Chapter S:VII

VII. Game Playing

- Game Playing Introduction
- Evaluation Functions for Game Trees
- Propagation Algorithms for Game Trees

Game Playing Introduction

The game tree search here focuses on two-player perfect-information games:

- The rules of the game define legal moves; there is no room for chance.
- There is definite initial state s.
- The players take turns drawing.
- Three different goal states are distinguished:
  1. win (W)
  2. loss (L)
  3. draw (D)
- Examples: chess, checkers, Go, but not poker or backgammon.

Game Playing Introduction

The game tree search here focuses on two-player perfect-information games:

- The rules of the game define legal moves; there is no room for chance.
- There is definite initial state s.
- The players take turns drawing.
- Three different goal states are distinguished:
  1. win (W)
  2. loss (L)
  3. draw (D)
- Examples: chess, checkers, Go, but not poker or backgammon.

A game tree is a representation of all possible plays of a game:

- The root node represents the initial state s. Leaf nodes represent goal states.
- Edges represent possible moves.
- Each path from the root s to a leaf nodes represents a complete game.

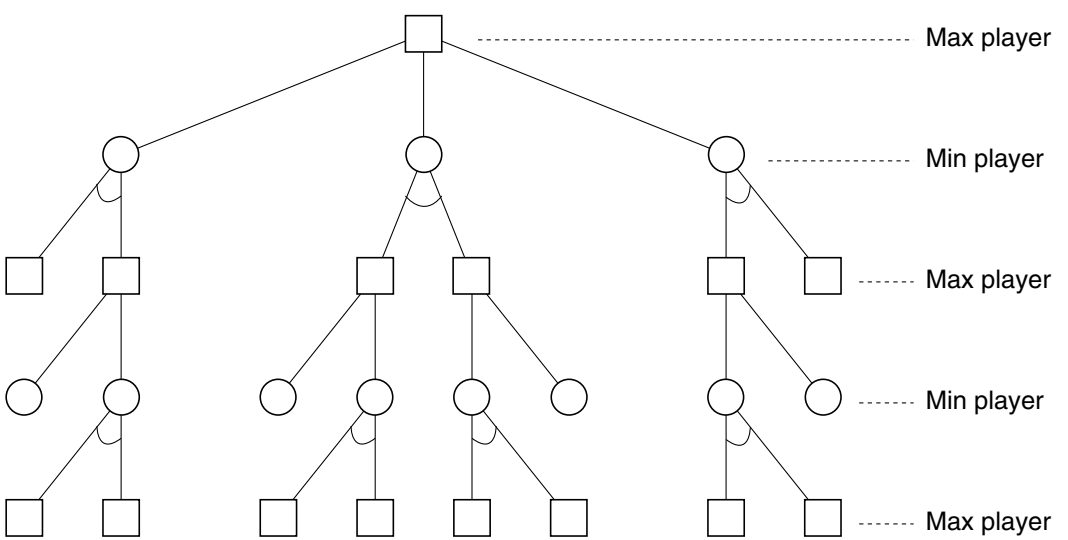
Game Playing Introduction

Definition 80 (Game Tree)

A game tree is a (finite) directed tree representing a 2-player perfect information game where nodes (both inner nodes and leaf nodes) are either of type "*Max*" or "*Min*". In particular holds:

1. All nodes at the same level of a game tree are of the same type.
2. *Max* nodes and *Min* nodes alternate between any two consecutive levels.
3. Terminal nodes represent the outcomes of the games (possible matches).

Illustration:



Remarks:

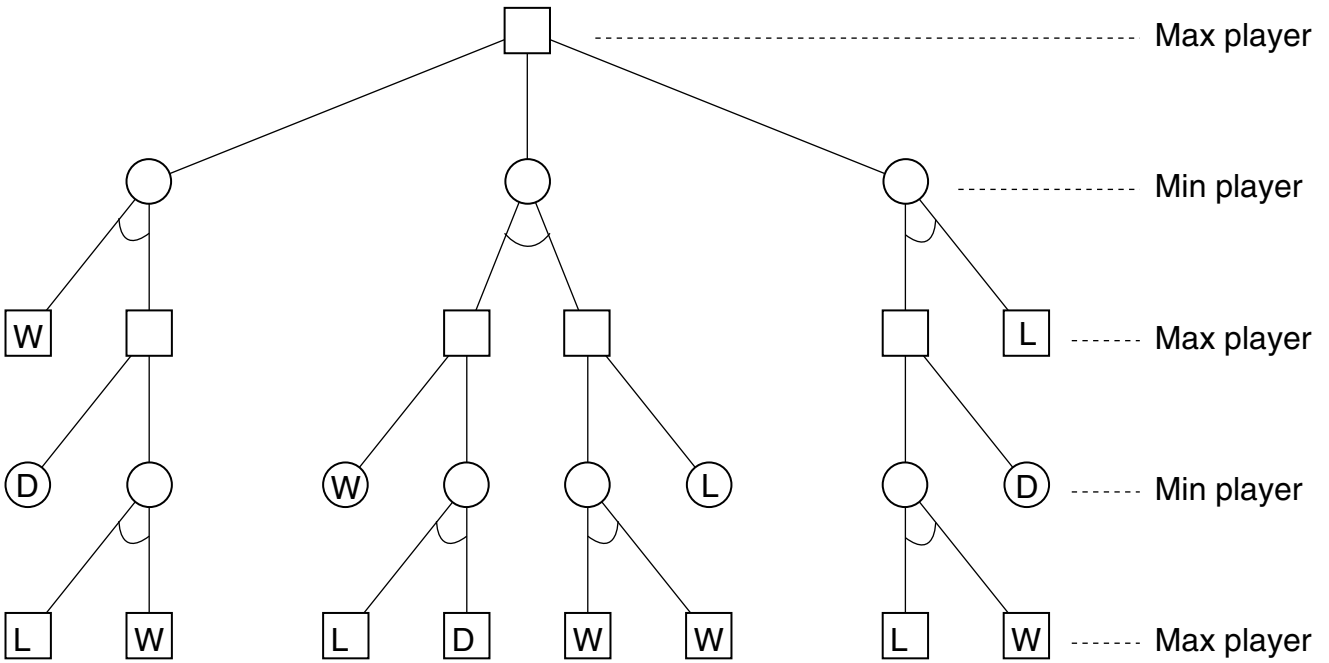
- As usual for directed trees, edge direction is form top to bottom.
- Without loss of generality we agree on the following:
  - The interpretation of game trees is always done from the perspective of the *Max* player. Hence, the nodes of the *Min* player are considered as AND nodes, and all labelings reflect *Max*'s view.
  - Player 1 is called "*Max* player" or *Max*, and player 2 is called "*Min* player" or *Min*.
  - Graphical notation and summary:

    □ = player 1 = *Max* player = own view = *Max* node = OR node

    ◯ = player 2 = *Min* player = adversarial view = *Min* node = AND node
  - Finally, we assume that both players play optimum: each player wants to win.
- Although a game (like chess) can theoretically have an infinite number of moves, every tournament has additional rules to limit the game.
  Therefore, for simplicity, we assume that a game tree is a finite tree.
- In general, the OR nodes of player 2 are the AND nodes of player 1—and vice versa.
- Game trees are AND-OR trees. Unifying nodes representing the same game state results in an AND-OR graph.
- In game theory, we often focus on payoff matrices that consider only the possible outcomes of a game and the corresponding rewards. Therefore, game trees are often called the extensive form of a game representation.

Game Playing Introduction

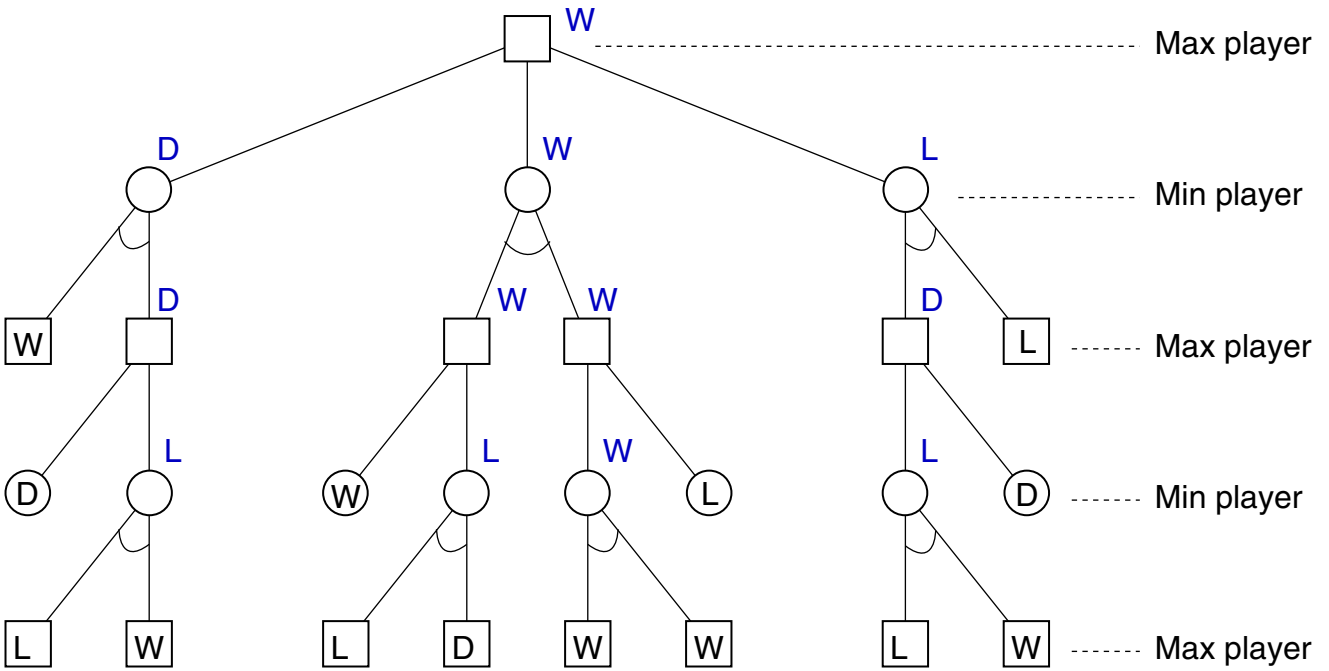
Illustration (continued):



Q: Is there a strategy such that *Max* will (start and) inevitably win?

