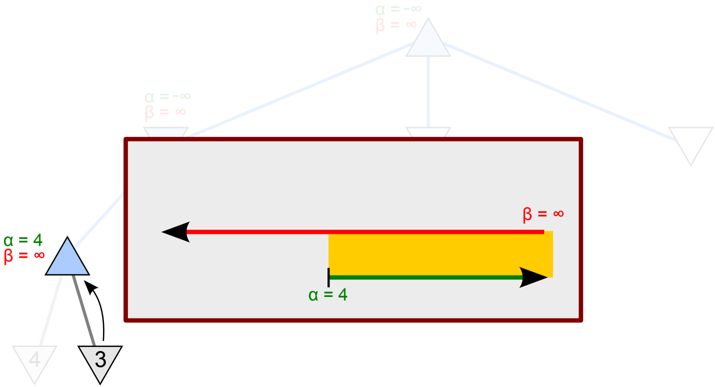
Minimax with alfa-beta prunning: Example



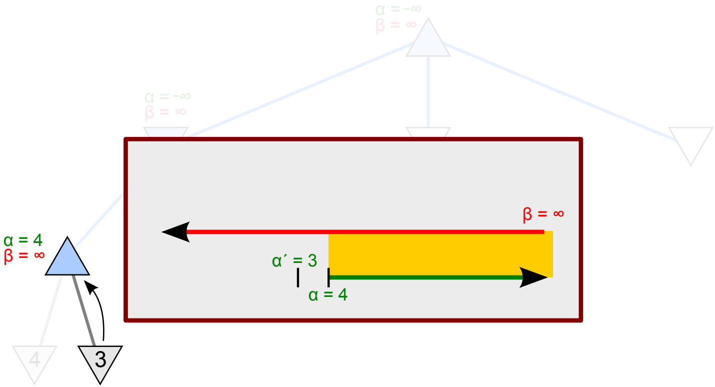
Minimax with alfa-beta prunning: Example



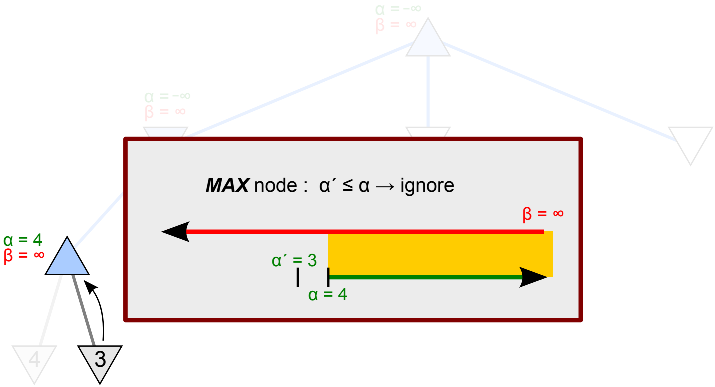
Minimax with alfa-beta prunning: Example



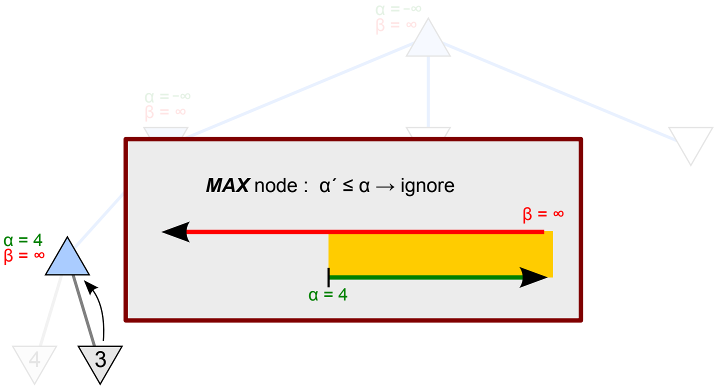
Minimax with alfa-beta prunning: Example



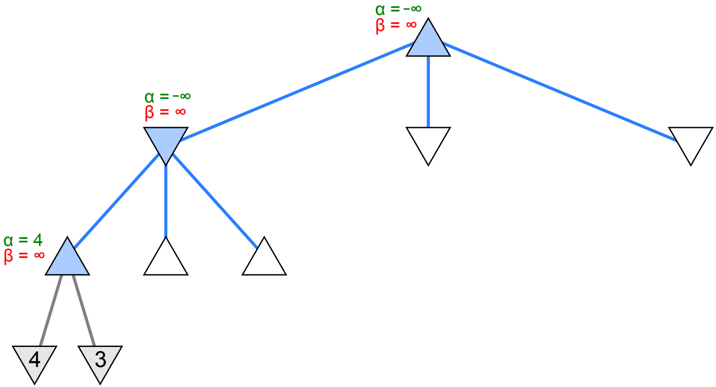
Minimax with alfa-beta prunning: Example



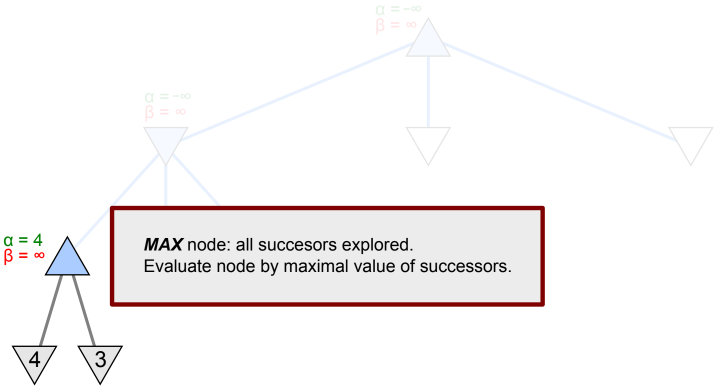
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example

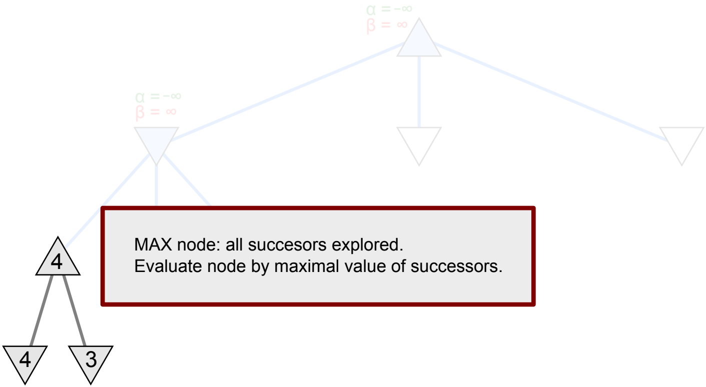


Minimax with alfa-beta prunning: Example



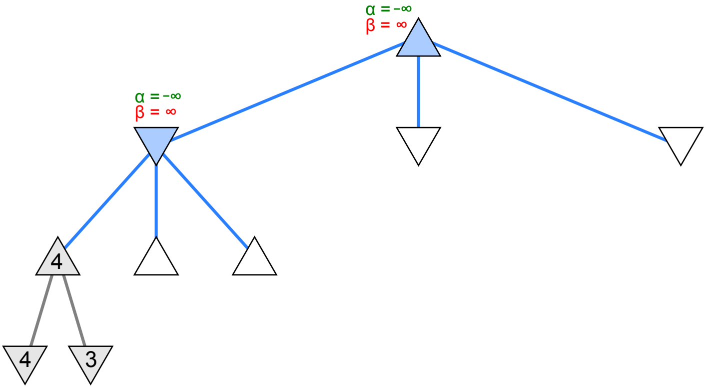
MAX node: all succesors explored.
Evaluate node by maximal value of successors.

Minimax with alfa-beta prunning: Example

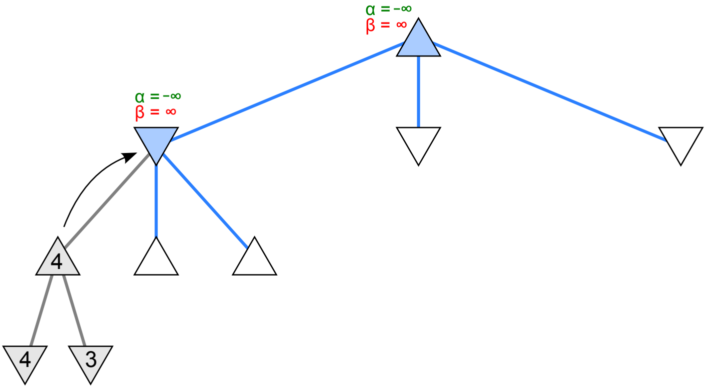


MAX node: all succesors explored.
Evaluate node by maximal value of successors.

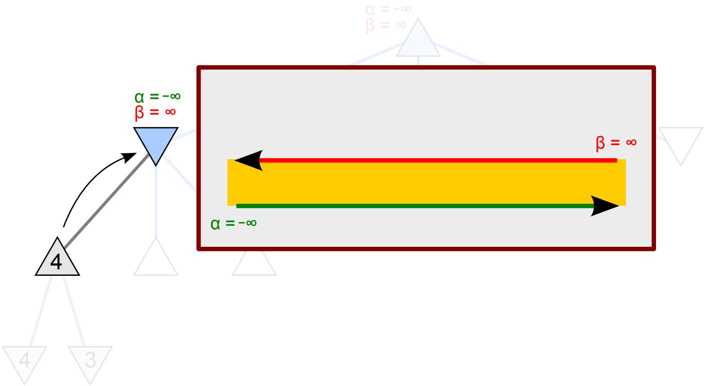
Minimax with alfa-beta prunning: Example



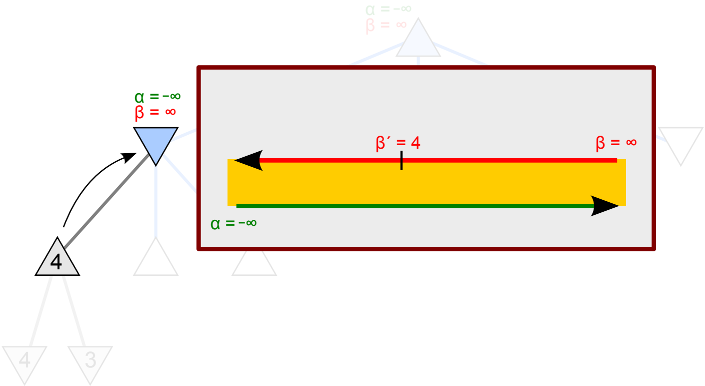
Minimax with alfa-beta prunning: Example



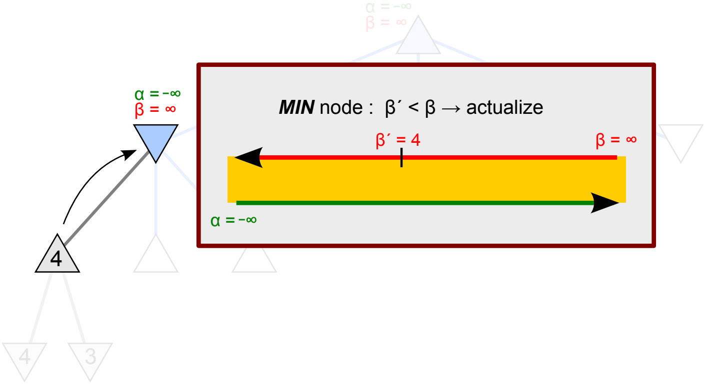
Minimax with alfa-beta prunning: Example



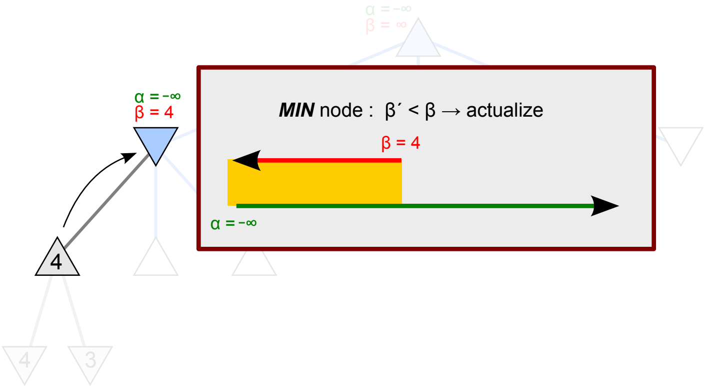
Minimax with alfa-beta prunning: Example



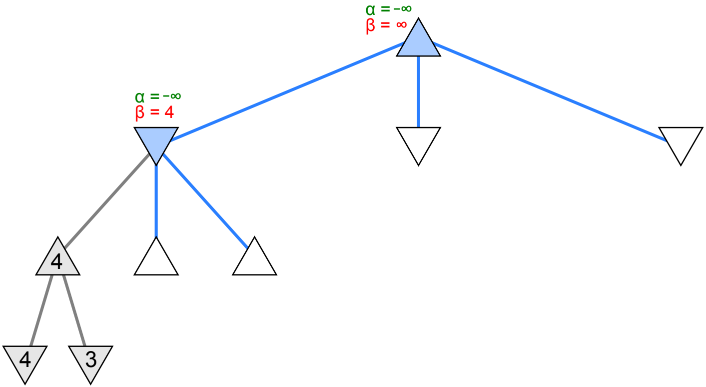
Minimax with alfa-beta prunning: Example



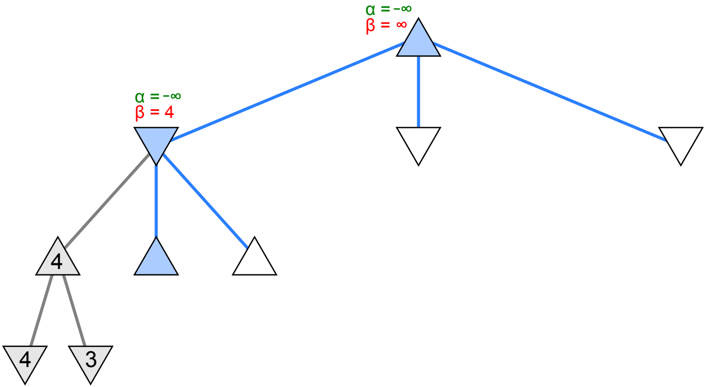
Minimax with alfa-beta prunning: Example



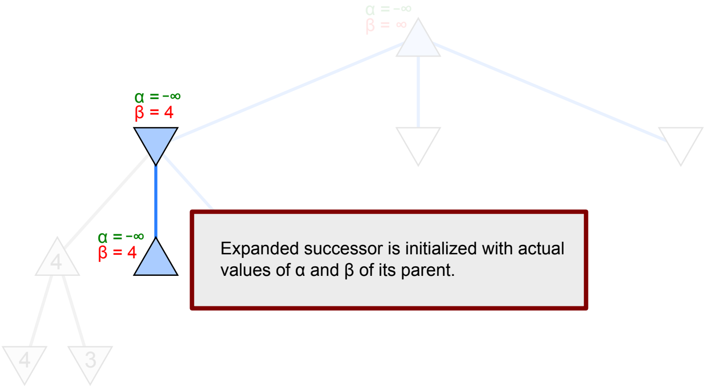
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example

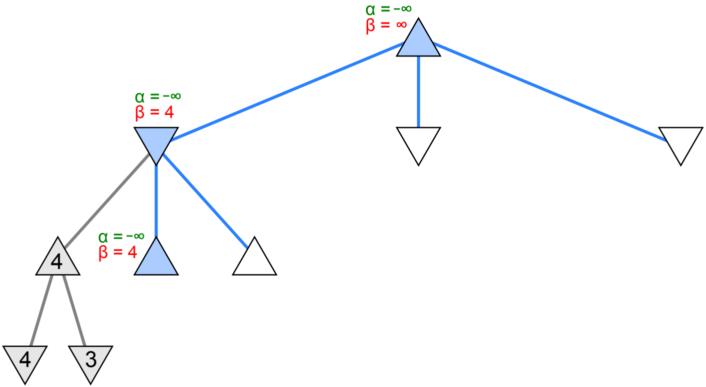


Minimax with alfa-beta prunning: Example

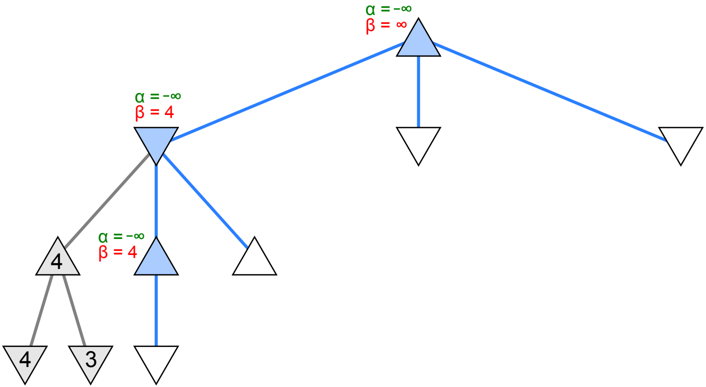


Expanded successor is initialized with actual values of α and β of its parent.

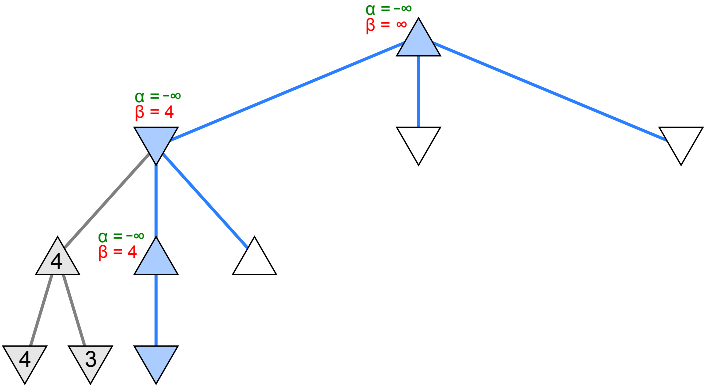
Minimax with alfa-beta prunning: Example



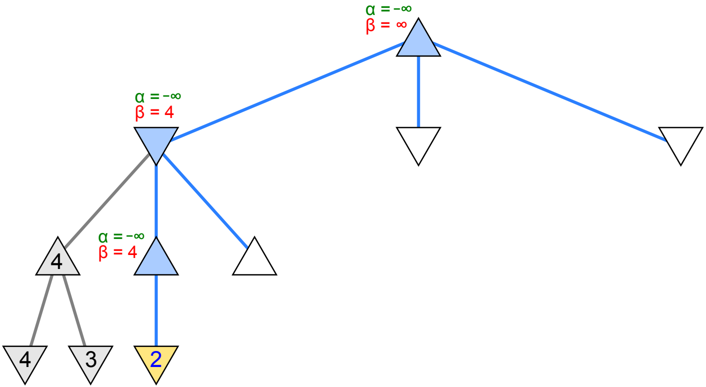
Minimax with alfa-beta prunning: Example



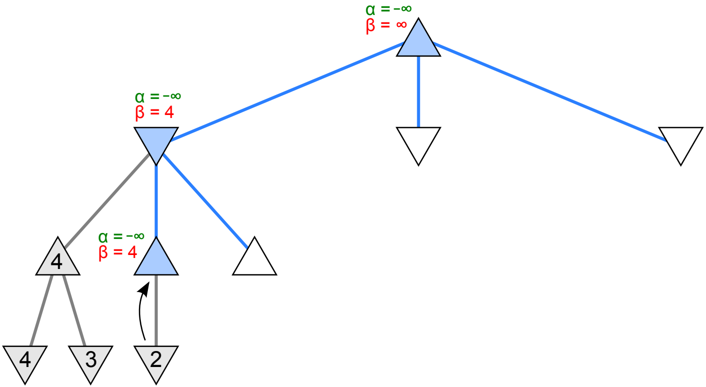
Minimax with alfa-beta prunning: Example



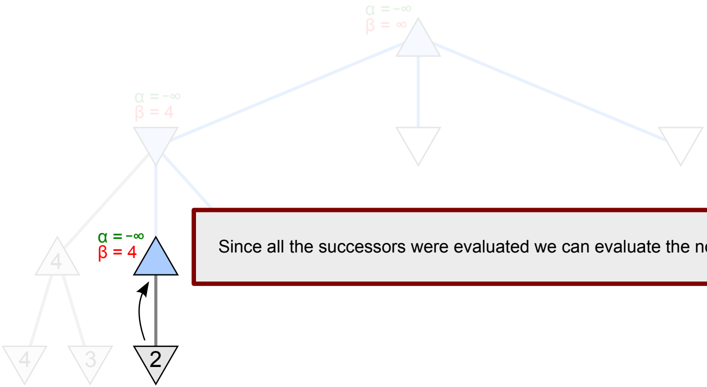
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example

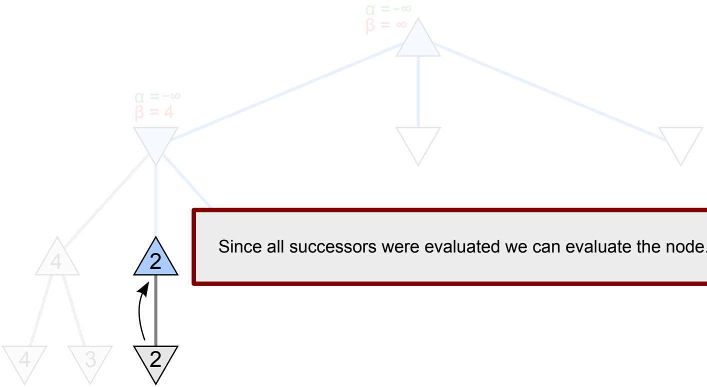


Minimax with alfa-beta prunning: Example



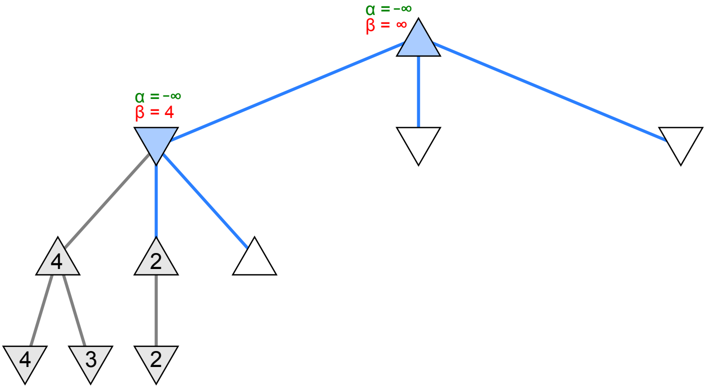
Since all the successors were evaluated we can evaluate the node.

Minimax with alfa-beta prunning: Example

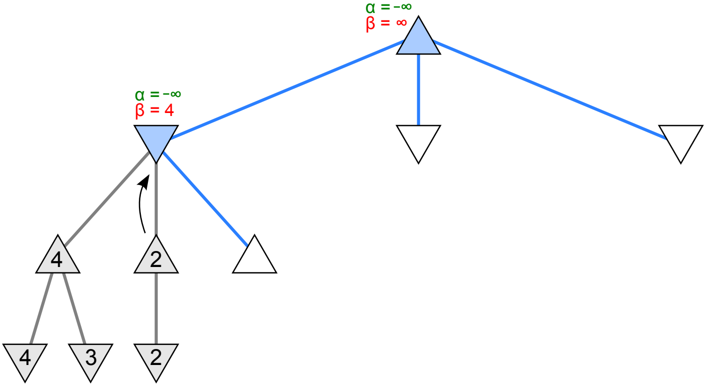


Since all successors were evaluated we can evaluate the node.

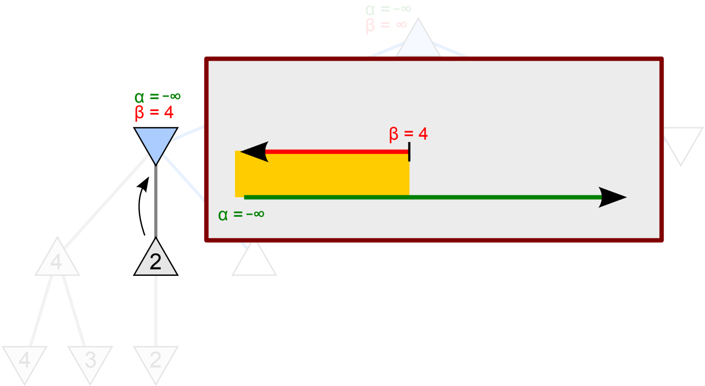
Minimax with alfa-beta prunning: Example



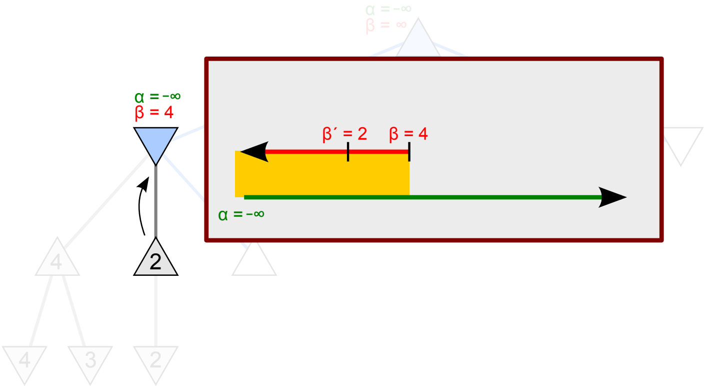
Minimax with alfa-beta prunning: Example



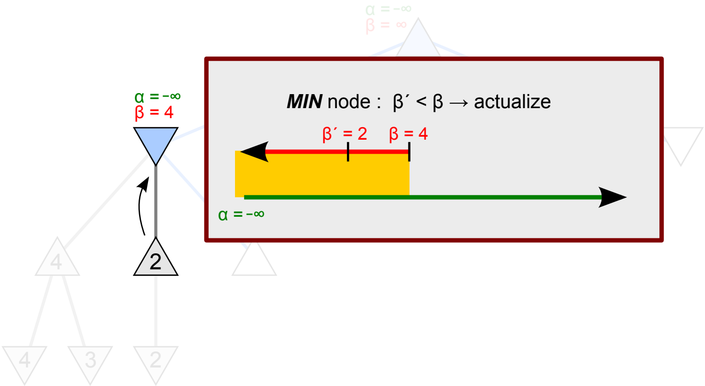
Minimax with alfa-beta prunning: Example



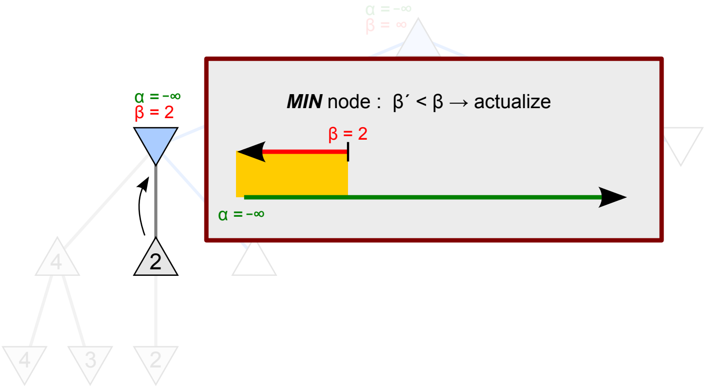
Minimax with alfa-beta prunning: Example



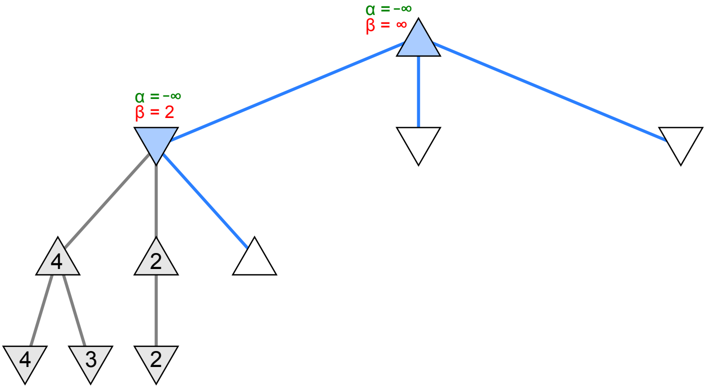
Minimax with alfa-beta prunning: Example