Game Playing Introduction

Illustration (continued):



Q: Is there a strategy such that *Max* will (start and) inevitably win?

Game Playing Introduction

Definition 81 (Status Labeling Procedure)

Let T be a finite game tree whose leaf nodes are labeled with "win", "loss", and "draw" respectively. Then the label or status of an inner node $n \in T$ with *successors*(n) is defined as follows:

- If n is of type "*Max*" then

$$\textit{status}(n) = \begin{cases} \text{win} & \text{if } \exists n' \in \textit{successors}(n) : \textit{status}(n') = \text{win} \\ \text{loss} & \text{if } \forall n' \in \textit{successors}(n) : \textit{status}(n') = \text{loss} \\ \text{draw} & \text{otherwise} \end{cases}$$

- If n is of type "*Min*" then

$$\textit{status}(n) = \begin{cases} \text{win} & \text{if } \forall n' \in \textit{successors}(n) : \textit{status}(n') = \text{win} \\ \text{loss} & \text{if } \exists n' \in \textit{successors}(n) : \textit{status}(n') = \text{loss} \\ \text{draw} & \text{otherwise} \end{cases}$$

Remarks:

- In the above definition, we obviously have:
  - If n is of type "*Max*" then

$$\textit{status}(n) = \text{draw} \quad \text{if and only if} \quad \left\{ \begin{array}{l} \forall n' \in \textit{successors}(n) : \textit{status}(n') \neq \text{win and} \\ \exists n' \in \textit{successors}(n) : \textit{status}(n') = \text{draw} \end{array} \right\}$$

  - If n is of type "*Min*" then

$$\textit{status}(n) = \text{draw} \quad \text{if and only if} \quad \left\{ \begin{array}{l} \forall n' \in \textit{successors}(n) : \textit{status}(n') \neq \text{loss and} \\ \exists n' \in \textit{successors}(n) : \textit{status}(n') = \text{draw} \end{array} \right\}$$

- Obviously node labeling in game trees happens bottom-up, which usually implies that the game tree must be completely known.
- Compare the Definition "Status Labeling Procedure" to the Definition "Solved-Labeling Procedure", which defines whether or not an AND-OR graph contains a solution graph. [S:II Basic Search Algorithms])

Game Playing Introduction

Definition 82 (Game Strategy, Solution Tree, Winning Strategy)

Let T be a game tree whose leaf nodes are labeled with “win”, “loss”, and “draw” respectively.

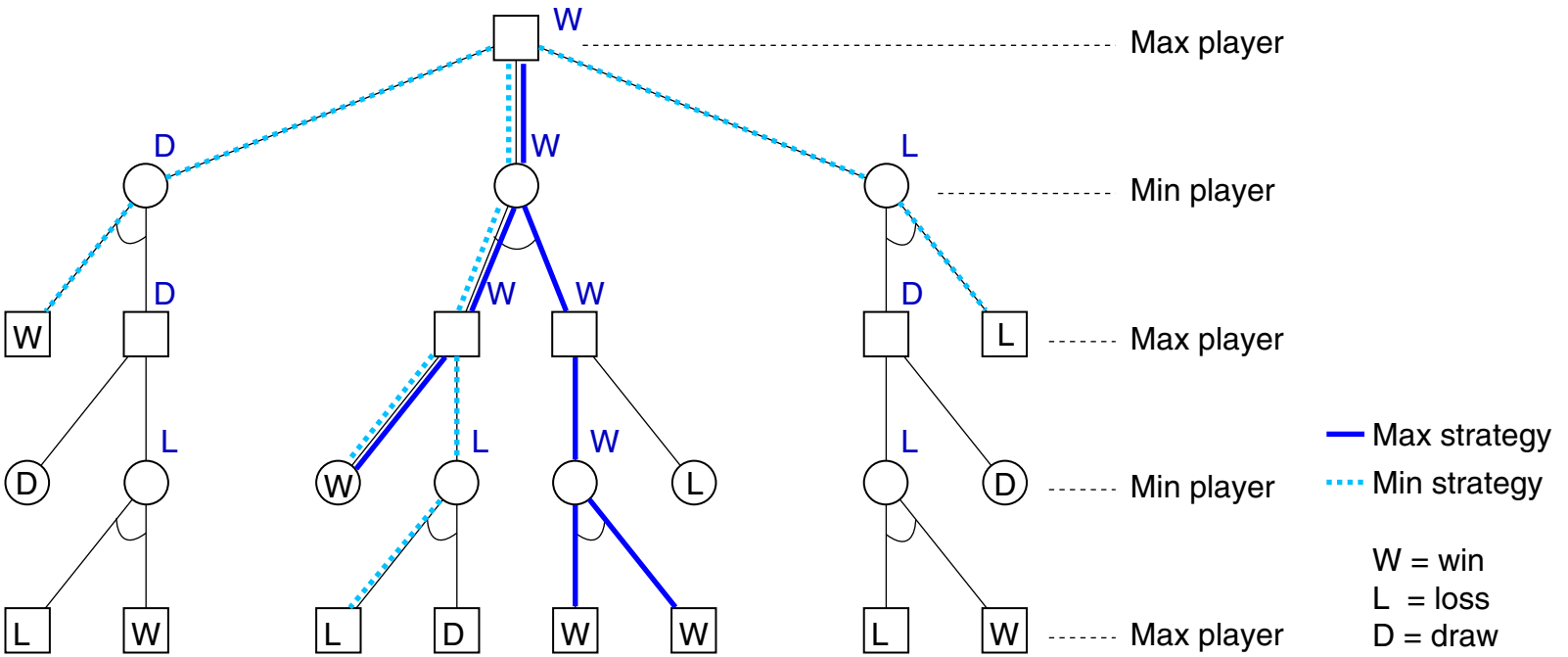
A game strategy for player *Max* is a subtree T^+ of T. T^+ has the root s, it contains for each inner *Max* node exactly one successor and for each inner *Min* node all successors.

A game strategy for player *Min* is a subtree T^- of T. T^- has the root s, it contains for each inner *Min* node exactly one successor and for each inner *Max* node all successors.

A winning strategy (for player *Max*) specifies how s is labeled with “win”, irrespective of the moves of player *Min*.

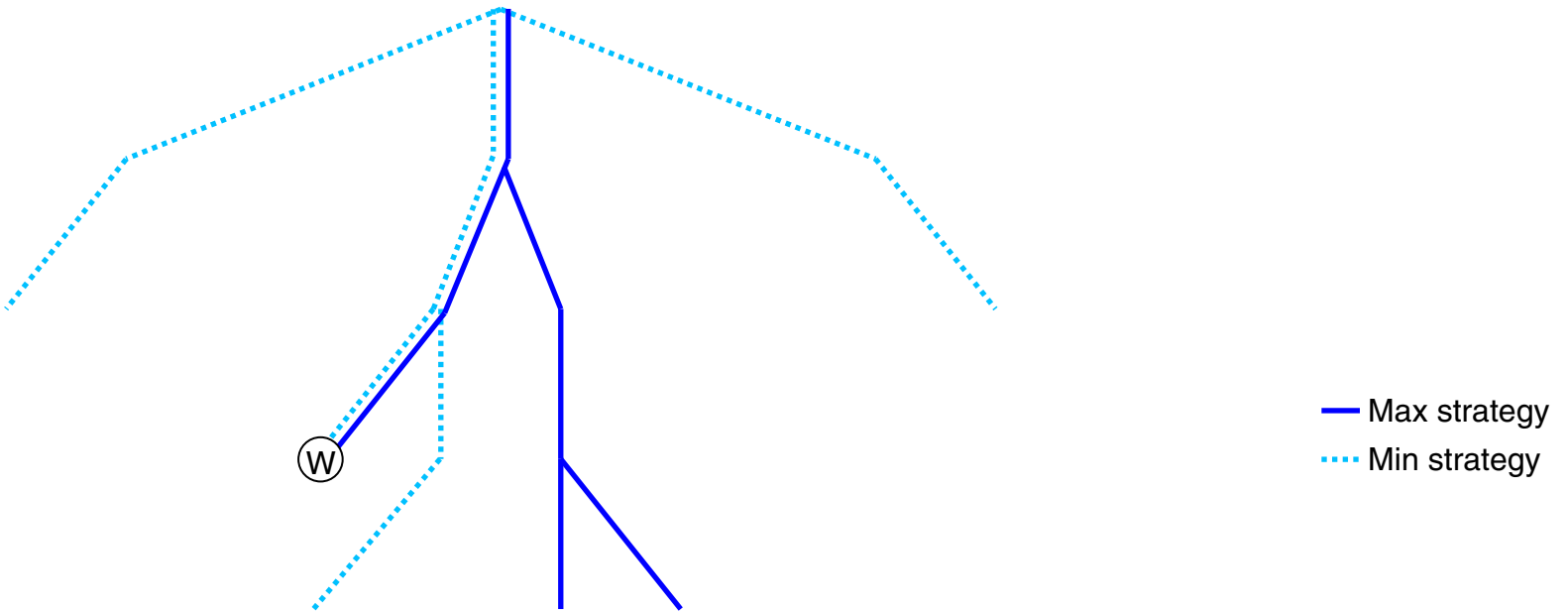
Game Playing Introduction

Illustration (continued):



Game Playing Introduction

Two strategies T^+, T^-, of *Max* and *Min* respectively, share at each level either one or no edge:



- The intersection of two strategies T^+, T^-, defines a path that corresponds to the play of the players *Max* and *Min* if both stick to their strategy.
- The intersection of two strategies T^+, T^-, defines the leaf $(T^+ \sqcap T^-)$, which corresponds to the final position of the game.

Game Playing Introduction

Strategy considerations for two-player, perfect-information games:

1. Let player *Max* choose a strategy T^+ and disclose it to player *Min*.
2. *Min* chooses T^- such that a leaf is reached that is as unfavorable as possible for *Max*. The label of this leaf computes as follows:

$$\min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

3. (With foresight) *Max* chooses T^+ such that the most unfavorable leaf is as good as possible for him. The label of this leaf computes as follows:

$$\max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

Game Playing Introduction

Strategy considerations for two-player, perfect-information games:

1. Let player *Max* choose a strategy T^+ and disclose it to player *Min*.
2. *Min* chooses T^- such that a leaf is reached that is as unfavorable as possible for *Max*. The label of this leaf computes as follows:

$$\min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

3. (With foresight) *Max* chooses T^+ such that the most unfavorable leaf is as good as possible for him. The label of this leaf computes as follows:

$$\max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

Game Playing Introduction

Strategy considerations for two-player, perfect-information games:

1. Let player *Max* choose a strategy T^+ and disclose it to player *Min*.
2. *Min* chooses T^- such that a leaf is reached that is as unfavorable as possible for *Max*. The label of this leaf computes as follows:

$$\min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

3. (With foresight) *Max* chooses T^+ such that the most unfavorable leaf is as good as possible for him. The label of this leaf computes as follows:

$$\max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

Game Playing Introduction

Strategy considerations for two-player, perfect-information games:

1. Let player *Max* choose a strategy T^+ and disclose it to player *Min*.

2. *Min* chooses T^- such that a leaf is reached that is as unfavorable as possible for *Max*. The label of this leaf computes as follows:

$$\min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

3. (With foresight) *Max* chooses T^+ such that the most unfavorable leaf is as good as possible for him. The label of this leaf computes as follows:

$$\max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

Game Playing Introduction

Strategy considerations for two-player, perfect-information games:

1. Let player *Max* choose a strategy T^+ and disclose it to player *Min*.
2. *Min* chooses T^- such that a leaf is reached that is as unfavorable as possible for *Max*. The label of this leaf computes as follows:

$$\min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

3. (With foresight) *Max* chooses T^+ such that the most unfavorable leaf is as good as possible for him. The label of this leaf computes as follows:

$$\max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

By changing the roles we obtain: $\min_{T^-} \max_{T^+} \mathit{status}(T^+ \sqcap T^-)$

It holds (here without proof):

$$\min_{T^-} \max_{T^+} \mathit{status}(T^+ \sqcap T^-) = \mathit{status}(s) = \max_{T^+} \min_{T^-} \mathit{status}(T^+ \sqcap T^-)$$

Remarks:

- The strategy considerations formalize the fact that both players play optimum.
- The following ranking between status values is supposed:
  “loss” $\prec$ “draw” $\prec$ “win” for *Max*
  “win” $\prec$ “draw” $\prec$ “loss” for *Min*
  Since we are using the view of player *Max* and, therefore, his ranking, player *Min* tries to achieve a minimum result.
- The shown connections regarding *status*$(T^+ \sqcap T^-)$ can be proven inductively, considering a generic game tree, and starting with the leaf nodes.
- Observation:
  It is irrelevant whether a strategy is chosen up-front and disclosed, or whether decisions are made during the play.
- Other consequences:
  In order to proof whether a root node can be labeled with “win” and “loss” respectively, only one strategy T^+ or T^- is required.
  However, in order to proof whether a root node can be labeled with “draw”, two strategies T^+ and T^- are required.

Game Playing Introduction

The labeling of game trees is possible without distinguishing between *Max* and *Min* players. Instead, each node can be labeled from the viewpoint of that player who is currently moving: mover-oriented status labeling.

Definition 83 (*mstatus* Labeling Procedure)

Let T be a game tree whose leaf nodes are labeled with “win”, “loss”, and “draw” respectively, from the perspective of the player whose turn it is. Then, under the mover-oriented viewpoint, the label or *mstatus*(n) of an inner node $n \in T$ with *successors*(n) is defined as follows.

$$\mathit{mstatus}(n) = \begin{cases} \text{win} & \text{if } \exists n' \in \mathit{successors}(n) : \mathit{mstatus}(n') = \text{loss} \\ \text{loss} & \text{if } \forall n' \in \mathit{successors}(n) : \mathit{mstatus}(n') = \text{win} \\ \text{draw} & \text{otherwise} \end{cases}$$

Remarks:

- By encoding the status “win” as 1, “loss” as -1, and “draw” as 0, the definition of *mstatus*(n) can be reformulated as follows:

$$\textit{mstatus}(n) = \max_{n' \in \textit{successors}(n)} \{-\textit{mstatus}(n')\}$$

  Therefore, the mover-oriented labeling is also called Neg-Max labeling.

Evaluation Functions for Game Trees

Status labeling requires the generation of nearly the complete game tree. The following order of magnitudes illustrate the infeasibility of this prerequisite.

- Checkers:
  A complete game tree contains about 10^{40} inner nodes. If we processed 3 billion nodes per second, tree generation would last about 10^{21} centuries.
  [Samuel 1959]
- Chess:
  A complete game tree contains about 10^{120} inner nodes, generated within about 10^{101} centuries.

  Even if time were not the problem, storage space would be:
  There are about 10^{80} atoms in the observable universe.

Evaluation Functions for Game Trees

Status labeling requires the generation of nearly the complete game tree. The following order of magnitudes illustrate the infeasibility of this prerequisite.

- Checkers:
  A complete game tree contains about 10^{40} inner nodes. If we processed 3 billion nodes per second, tree generation would last about 10^{21} centuries.
  [Samuel 1959]
- Chess:
  A complete game tree contains about 10^{120} inner nodes, generated within about 10^{101} centuries.

  Even if time were not the problem, storage space would be:
  There are about 10^{80} atoms in the observable universe.

➜ We need heuristics to evaluate game positions.

➜ Evaluation is based on features that characterize game positions.

➜ Distinguish an immediate and a look-ahead evaluation of game positions.

Remarks:

- Immediate (also called “static”) evaluations of game positions are usually not used for a decision. Instead, for a certain search horizon (= search depth, bounded look-ahead) the possible game position are generated, and then the nodes at the search horizon are evaluated.
- The evaluations at the search horizon are considered as “true” values (recall: *face-value principle*). The values are propagated back up to that point where a decision is to be met, i.e., the starting point of the search.

  > *“The backed-up evaluations give a more accurate estimate of the true values of Max’s possible moves than would be obtained by applying the static evaluation function directly to those moves and not looking ahead to their consequences.”*
  >
  > [Barr/Feigenbaum 1981]

Evaluation Functions for Game Trees

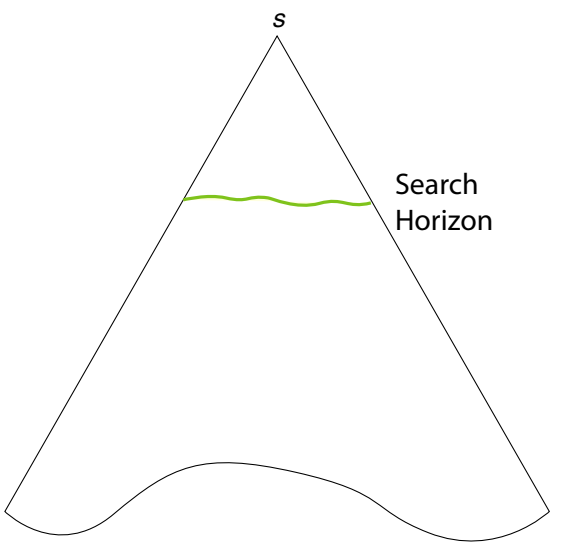
Handling Huge or Even Infinite Game Trees

- An evaluation function $e(n)$ analyzes the game situation in a node n of a game tree T, e.g.,

  $$e : V(T) \rightarrow [0, 1]$$

  Values near 1 represent good situation for the *Max*-player, values near 0 represent good situation for the *Min*-player.

- For simplicity, we consider the evaluation of a search tree as a separate process using a depth bound to limit the look-ahead.
- The game tree is processed in a depth-first framework.
- A leaf node or a node at the search horizon is evaluated by $e(n)$.
- Evaluations are propagated back to s and aggregated.