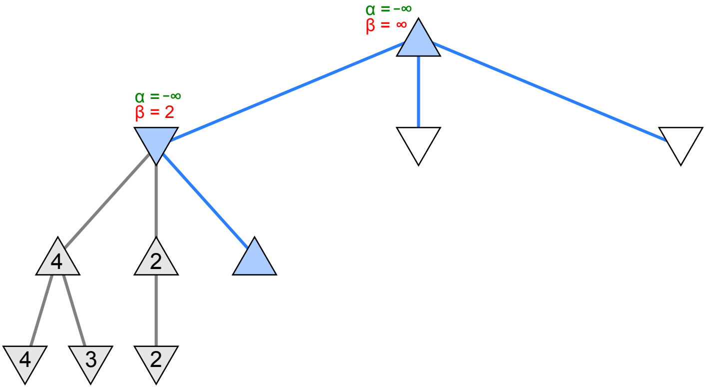
Minimax with alfa-beta prunning: Example



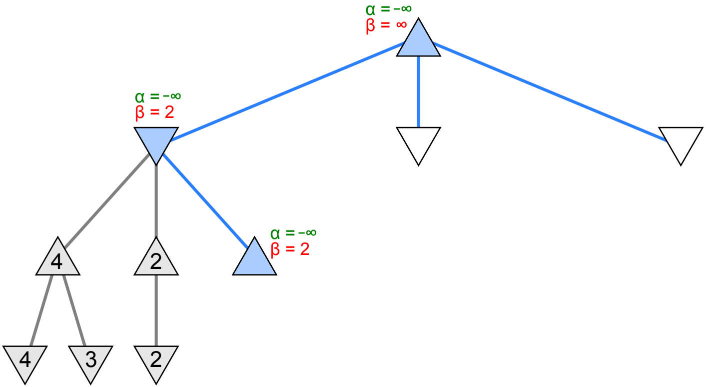
Minimax with alfa-beta prunning: Example



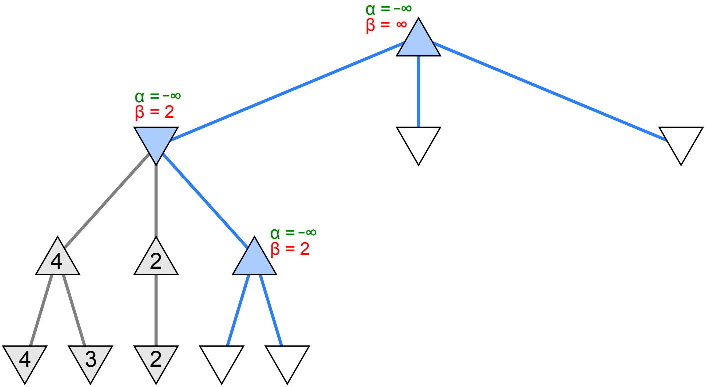
Minimax with alfa-beta prunning: Example



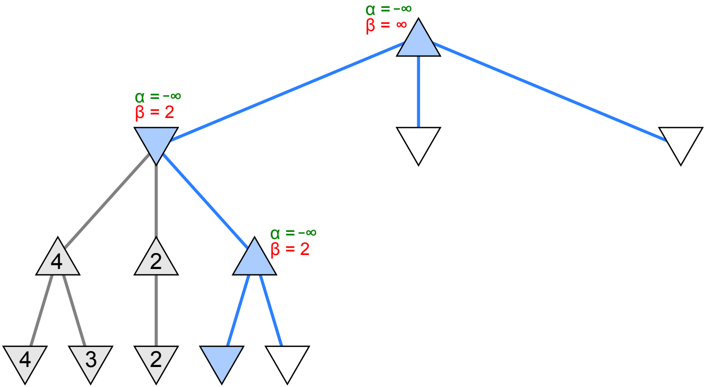
Minimax with alfa-beta prunning: Example



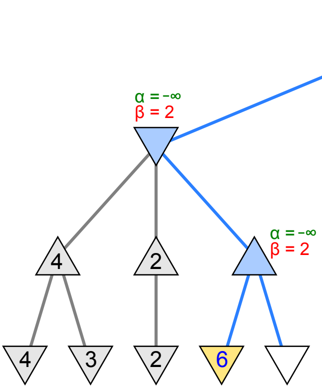
Minimax with alfa-beta prunning: Example



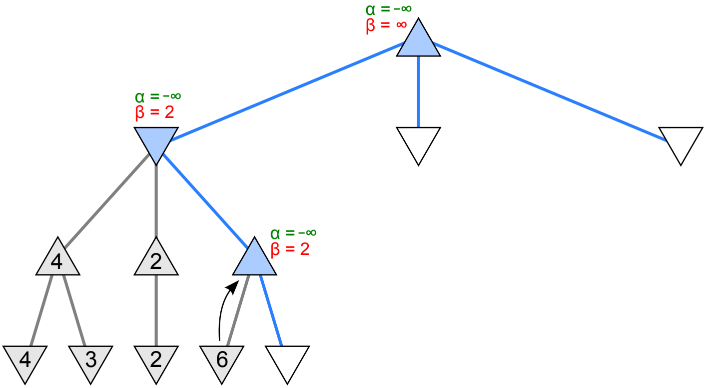
Minimax with alfa-beta prunning: Example



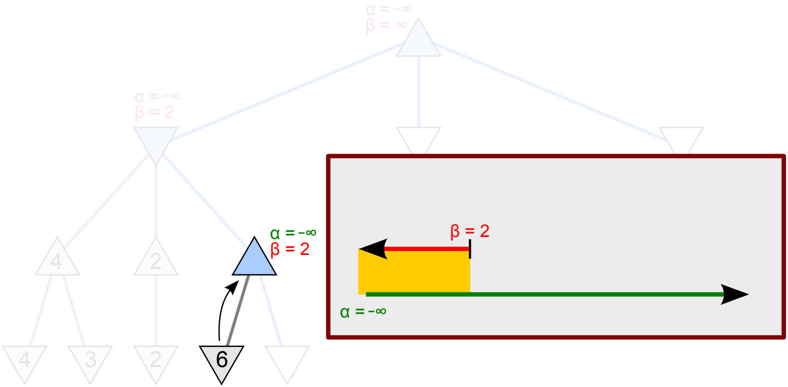
Minimax with alfa-beta prunning: Example



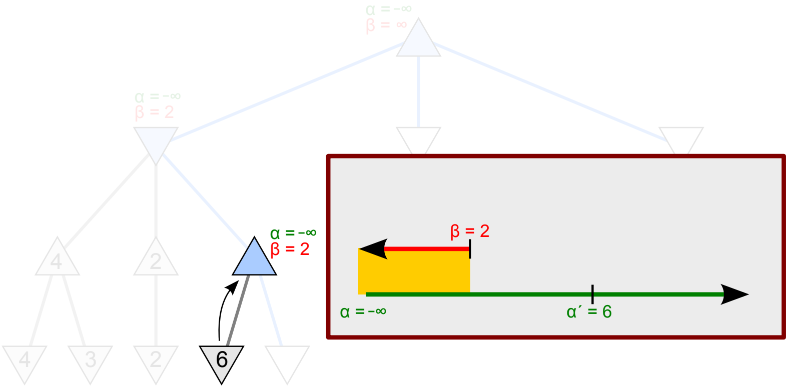
Minimax with alfa-beta prunning: Example



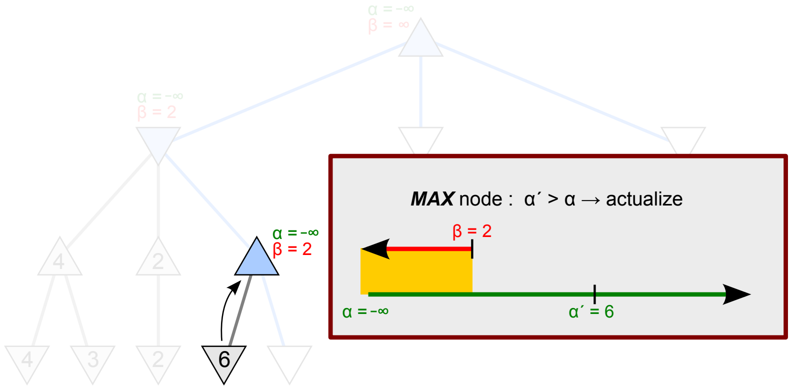
Minimax with alfa-beta prunning: Example



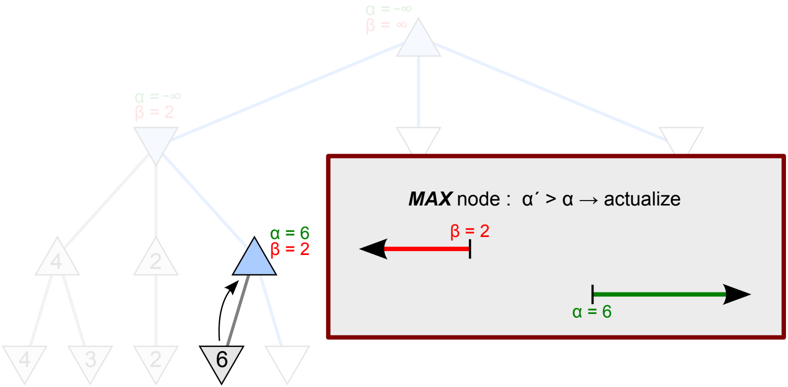
Minimax with alfa-beta prunning: Example



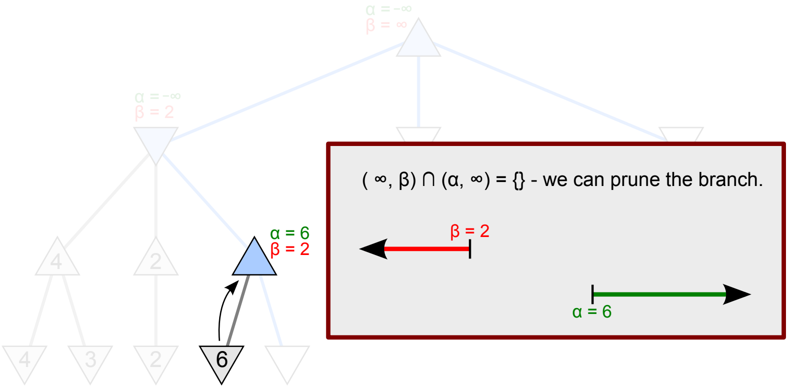
Minimax with alfa-beta prunning: Example



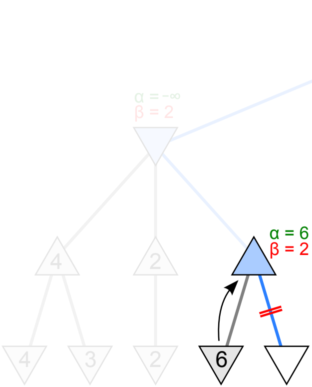
Minimax with alfa-beta prunning: Example



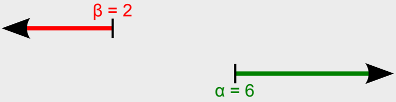
Minimax with alfa-beta prunning: Example



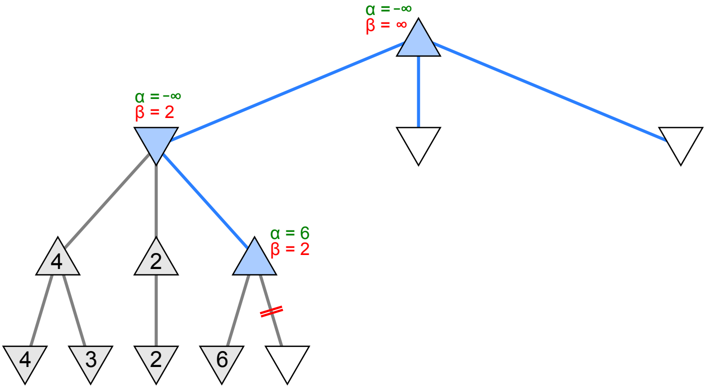
Minimax with alfa-beta prunning: Example



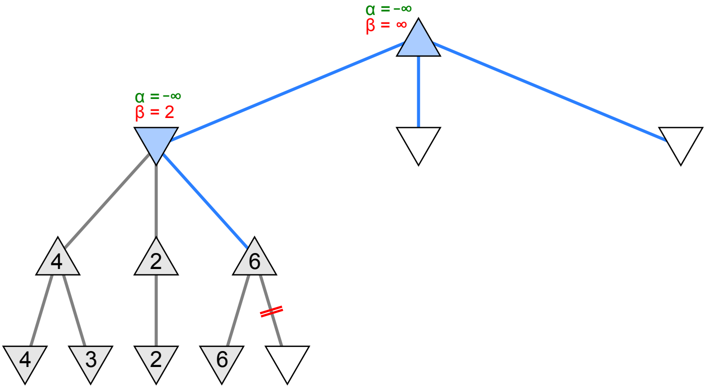
(∞, β) ∩ (α, ∞) = {} - we can prune the branch.