Evaluation Functions for Game Trees

Definition 84 (Minimax Rule)

Let T be a game tree and let $k, k \geq 0$, be a depth bound. The nodes of T in depth k and also the leafs of T in at most depth k form the current search horizon and can be evaluated with an evaluation function e. Then, the value $v(n)$ of a non-leaf node $n \in T$ in at most depth k is defined as follows.

$$v(n) = \begin{cases} e(n) & n \text{ is leaf node or} \\ & n \text{ is a node at depth } k \\ \max\limits_{n' \in \textit{successors}(n)} v(n') & n \text{ is of type “}\textit{Max}\text{”} \\ \min\limits_{n' \in \textit{successors}(n)} v(n') & n \text{ is of type “}\textit{Min}\text{”} \end{cases}$$

Remarks:

- The minimax rule is a natural extension of the status labeling procedure for partially explored game trees.
- Most game playing algorithms are based on variants of the minimax rule.
- When operationalizing (= implementing) the minimax rule, the available computing resources are spent for two aspects:
  1. Generation of a portion of the game tree.
  2. Evaluation of the game positions at the leafs of the generated game tree portion.

  Consider the tradeoff between the quality (complexity) of an evaluation function e and the attainable search depth (amount of search effort) for look-ahead: Where to invest the available computing resources?
- If a finite portion of the game tree was already created and evaluation is only required for this portion, we can omit the depth bound in the above definition and all the following algorithms.
- For a fixed evaluation function e the search efforts is proportional to the number of generated leaf nodes. Hence, in game theory the leaf node number is the standard measure to asses the complexity of game playing algorithms.

Propagation Algorithms for Game Trees [SOLVE, ALPHA-BETA]

Algorithm: MINIMAX-DFS

Input: n. A node in a game tree T.
successors(n). Returns the successors of node n.
k. Depth bound for evaluation.
$e(n)$. Evaluation function (real-valued) for a leaf node $n \in T$.

Output: The value $v(n)$ of the node n.

MINIMAX-DFS$(n, \textit{successors}, k, e)$

```
1.  IF ( successors(n) = ∅ OR depth(n) = k )
    THEN RETURN(e(n));
2.  FOREACH n' IN successors(n) DO
      v(n') = MINIMAX-DFS(n', successors, e);
    ENDDO
3.  IF nodeType(n) = "Max"
    THEN RETURN(max{v(n') | n' ∈ successors(n)})
    ELSE RETURN(min{v(n') | n' ∈ successors(n)})
```

Remarks:

- A backtracking variant of algorithm MINIMAX-DFS, the algorithm MINIMAX-BT, would not generate all successors of a node at once, but generate and evaluate only one successor node at a time.
- Usually, the algorithm MINIMAX-DFS (as well as the algorithm MINIMAX-BT) generates more nodes than necessary. The algorithm SOLVE introduced below illustrates this fact. Successor nodes have to be processed recursively only, if the resulting value may influence the maximum/minimum. Note, however, that the algorithm SOLVE employs a two-valued evaluation function e.

Propagation Algorithms for Game Trees [MINIMAX-DFS, ALPHA-BETA]

Algorithm: SOLVE

Input: n. A node in a game tree T.
successors(n). Returns the successors of node n.
k. Depth bound for evaluation.
$e(n)$. Evaluation function (two-valued $\{win, loss\}$) for a leaf node $n \in T$.

Output: The value $v(n)$ of the node n.

SOLVE$(n,$ *successors*$, k, e)$

```
1.  IF ( successors(n) = ∅ OR depth(n) = k )
    THEN RETURN(e(n));
2.  FOREACH n' IN successors(n) DO
      v(n') = SOLVE(n', successors, e);
      IF nodeType(n) = "Max"
      THEN IF v(n') = "win" THEN RETURN("win") // Early termination.
      ELSE IF v(n') = "loss" THEN RETURN("loss") // Early termination.
    ENDDO
3.  IF nodeType(n) = "Max"
    THEN RETURN("loss")
    ELSE RETURN("win")
```

Propagation Algorithms for Game Trees

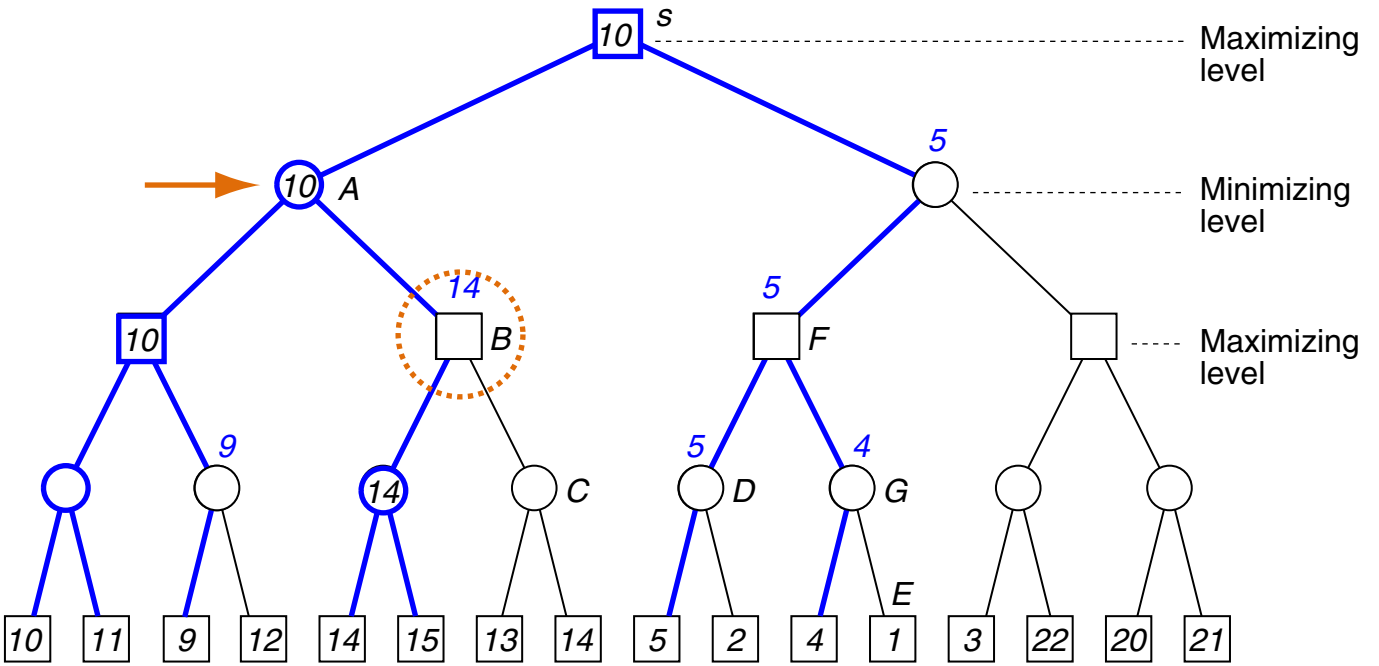
The pruning rationale used by the algorithm SOLVE is not restricted to two-valued evaluation functions but can be applied to multivalued (continuous) evaluation functions as well.

Overview of propagation algorithms:

	two-valued evaluation function	multivalued evaluation function
without pruning	MINIMAX-DFS	
with pruning	SOLVE	ALPHA-BETA

Propagation Algorithms for Game Trees

Game tree with propagated minimax values:

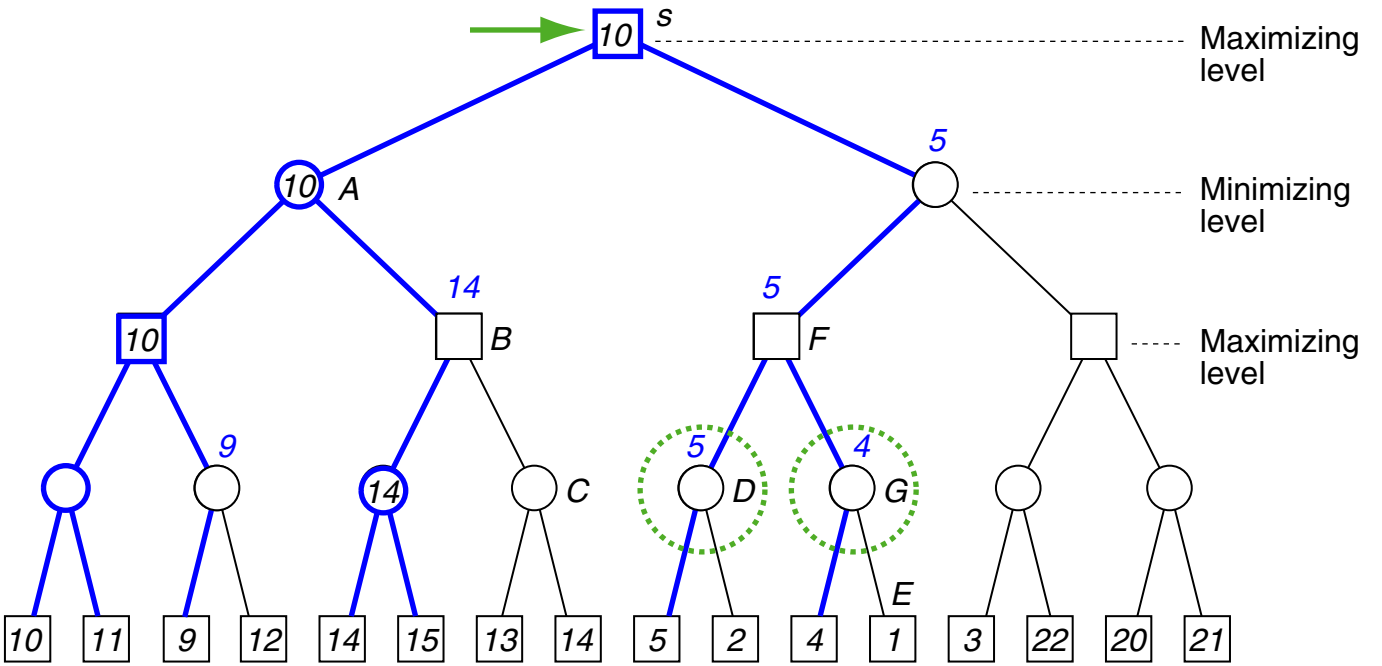


Argumentation:

1. The inspection of node B (its successors) yields information whether $v(A)$ must be decreased. What must hold for B to decrease $v(A)$?
   If we learn that this condition cannot be fulfilled anymore, the inspection of B can be aborted.

Propagation Algorithms for Game Trees

Game tree with propagated minimax values:



Argumentation:

2. The inspection of the nodes D or G (their successors) yield information whether $v(s)$ can be increased. What must hold for D or G to increase $v(s)$?
   If we learn that this condition cannot be fulfilled anymore, the inspection of D (G) can be aborted.

Remarks:

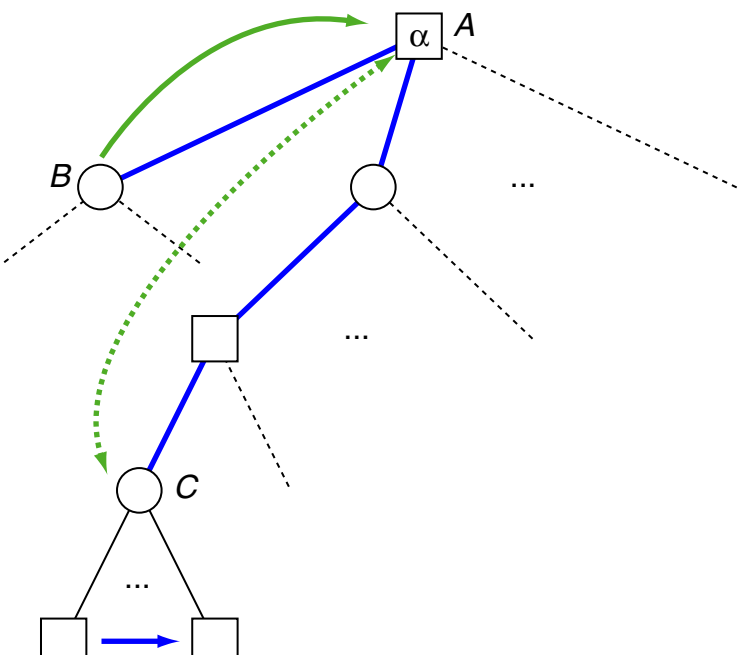
- The generalization of the previous argumentation will lead to the concept of α-bounds and β-bounds.
- For the node D, the value 10, which is here obtained from s, forms an α-bound.
- For the node B, the value 10, which is here obtained from A, forms a β-bound.
- Without loss of generality, the inspection of the nodes goes strictly from left to right.

Propagation Algorithms for Game Trees

Intuition of α-Bounds

- The α-bound is a lower bound and is used to prune (= to abort the inspection) of a *Min* node n.
- The value of α is defined as the currently maximum value of all predecessors of n that are of type "*Max*".
- The inspection of the *Min* node n can be aborted if $v(n) \leq \alpha$.

Illustration:



Remarks:

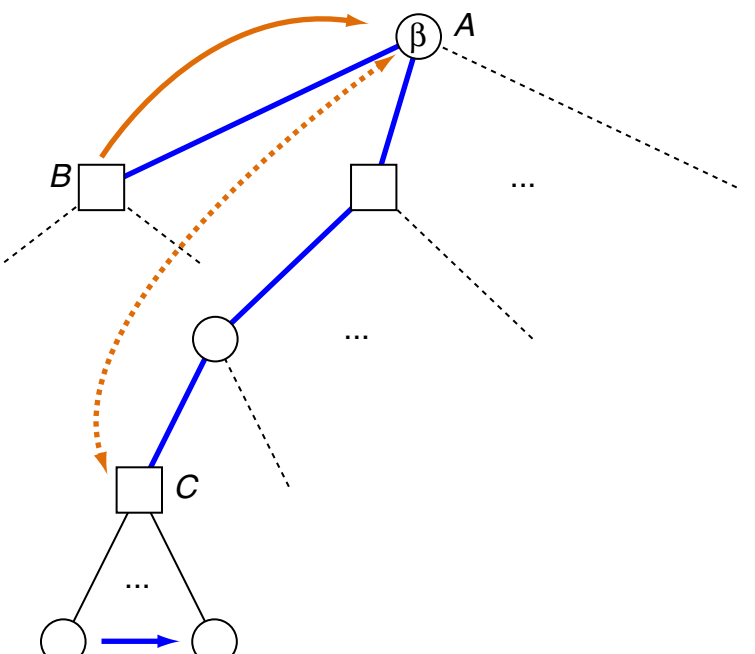
- The current α-bound can be found at the *Max* node A; it was propagated from the *Min* node B.
  Q. What must hold for the *Min* node C to increase $v(A) = \alpha$?
  A. C can increase α as long as $v(C) > \alpha$ holds.
- Observe that the successors of the *Min* node C can never increase $v(C)$. Hence, as soon as C got a value $v(C) \leq \alpha$, the investigation of the remaining successors of C can be aborted.
- At each point in time holds: The current value of α defines for the highest (closest to the root) *Max* node $\hat{n}$ with $v(\hat{n}) = \alpha$ a lower bound of the final value that would be computed for $\hat{n}$ by the algorithm MINIMAX-DFS.

Propagation Algorithms for Game Trees

Intuition of β-Bounds

- The β-bound is an upper bound and is used to prune (= to abort the inspection) of a *Max* node n.
- The value of β is defined as the currently minimum value of all predecessors of n that are of type "*Min*".
- The inspection of the *Max* node n can be aborted if $v(n) \geq \beta$.

Illustration:



Remarks:

- The current β-bound can be found at the *Min* node A; it was propagated from the *Max* node B.
  Q. What must hold for the *Max* node C to decrease $v(A) = \beta$?
  A. C can decrease β as long as $v(C) < \beta$ holds.
- Observe that the successors of the *Max* node C can never decrease $v(C)$. Hence, as soon as C got a value $v(C) \geq \beta$, the investigation of the remaining successors of C can be aborted.
- At each point in time holds: The current value of β defines for the highest (closest to the root) *Min* node $\hat{n}$ with $v(\hat{n}) = \beta$ an upper bound of the final value that would be computed for $\hat{n}$ by the algorithm MINIMAX-DFS.

Propagation Algorithms for Game Trees

The α-β-pruning scheme:

> *"Perform the backtracking version of minimax search with one exception; if in the course of updating the minimax value of a given node n crosses a certain bound, then no further exploration is needed beneath that node; its current-value $v(n)$ can be transmitted to its father as if all of its sons have been evaluated."*

[Pearl 1981, p. 233]

Remarks:

- The effectiveness of α-β pruning depends on the evaluation order of the nodes in depth k.
- Given a fixed evaluation effort, it can be shown that—if the depth k nodes (evaluation function values) are randomly ordered—the attainable search depth is extended by 33% with α-β pruning. [Pearl 1981]

Propagation Algorithms for Game Trees

Algorithm: ALPHA-BETA

Input: n. A node in a game tree T.
successors(n). Returns the successors of node n.
k. Depth bound for evaluation.
$e(n)$. Evaluation function (real-valued) for a node $n \in T$.
α, β. Two numbers in $\mathbf{R}$ with $\alpha < \beta$. Initially, $\alpha = -\infty$, $\beta = +\infty$.