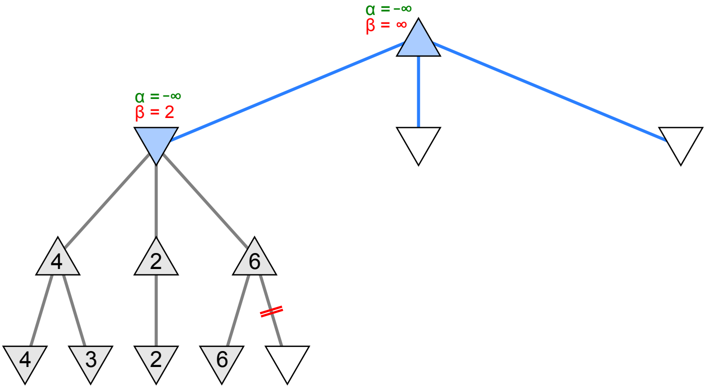
Minimax with alfa-beta prunning: Example



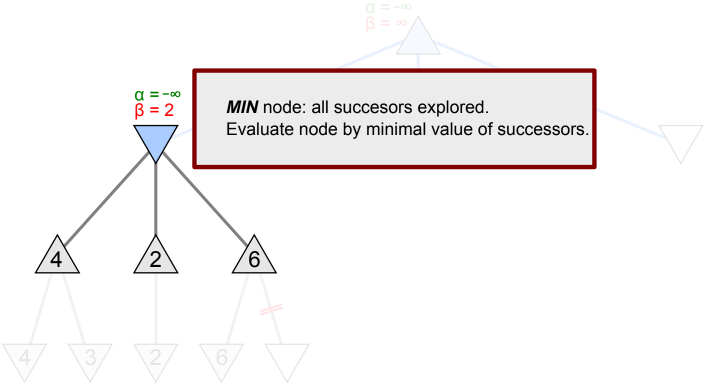
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example

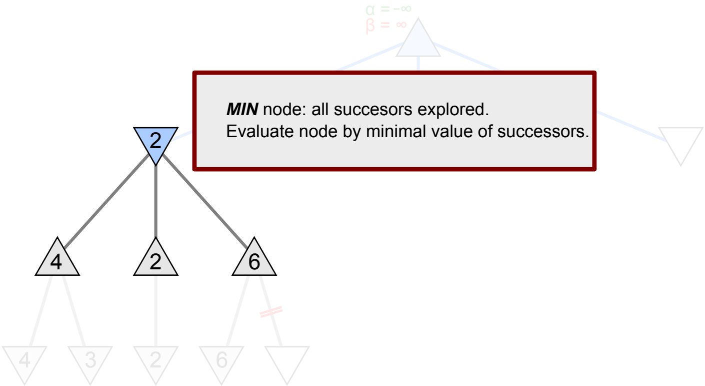


Minimax with alfa-beta prunning: Example



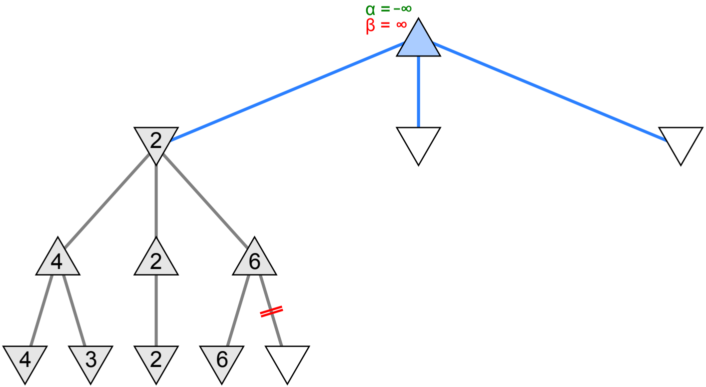
MIN node: all succesors explored.
Evaluate node by minimal value of successors.

Minimax with alfa-beta prunning: Example

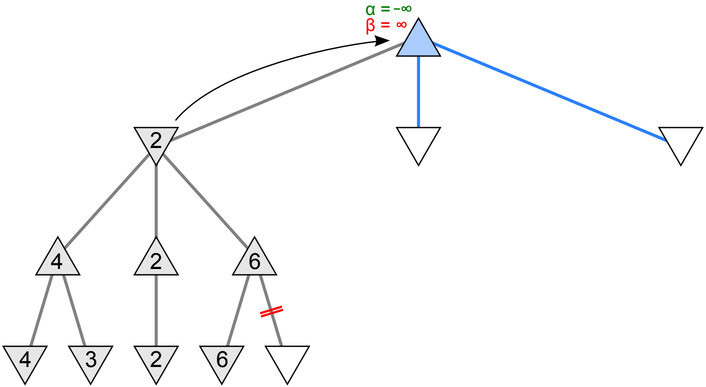


MIN node: all succesors explored.
Evaluate node by minimal value of successors.

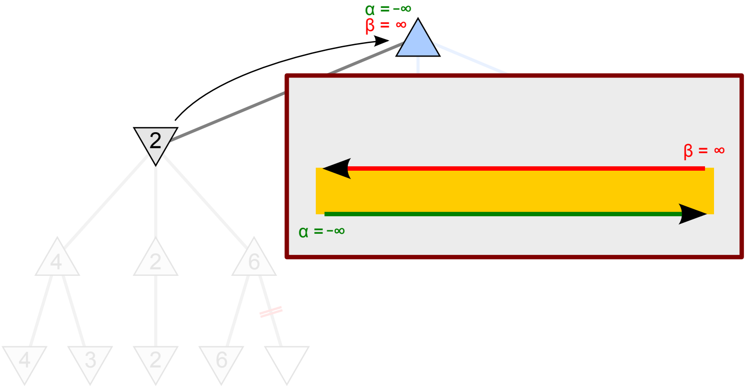
Minimax with alfa-beta prunning: Example



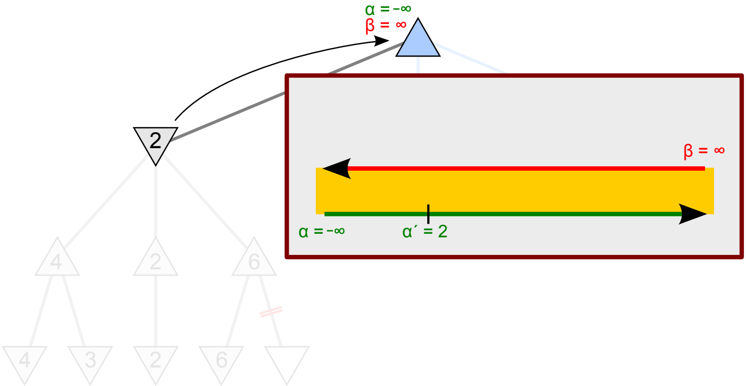
Minimax with alfa-beta prunning: Example



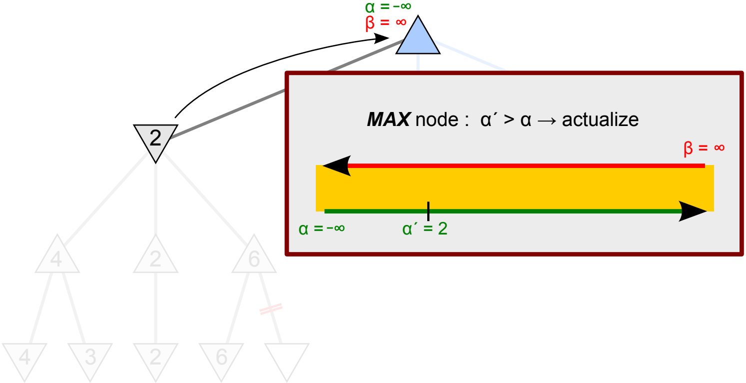
Minimax with alfa-beta prunning: Example



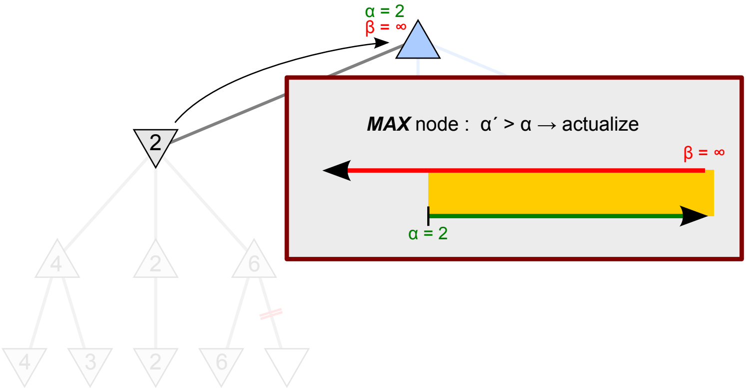
Minimax with alfa-beta prunning: Example



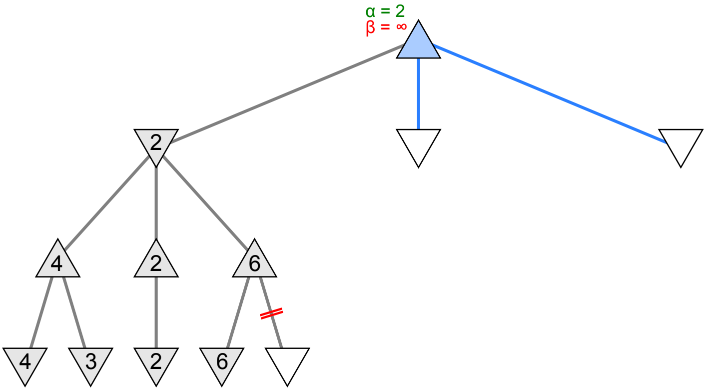
Minimax with alfa-beta prunning: Example



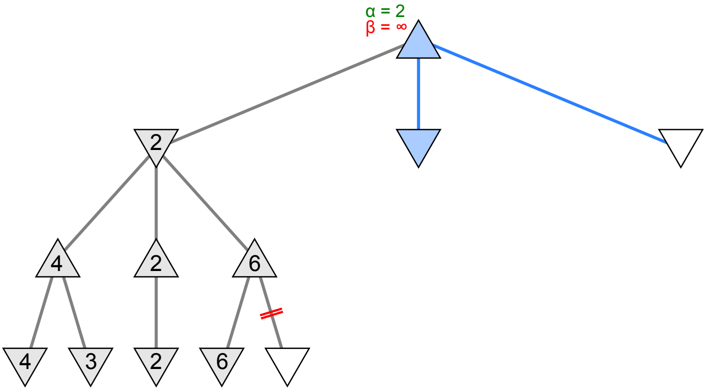
Minimax with alfa-beta prunning: Example



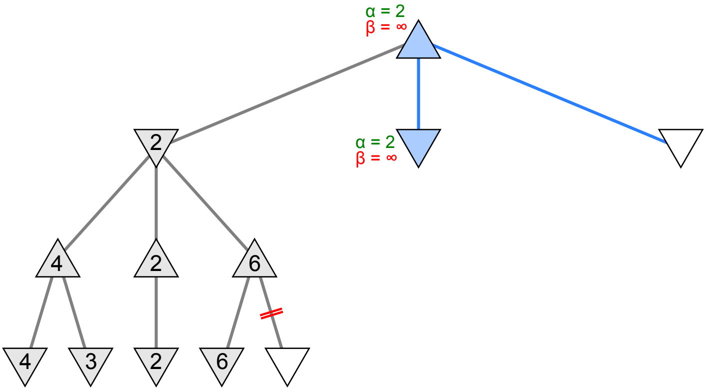
Minimax with alfa-beta prunning: Example



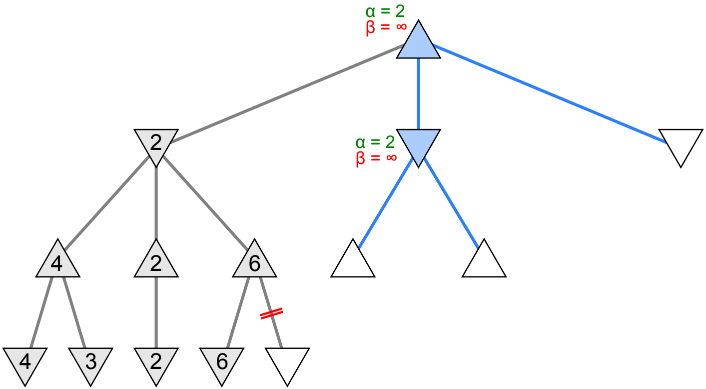
Minimax with alfa-beta prunning: Example



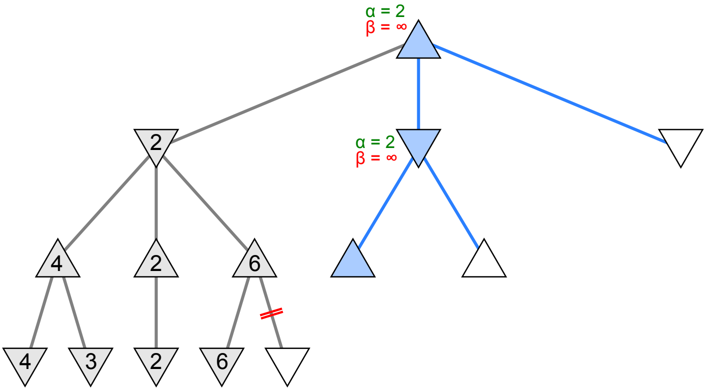
Minimax with alfa-beta prunning: Example



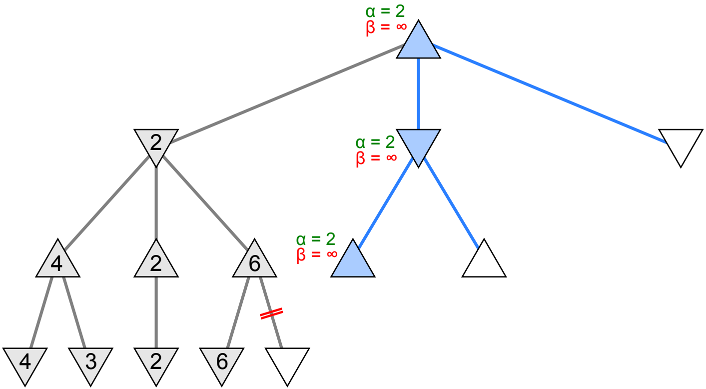
Minimax with alfa-beta prunning: Example



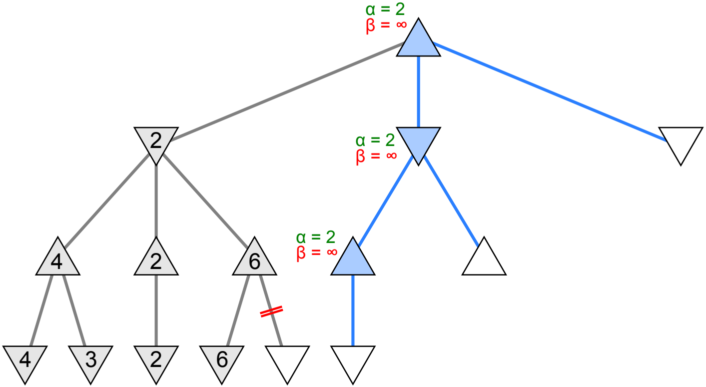
Minimax with alfa-beta prunning: Example



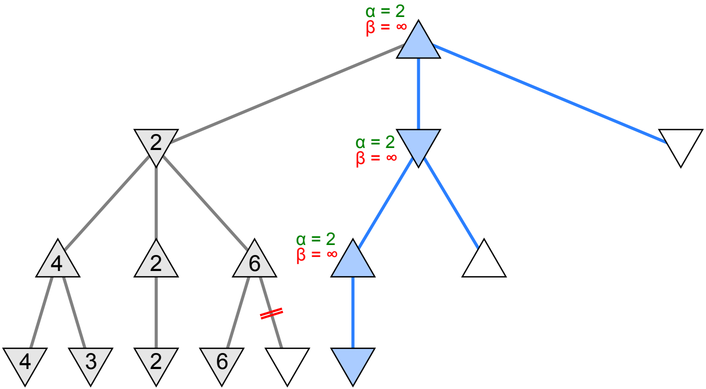
Minimax with alfa-beta prunning: Example



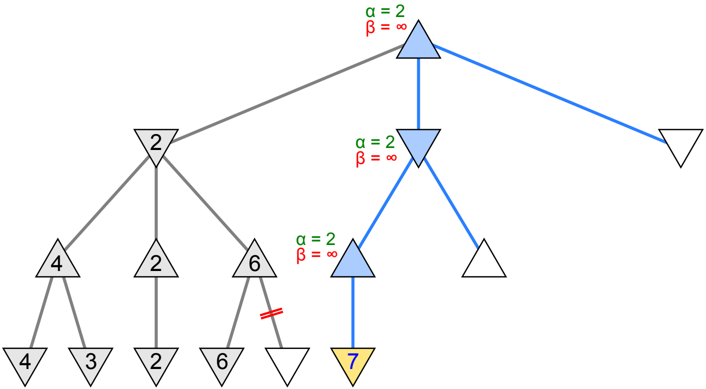
Minimax with alfa-beta prunning: Example



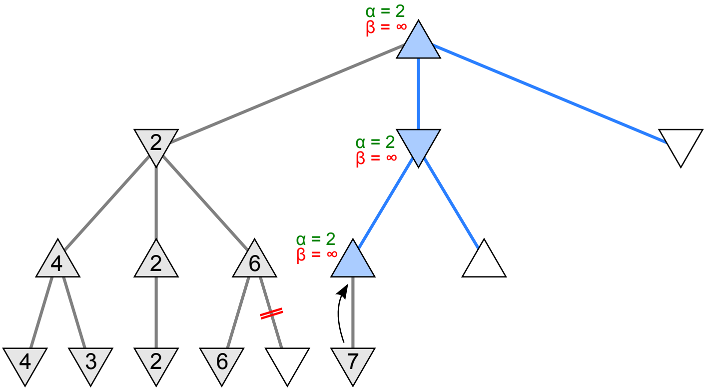
Minimax with alfa-beta prunning: Example



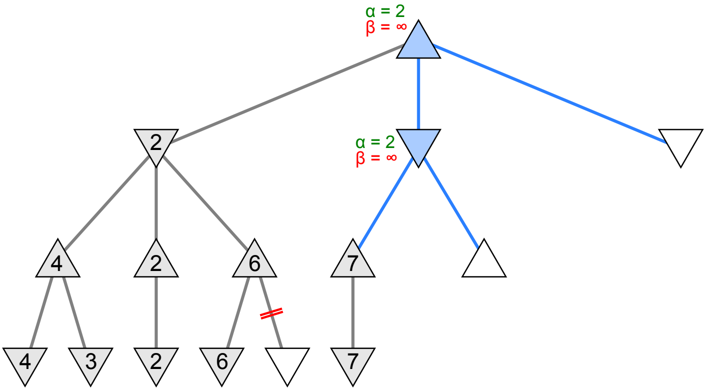
Minimax with alfa-beta prunning: Example



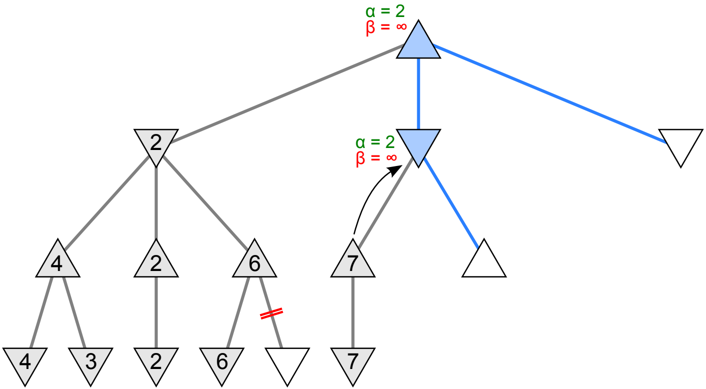
Minimax with alfa-beta prunning: Example



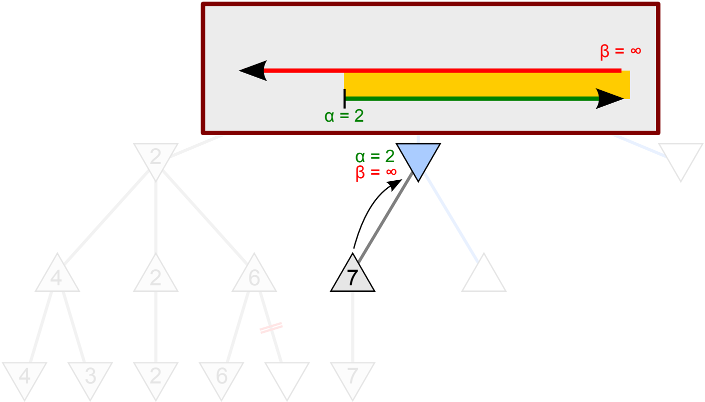
Minimax with alfa-beta prunning: Example



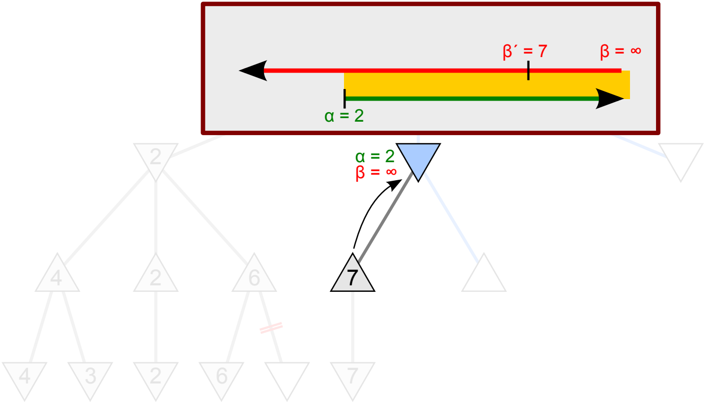
Minimax with alfa-beta prunning: Example



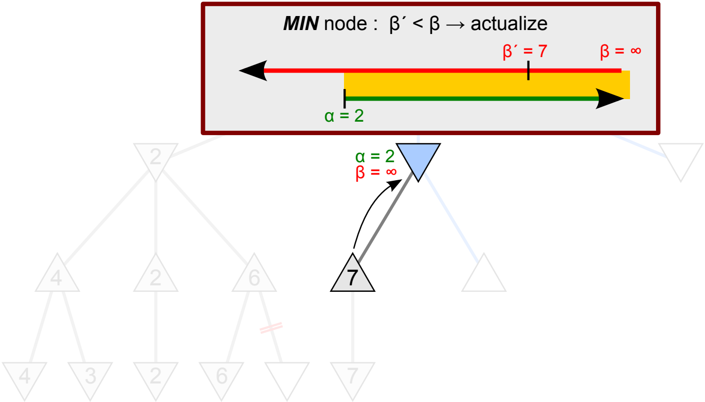
Minimax with alfa-beta prunning: Example



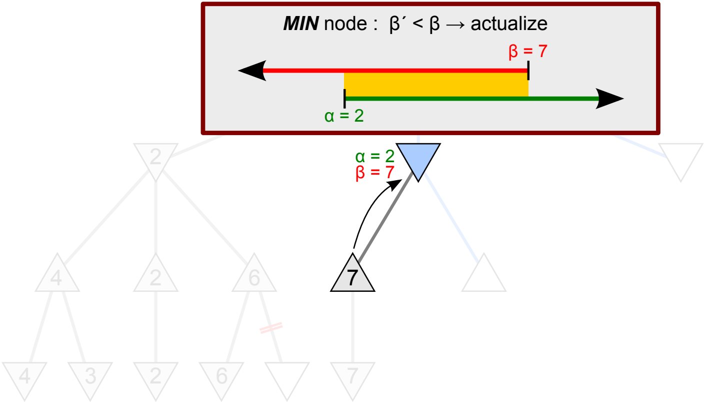
Minimax with alfa-beta prunning: Example



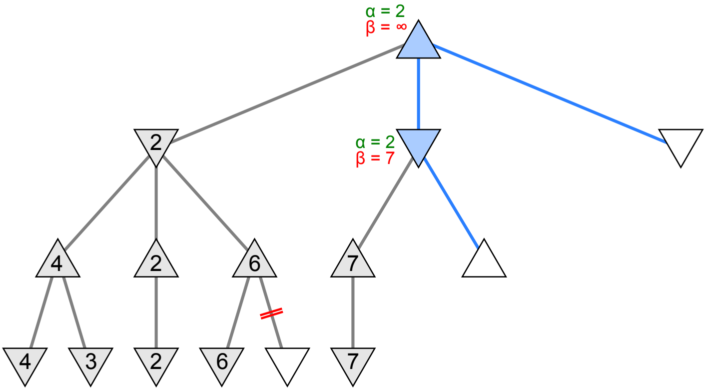
Minimax with alfa-beta prunning: Example



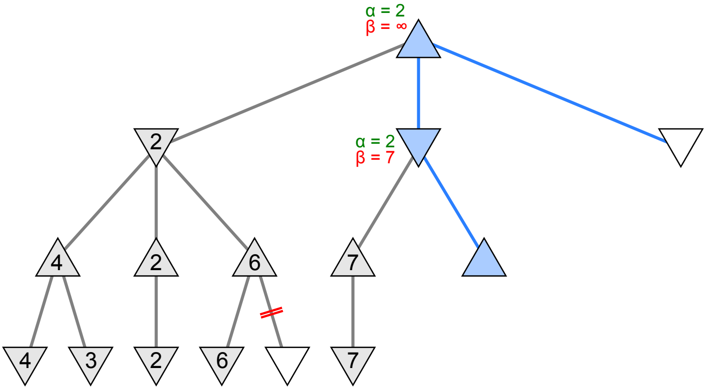
Minimax with alfa-beta prunning: Example



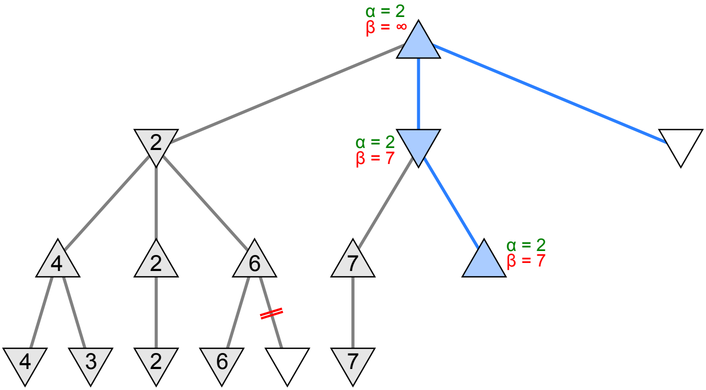
Minimax with alfa-beta prunning: Example



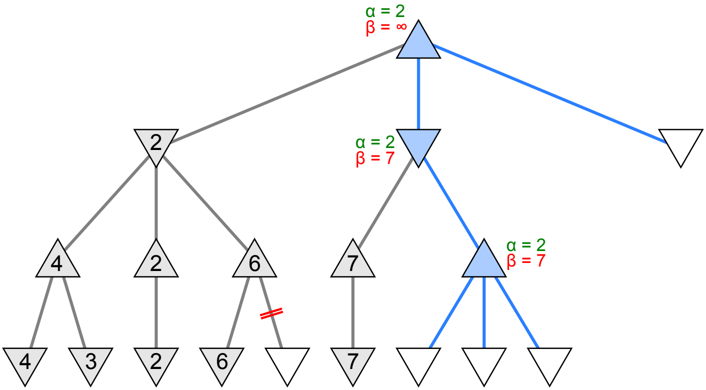
Minimax with alfa-beta prunning: Example



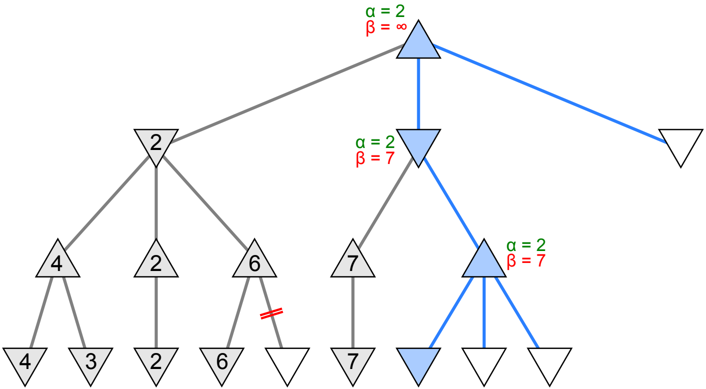
Minimax with alfa-beta prunning: Example



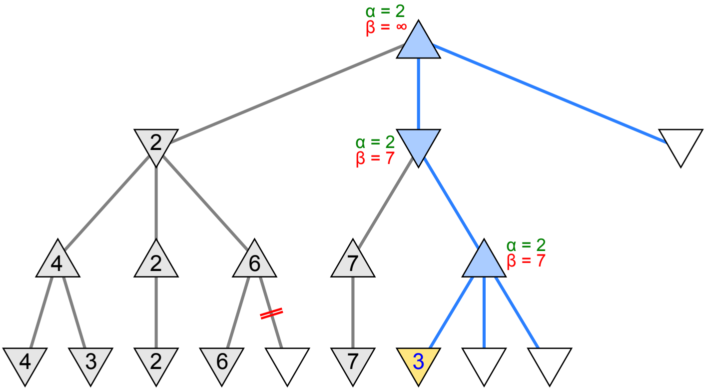
Minimax with alfa-beta prunning: Example



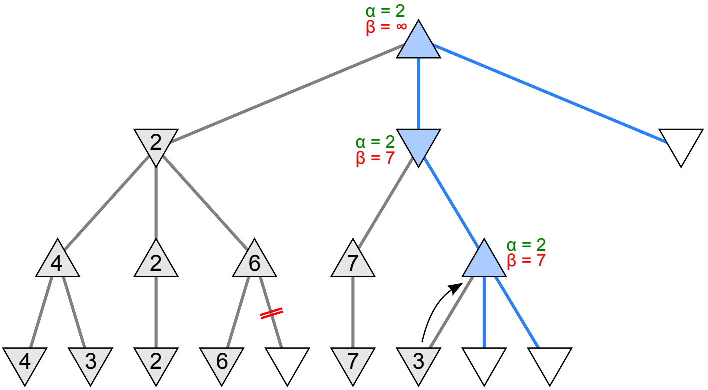
Minimax with alfa-beta prunning: Example