Output: The minimax value $v(n)$ of n for ALPHA-BETA$(n, \textit{successors}, k, e, \alpha = -\infty, \beta = +\infty)$,

in general a value $v'(n)$ with $v'(n) = \begin{cases} v(n) & \text{if } v(n) \in]\alpha, \beta[\\ \alpha & \text{if } v(n) \leq \alpha \\ \beta & \text{if } v(n) \geq \beta \end{cases}$

Propagation Algorithms for Game Trees [MINIMAX, SOLVE]

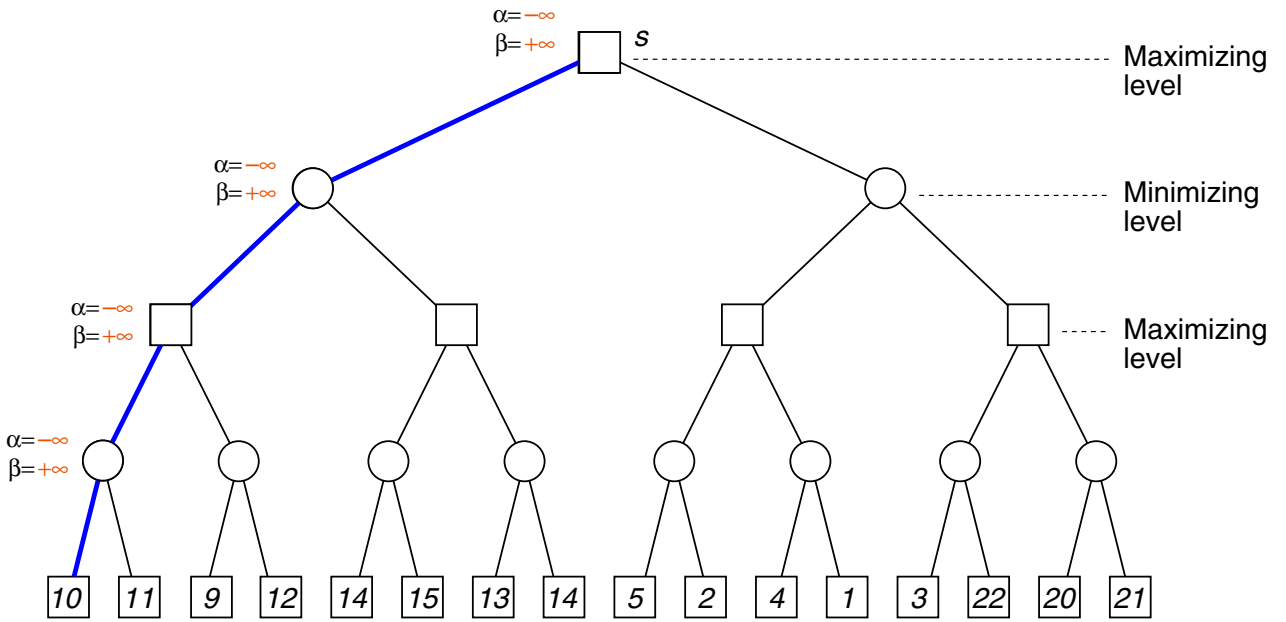
```
ALPHA-BETA(n, successors, k, e, α, β)
1.    IF ( successors(n) = ∅ OR depth(n) = k )
      THEN RETURN(e(n));
2.a   IF nodeType(n) = 'Max'
      THEN
        // Try to increase α in interval ]α,β[.
        FOREACH n' IN successors(n) DO
          α = max(α, ALPHA-BETA(n', successors, e, α, β));
          IF α ≥ β THEN RETURN(β);   // No improvement of bounds possible.
                   // A better option for the Min-player is already known.
        ENDDO
        RETURN(α);
2.b   ELSE   // nodeType(n) = 'Min'.
        // Try to decrease β in interval ]α,β[.
        FOREACH n' IN successors(n) DO
          β = min(β, ALPHA-BETA(n', successors, e, α, β));
          IF β ≤ α THEN RETURN(α);   // No improvement of bounds possible.
                   // A better option for the Max-player is already known.
        ENDDO
        RETURN(β);
      ENDIF
```

Remarks:

- The evaluation $v(n)$ of a node is not stored explicitly. Instead, the bound that is to be improved by $v(n)$ will be adapted directly: α bounds for *Max* nodes and β bounds for *Min* nodes.
- Therefore, the processing of a node is aborted if $\alpha \geq \beta$ (corresponding to $v(n) \geq \beta$ for *Max* nodes and $v(n) \leq \alpha$ for *Min* nodes).

Propagation Algorithms for Game Trees

Example: Propagating evaluations with ALPHA-BETA



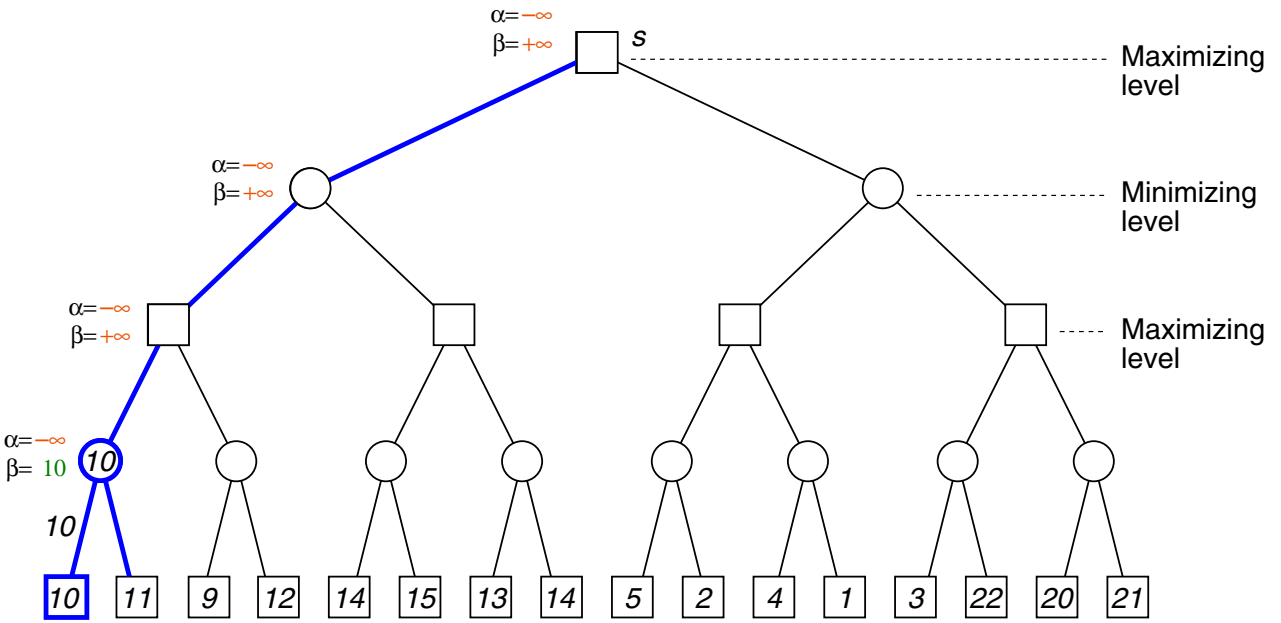
- The initial call of the algorithm is

$$\text{ALPHA-BETA}(s, \textit{successors}, k = 4, e(.), \alpha = -\infty, \beta = +\infty).$$

- Nodes in depth 4 can be seen as leaf nodes.
- Recursive calls with current bounds α, β are performed up to the maximum depth level.

Propagation Algorithms for Game Trees

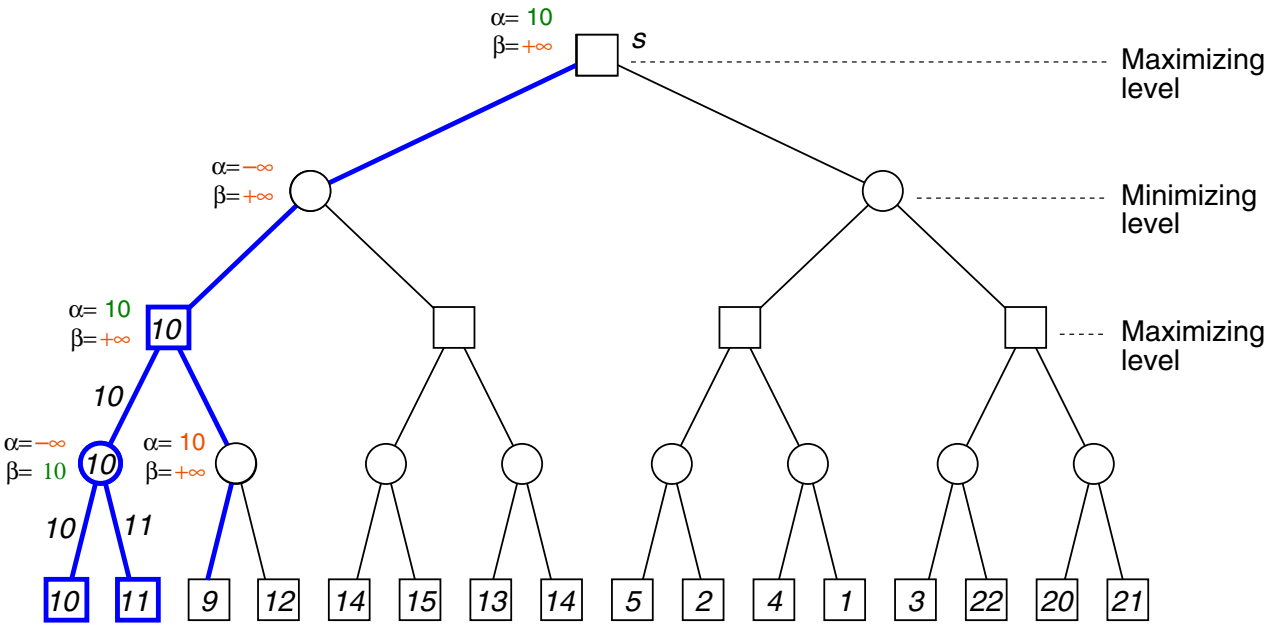
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 10$ is computed at maximum depth level and returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = -\infty < 10 = \beta$.
- A recursive call with current bounds α, β is performed for the next successor up to the maximum depth level.