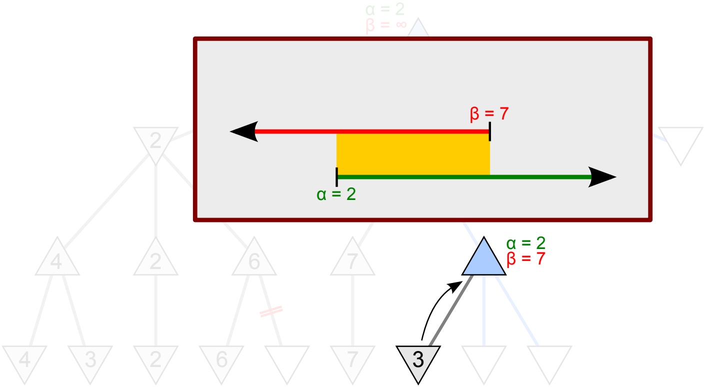
Minimax with alfa-beta prunning: Example



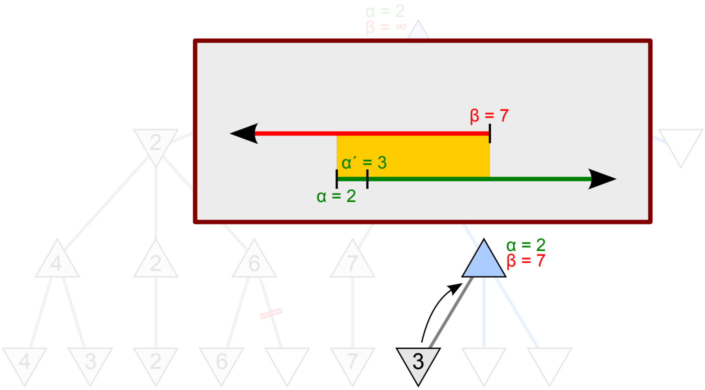
Minimax with alfa-beta prunning: Example



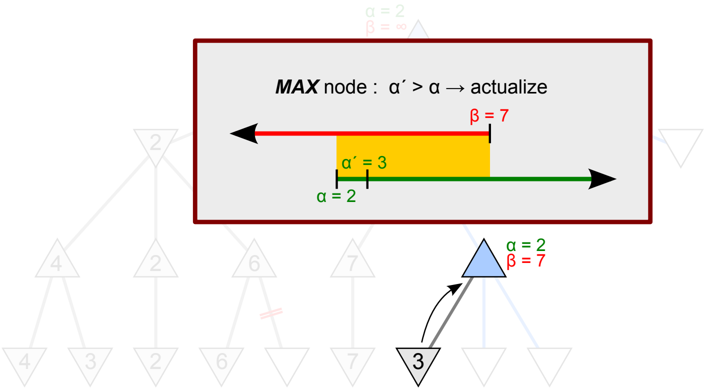
Minimax with alfa-beta prunning: Example



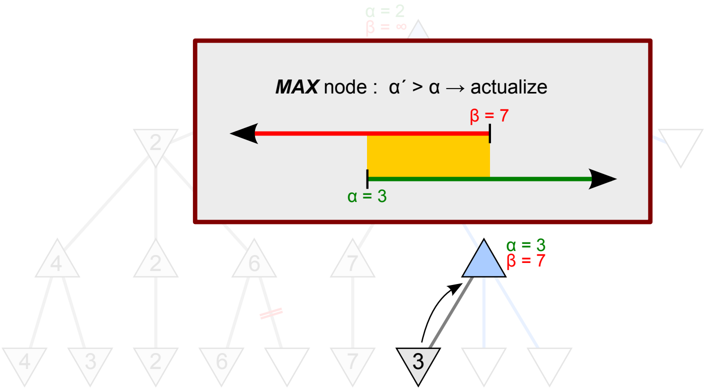
Minimax with alfa-beta prunning: Example



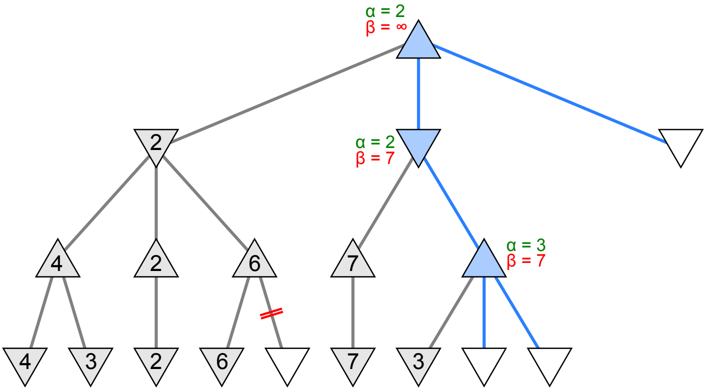
Minimax with alfa-beta prunning: Example



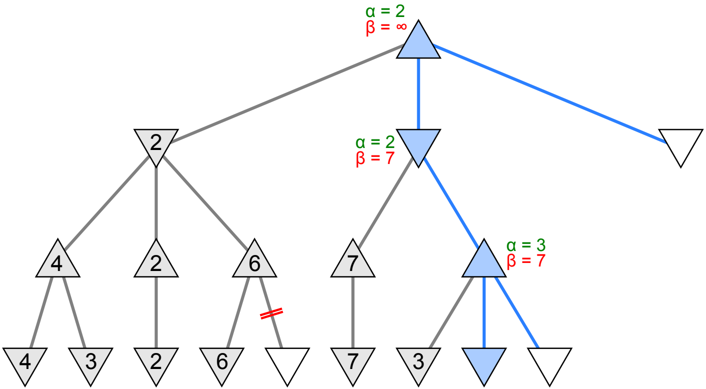
Minimax with alfa-beta prunning: Example



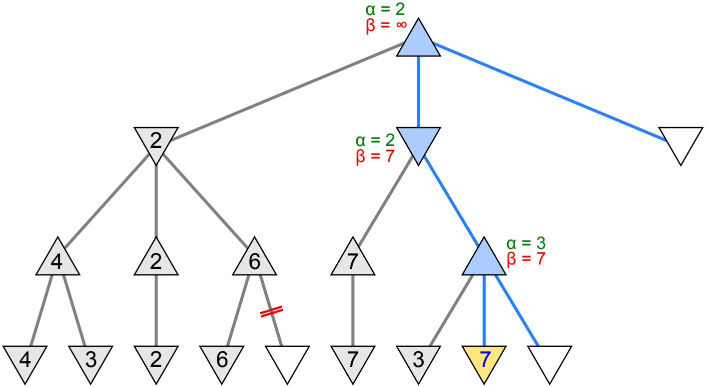
Minimax with alfa-beta prunning: Example



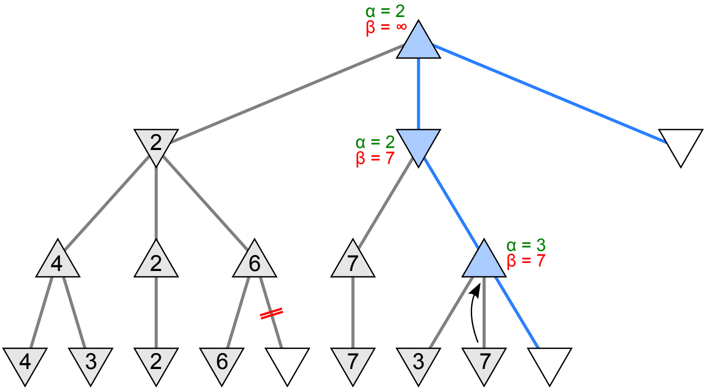
Minimax with alfa-beta prunning: Example



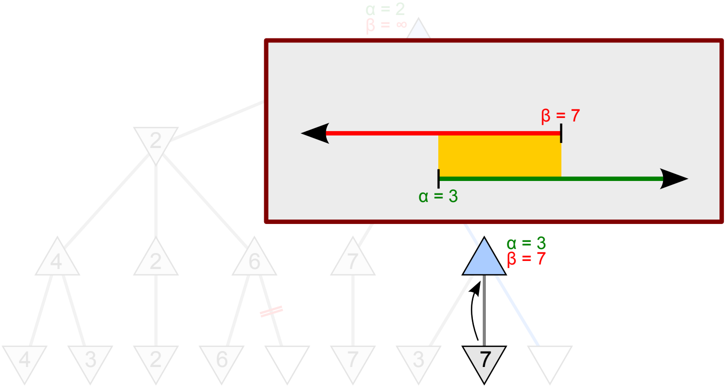
Minimax with alfa-beta prunning: Example



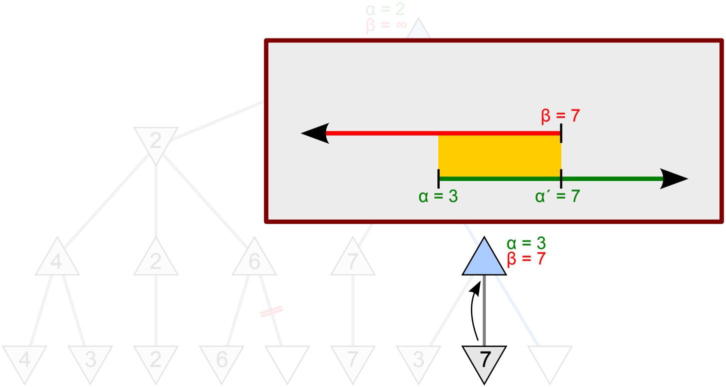
Minimax with alfa-beta prunning: Example



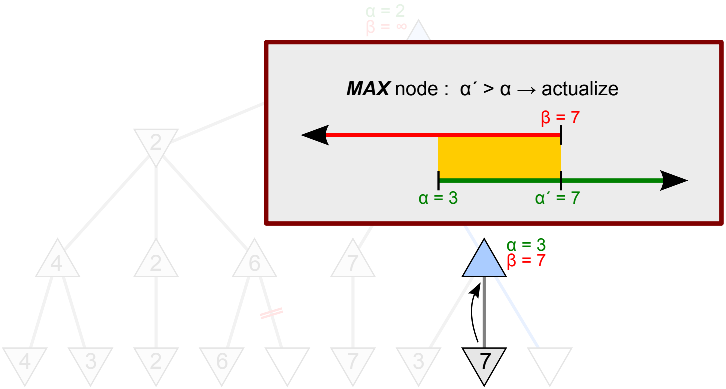
Minimax with alfa-beta prunning: Example



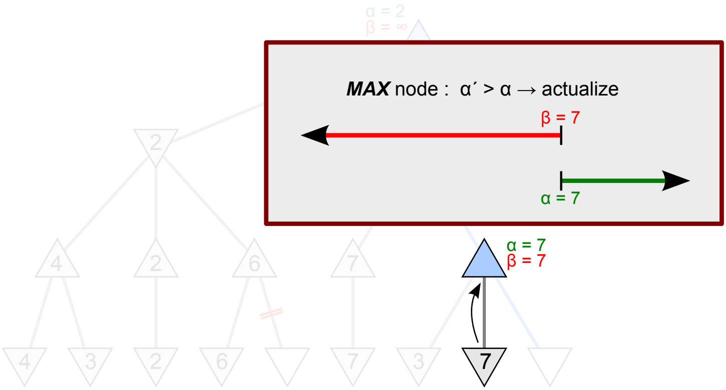
Minimax with alfa-beta prunning: Example



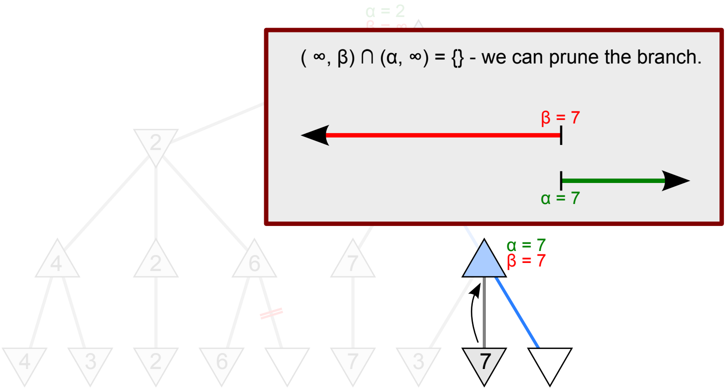
Minimax with alfa-beta prunning: Example



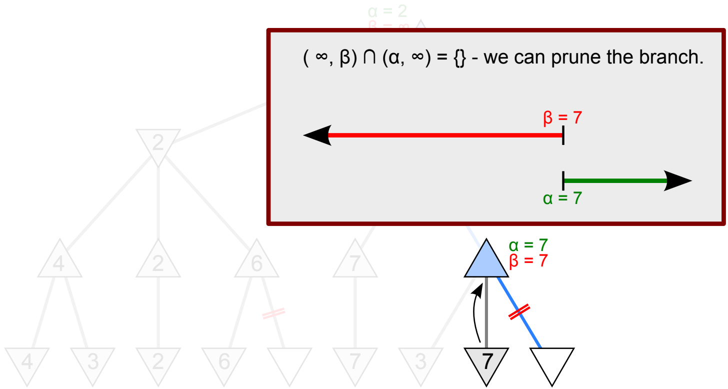
Minimax with alfa-beta prunning: Example



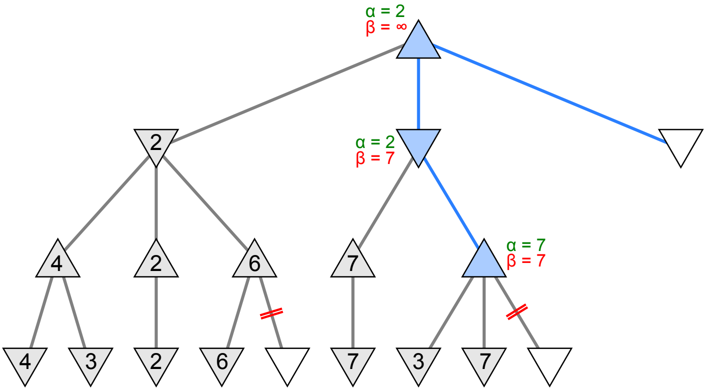
Minimax with alfa-beta prunning: Example



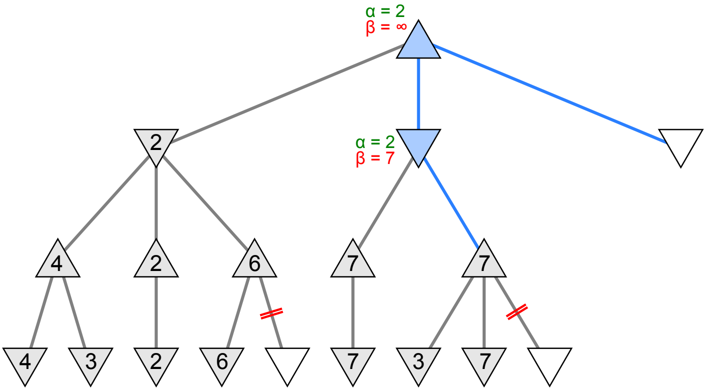
Minimax with alfa-beta prunning: Example



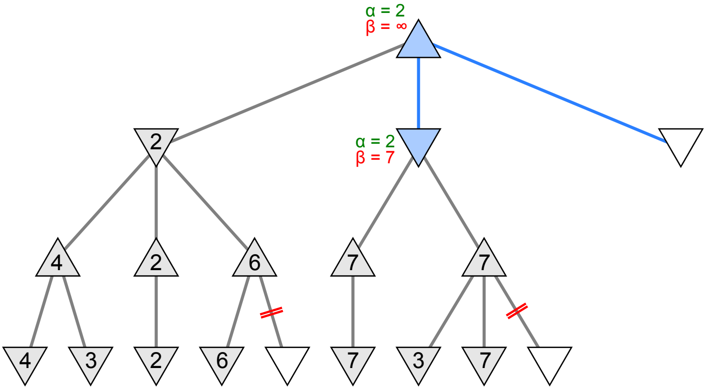
Minimax with alfa-beta prunning: Example



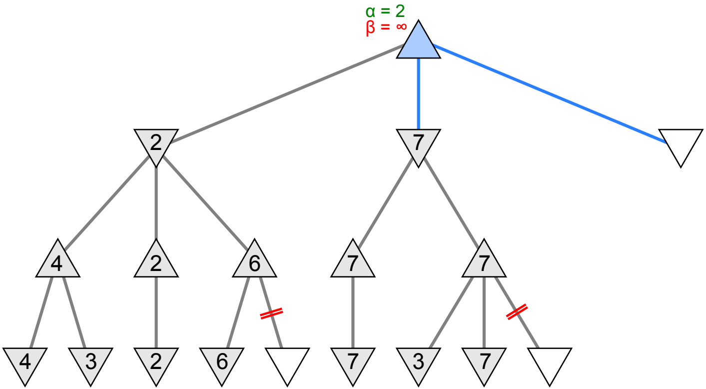
Minimax with alfa-beta prunning: Example



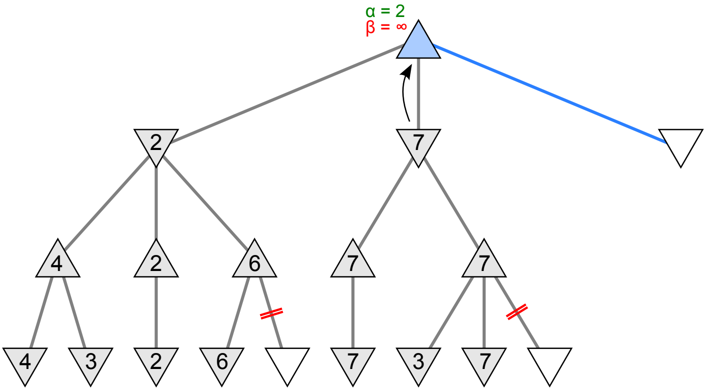
Minimax with alfa-beta prunning: Example



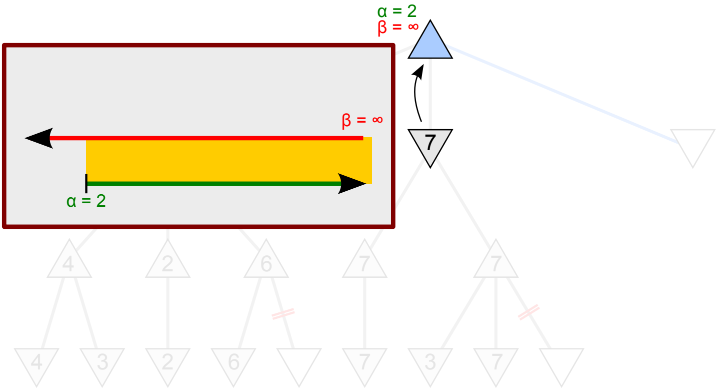
Minimax with alfa-beta prunning: Example



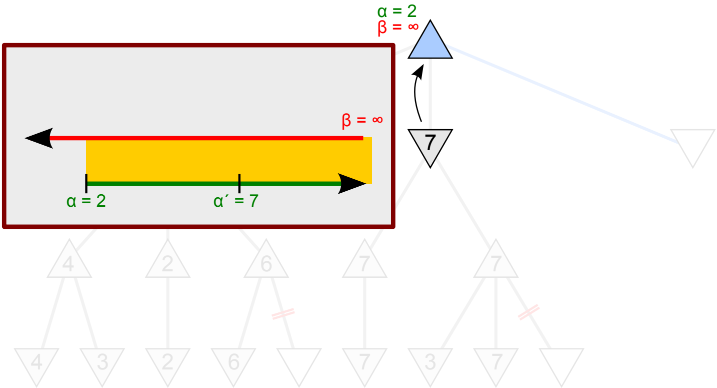
Minimax with alfa-beta prunning: Example



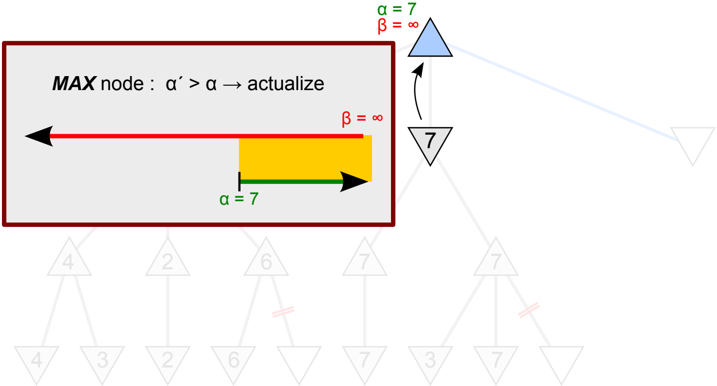
Minimax with alfa-beta prunning: Example



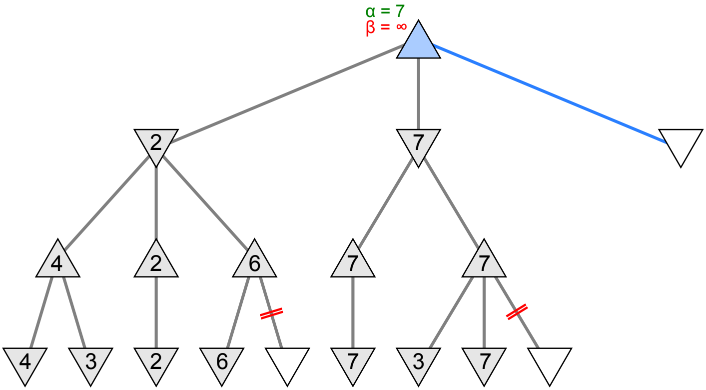
Minimax with alfa-beta prunning: Example



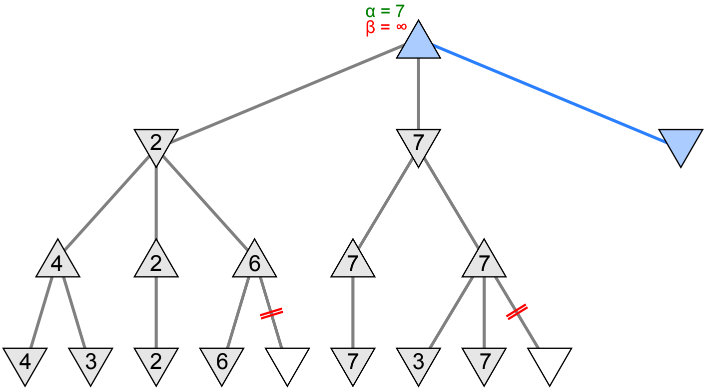
Minimax with alfa-beta prunning: Example



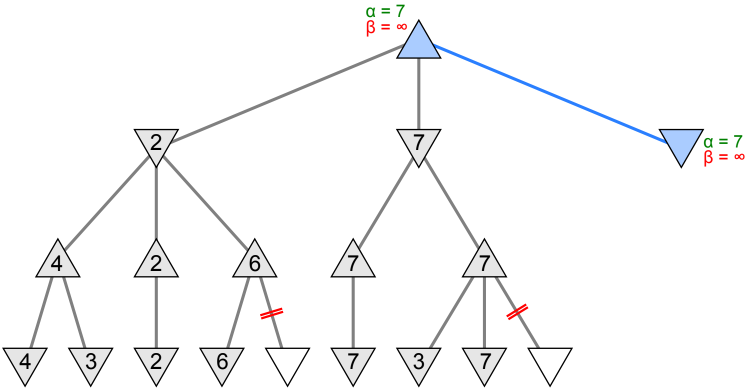
Minimax with alfa-beta prunning: Example



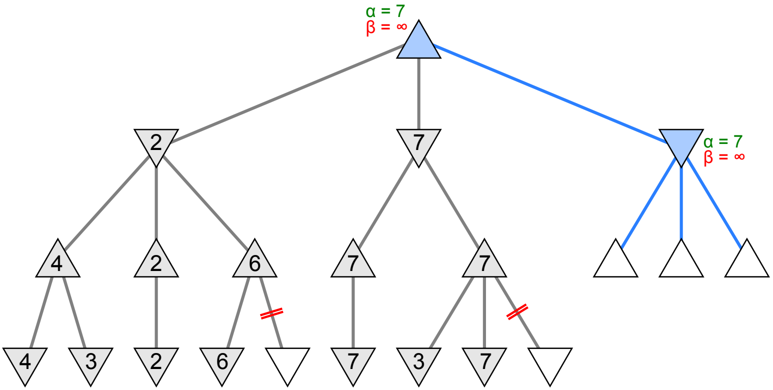
Minimax with alfa-beta prunning: Example



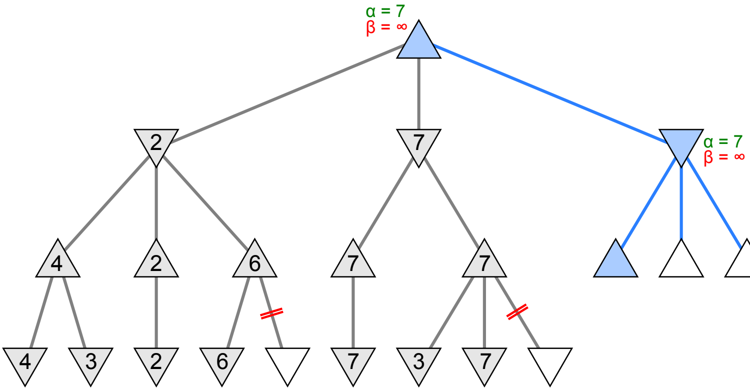
Minimax with alfa-beta prunning: Example



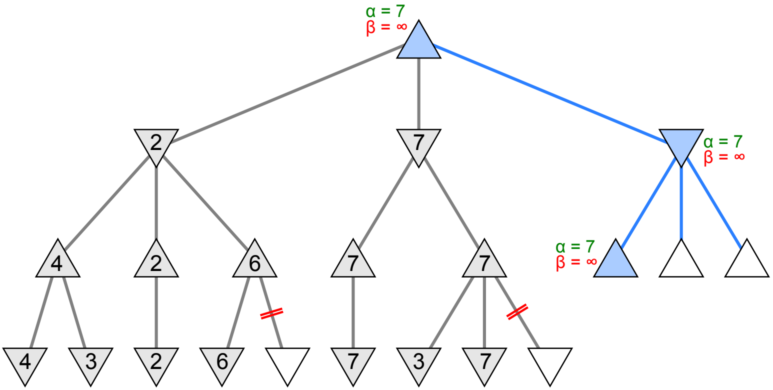
Minimax with alfa-beta prunning: Example



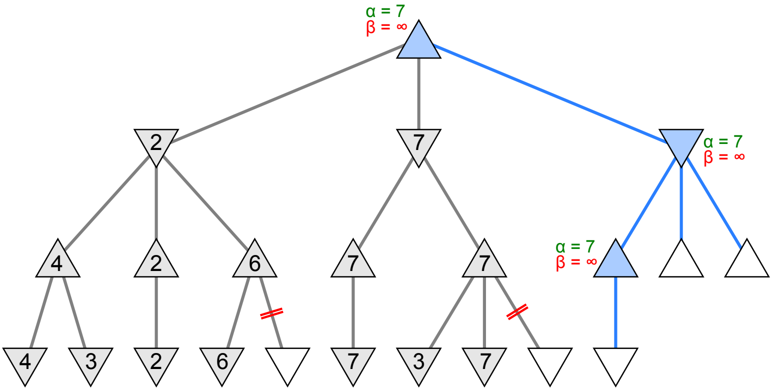
Minimax with alfa-beta prunning: Example



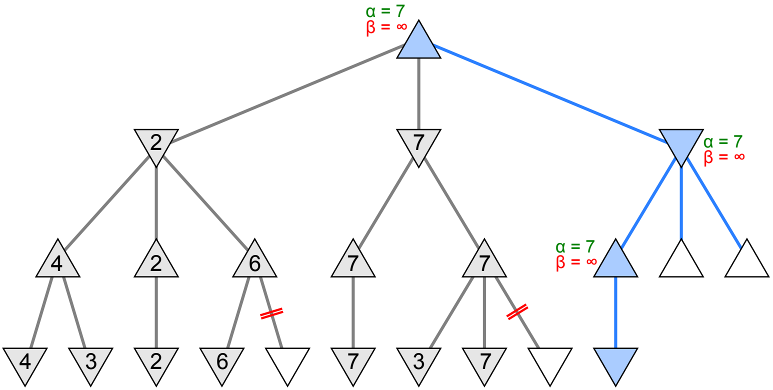
Minimax with alfa-beta prunning: Example