Propagation Algorithms for Game Trees

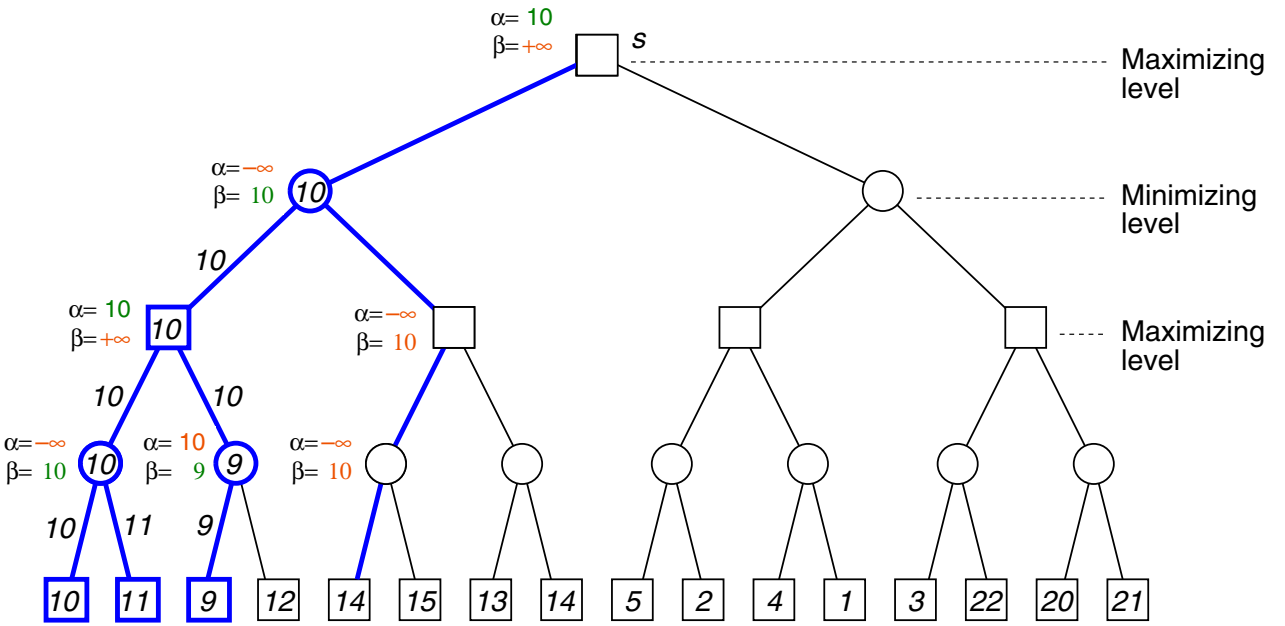
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 11$ is computed at maximum depth level and returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = -\infty < 10 = \beta$. Then, β is returned.
- Adaption of the α-bound in the *Max*-node with the first successor's evaluation results in $\alpha = 10 < +\infty = \beta$.
- A recursive call with current bounds α, β is performed for the next successor up to the maximum depth level.

Propagation Algorithms for Game Trees

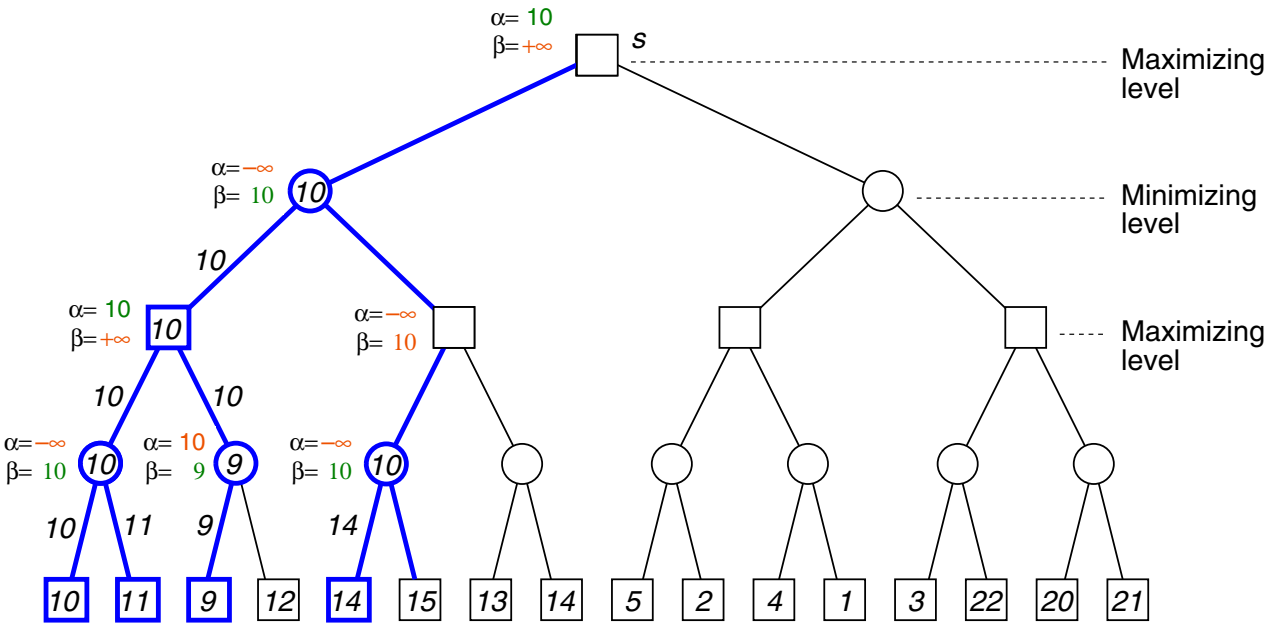
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 9$ is computed at maximum depth level and returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = 10 \geq 9 = \beta$. Then, α is returned.
- Adaption of the α-bound in the *Max*-node with the first successor's evaluation results in $\alpha = 10 < +\infty = \beta$. Then, α is returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = -\infty < 10 = \beta$. Then, the next recursive call with current bounds α, β is performed.

Propagation Algorithms for Game Trees

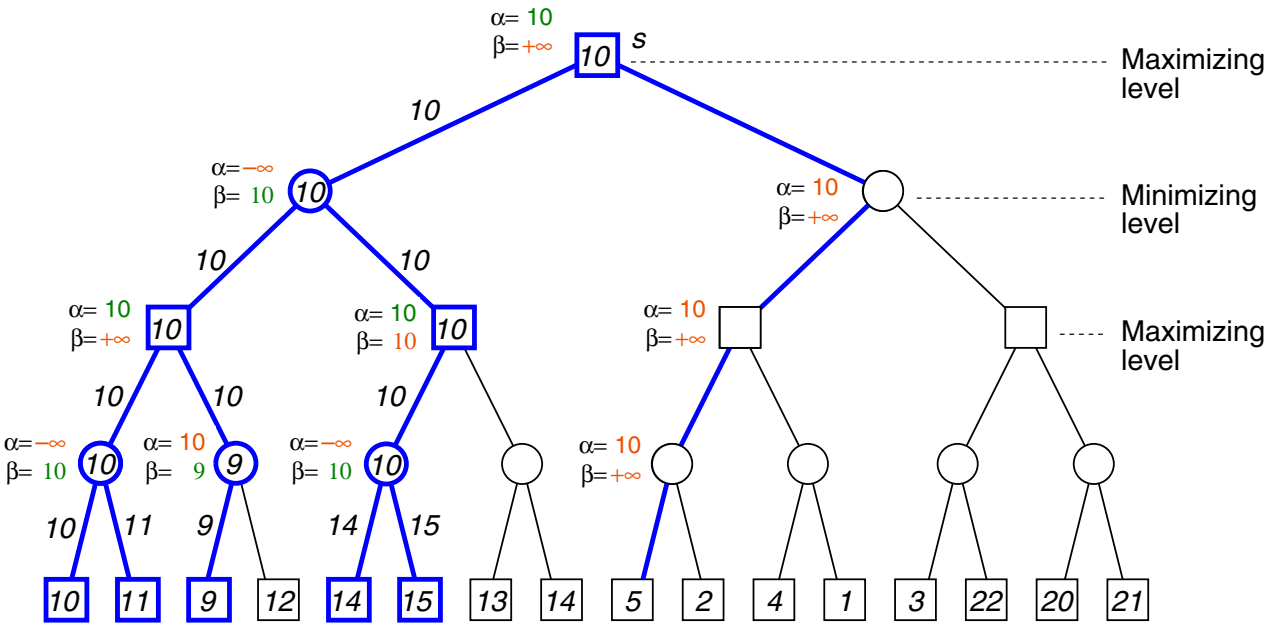
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 14$ is computed at maximum depth level and returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = -\infty < 10 = \beta$. There is no change in the value for β.
- A recursive call with current bounds α, β is performed for the next successor up to the maximum depth level.

Propagation Algorithms for Game Trees

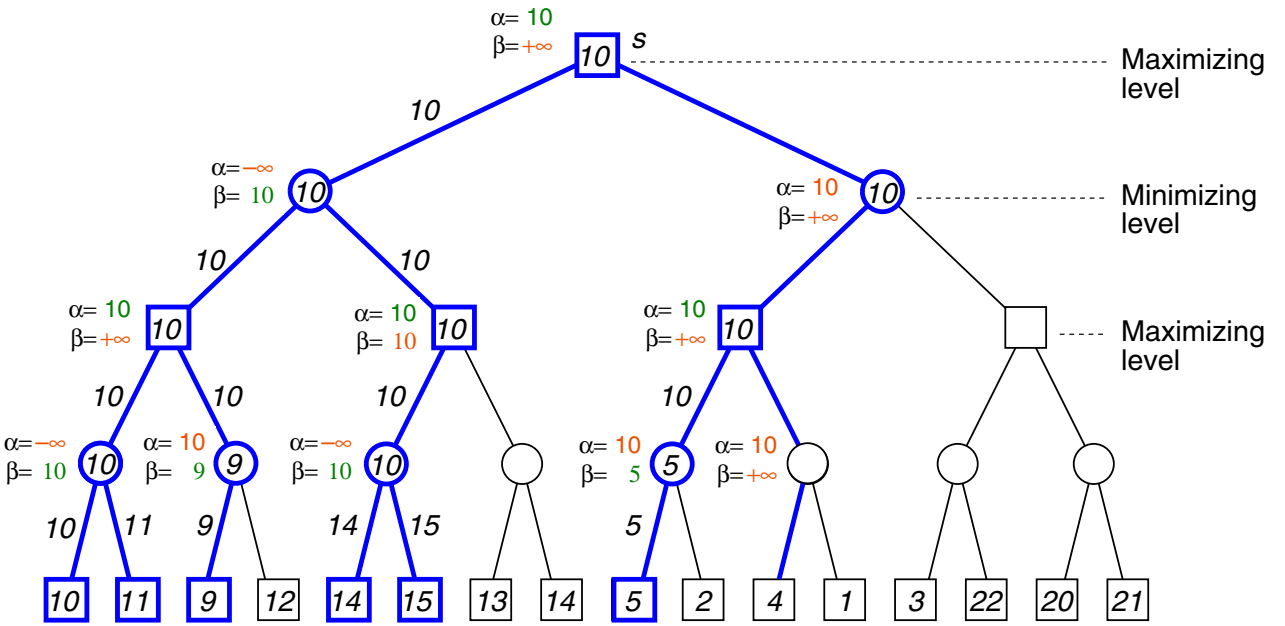
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 15$ is computed at maximum depth level and returned.
- Adaption of β in the *Min*-node results in $\alpha = -\infty < 10 = \beta$. Then, β is returned.
- Adaption of α in the *Max*-node results in $\alpha = 10 \geq 10 = \beta$. Then, β is returned.
- Adaption of β in the *Min*-node results in $\alpha = -\infty < 10 = \beta$. Then, β is returned.
- Adaption of α in the *Max*-node results in $\alpha = 10 \geq +\infty = \beta$.
- A recursive call with current bounds α, β is performed for the next successor up to the maximum depth level.

Propagation Algorithms for Game Trees

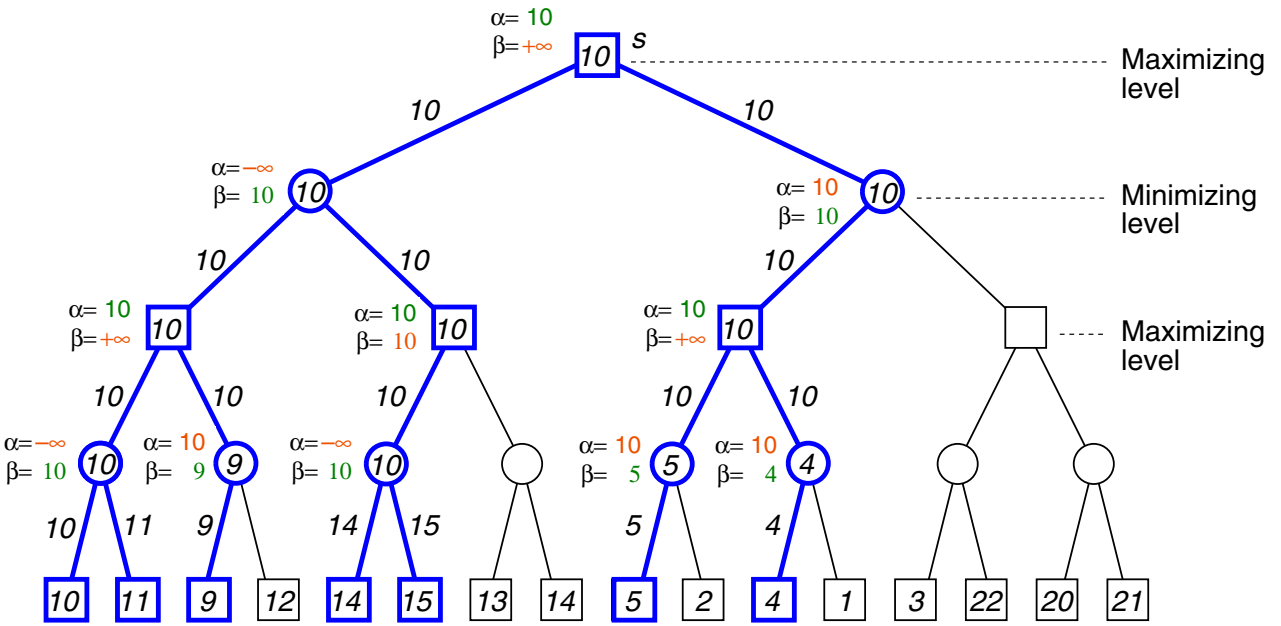
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 5$ is computed at maximum depth level and returned.
- Adaption of the β-bound in the *Min*-node with the first successor's evaluation results in $\alpha = 10 \geq 5 = \beta$. Then, α is returned.
- Adaption of the α-bound in the *Max*-node with the first successor's evaluation results in $\alpha = 10 < +\infty = \beta$. There is no change in the value for α.
- A recursive call with current bounds α, β is performed for the next successor up to the maximum depth level.

Propagation Algorithms for Game Trees

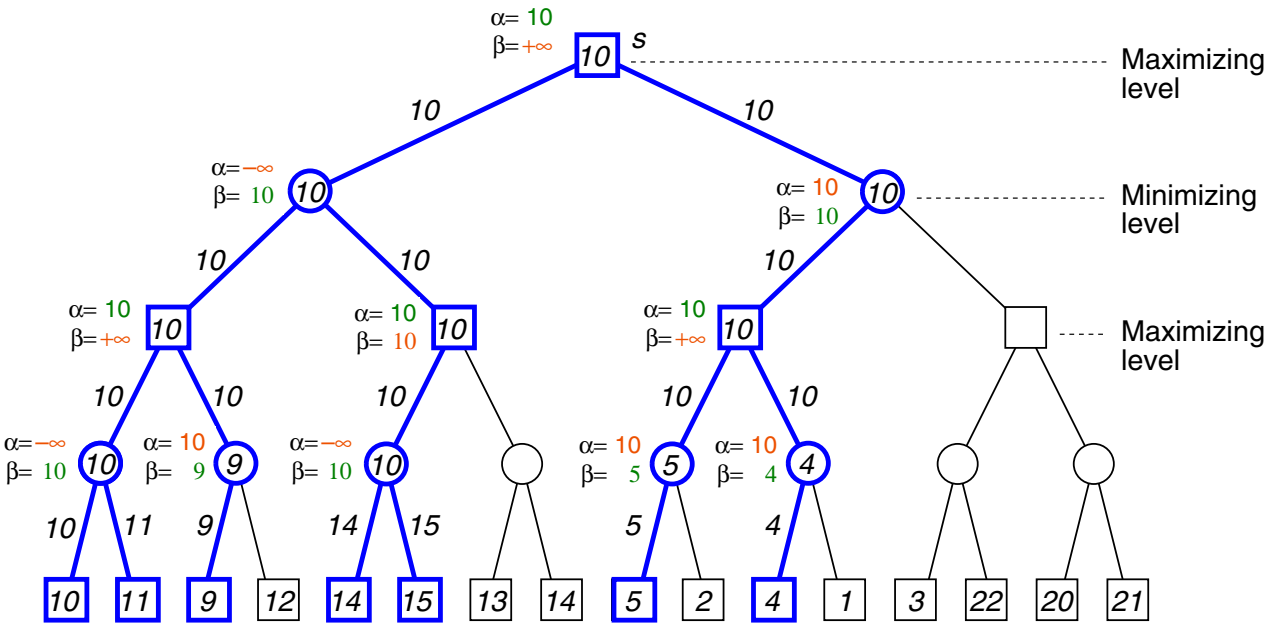
Example: Propagating evaluations with ALPHA-BETA (continued)



- The value $e(n) = 4$ is computed at maximum depth level and returned.
- Adaption of β in the *Min*-node results in $\alpha = 10 \geq 4 = \beta$. Then, α is returned.
- Adaption of α in the *Max*-node results in $\alpha = 10 < +\infty = \beta$. Then, α is returned.
- Adaption of β in the *Min*-node results in $\alpha = 10 \geq 10 = \beta$. Then, α is returned.
- Adaption of α in the *Max*-node results in $\alpha = 10 < +\infty = \beta$. Then, α is returned.
- It holds $\alpha = v(s)$.

Propagation Algorithms for Game Trees

Example: Propagating evaluations with ALPHA-BETA (continued)



Summary:

- α-bounds increase, β-bounds decrease.
- The interval $]\alpha, \beta[$ provided in a (recursive) call defines the range of helpful values.
- The interval $]\alpha, \beta[$ is decreased in a node by either decreasing β or increasing α.
- If the interval $]\alpha, \beta[$ runs empty, processing is stopped, the unchanged bound is returned.
- Otherwise, the changed bound is returned, i.e., the subtree gives an improvement.