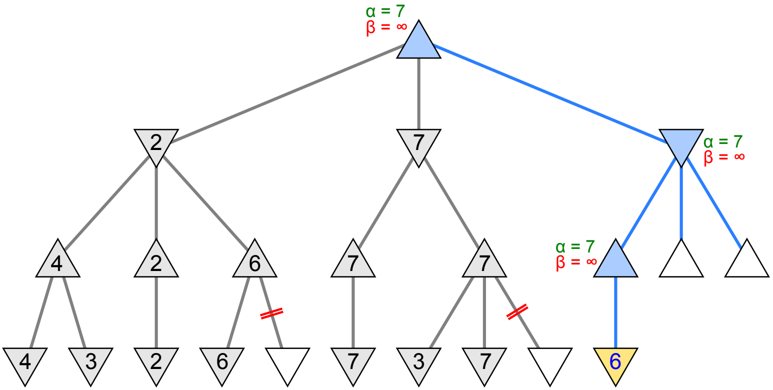
Minimax with alfa-beta prunning: Example



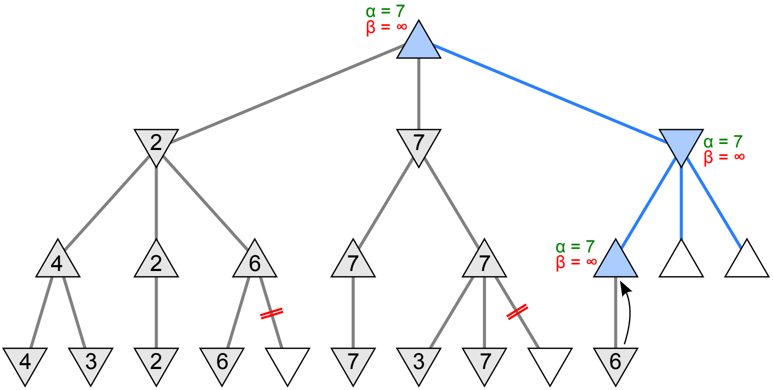
Minimax with alfa-beta prunning: Example



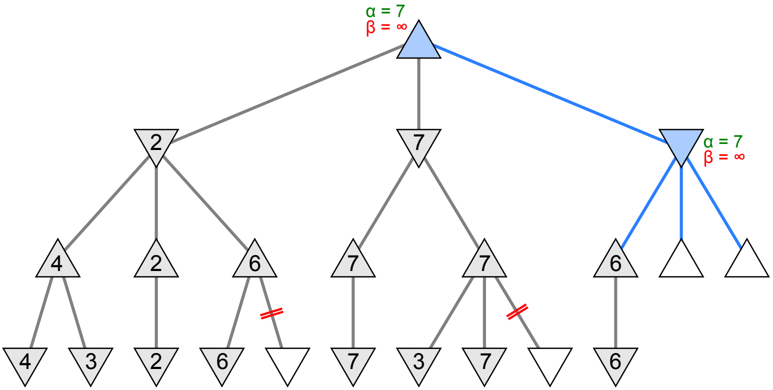
Minimax with alfa-beta prunning: Example



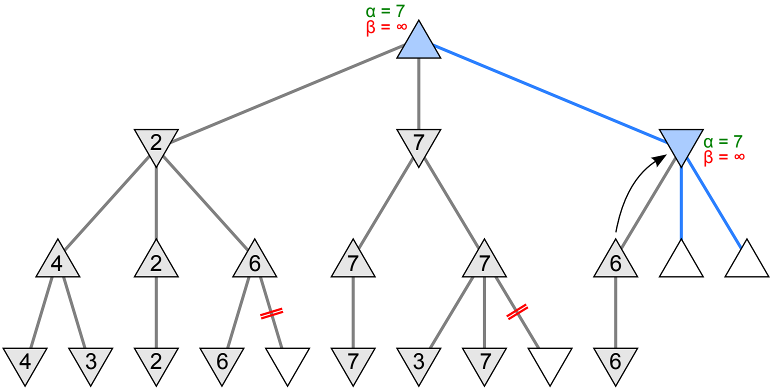
Minimax with alfa-beta prunning: Example



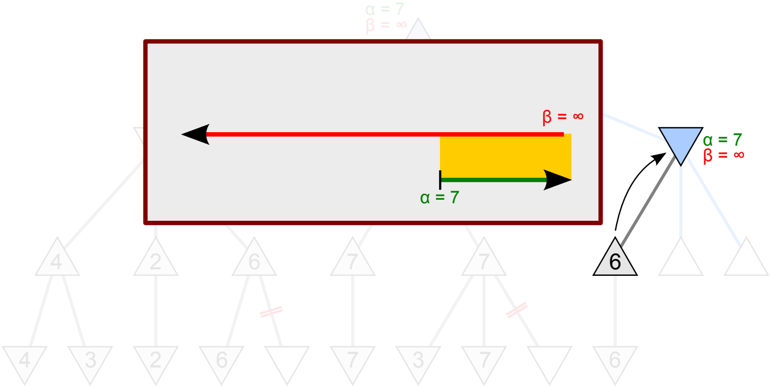
Minimax with alfa-beta prunning: Example



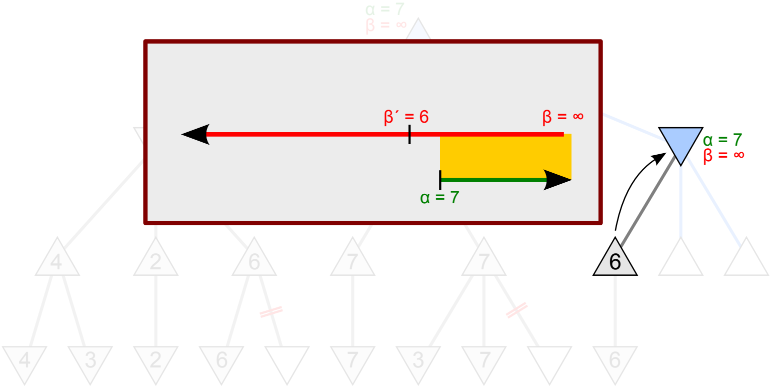
Minimax with alfa-beta prunning: Example



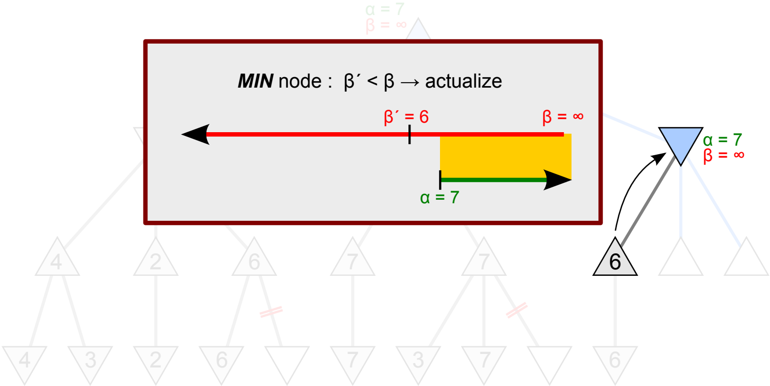
Minimax with alfa-beta prunning: Example



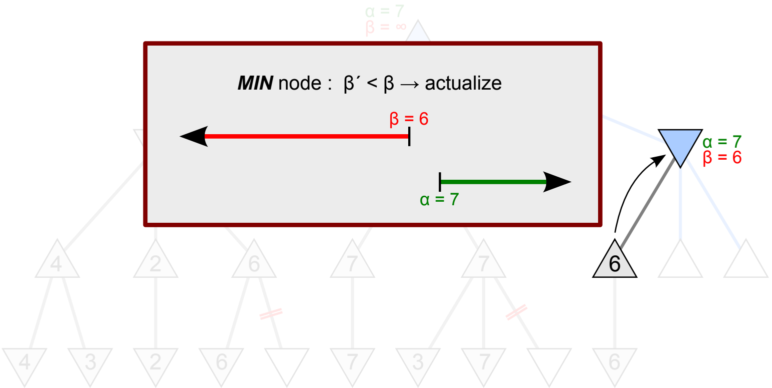
Minimax with alfa-beta prunning: Example



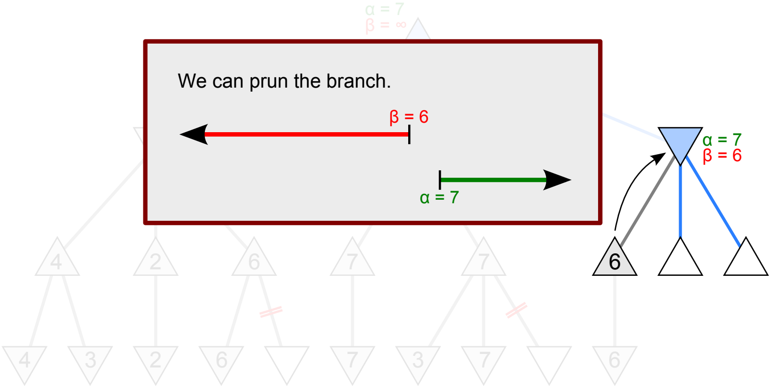
Minimax with alfa-beta prunning: Example



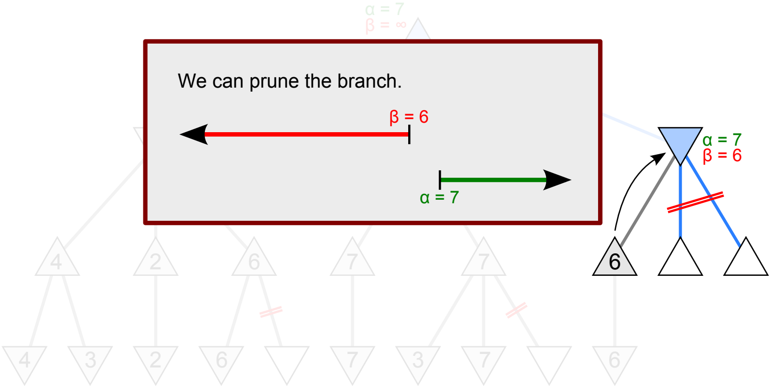
Minimax with alfa-beta prunning: Example



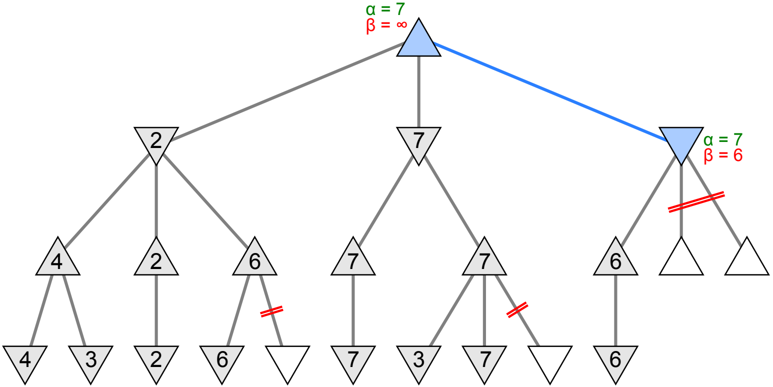
Minimax with alfa-beta prunning: Example



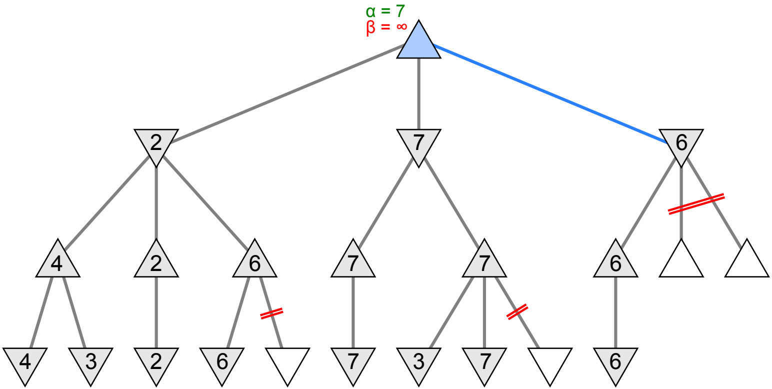
Minimax with alfa-beta prunning: Example



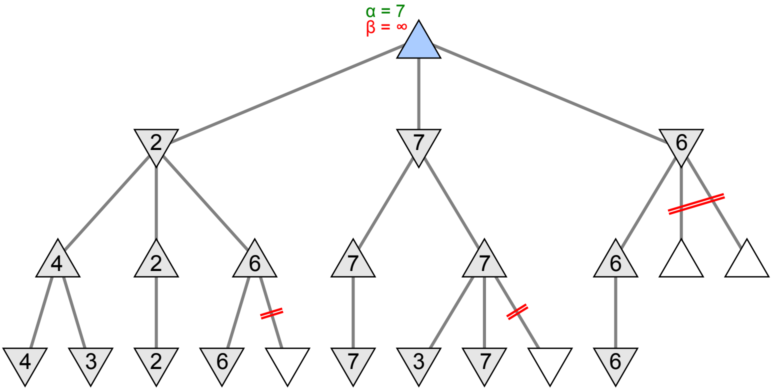
Minimax with alfa-beta prunning: Example



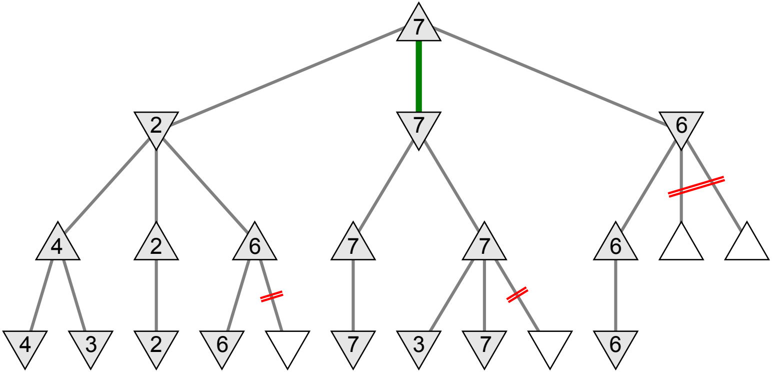
Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example



Minimax with alfa-beta prunning: Example



Evaluation Function

- Minimax with alfa-beta pruning selects optimal action in the root based on expected utility.
    - ... all depends on the quality of evaluation function!
- Evaluation function should be:
    - **fast**,
    - **accurate**.
- Horizon problem with limited depth:
    - in $d = 5$ is utility high, but in $d = 6$ might be significantly reduced (for example opponent takes queen).
    - Expansion should end in stable state (quiescence search).