Chapter Overview
Games

- Motivation
- Objectives
- Games and AI
- Games and Search
- Perfect Decisions
- Imperfect Decisions
- Alpha-Beta Pruning
- Games with Chance
- Games and Computers
- Important Concepts and Terms
- Chapter Summary

http://media.arstechnica.com/news.media/dogs-playing-poker.jpg

Logistics - Oct. 18, 2012

- **AI Nugget presentations scheduled**
  - Section 1:
    - William Budney: SwiftKey (delayed from Oct. 18)
    - Haikal Saliba: quantum algorithms in machine learning (delayed from Oct. 18)
    - Joseph Hain: Linux MCE - Home Automation
    - Jonathan Uder: Google's Autonomous Vehicle
    - Doug Gallatin: BWAPI and competitions, Overmind AI in detail
    - Dennis Waldron: ICODES
  - Section 3:
    - Andrew Guenther: Valve's Left 4 Dead AI Director (delayed from Oct. 18)
    - Kris Almario: Multi Robot Soccer AI
    - Ilya Seletsky: Action Game AI (FPS)
- **Assignments**
  - A1 due tonight (Tue, Oct. 23, end of the day)
  - late submission penalty: 10% per business day
- **Labs**
  - Lab 5 due tonight
  - Lab 6 available
- **Quizzes**
  - Quiz 5 available
- **Project**
  - mid-quarter project fair on Thu, Oct. 25
  - revise project documentation

Motivation

- examine the role of AI methods in games
- some game provide challenges that can be formulated as abstract competitions with clearly defined states and rules
  - programs for some games can be derived from search methods
  - narrow view of games
- games can be used to demonstrate the power of computer-based techniques and methods
- more challenging games require the incorporation of specific knowledge and information
- expansion of the use of games
  - from entertainment to training and education

Objectives

- explore the combination of AI and games
- understand the use and application of search methods to game programs
  - apply refined search methods such as minimax to simple game configurations
  - use alpha-beta pruning to improve the efficiency of game programs
  - understand the influence of chance on the solvability of chance-based games
- evaluation of methods
  - suitability of game techniques for specific games
  - suitability of AI methods for games

Games and Computers

- games offer concrete or abstract competitions
  - “I’m better than you!”
- some games are amenable to computer treatment
  - mostly mental activities
  - well-formulated rules and operators
  - accessible state
- others are not
  - emphasis on physical activities
  - rules and operators open to interpretation
    - need for referees, mitigation procedures
  - state not (easily or fully) accessible

Games and AI

- traditionally, the emphasis has been on a narrow view of games
  - formal treatment, often as an expansion of search algorithms
- more recently, AI techniques have become more important in computer games
  - computer-controlled characters (agents)
  - more sophisticated story lines
  - more complex environments
  - better overall user experience

Cognitive Game Design

- story development
  - generation of interesting and appealing story lines
  - variations in story lines
  - analysis of large-scale game play
- character development
  - modeling and simulation of computer-controlled agents
  - possibly enhancement of user-controlled agents
- immersion
  - strong engagement of the player’s mind
- emotion
  - integration of plausible and believable motion in characters
  - consideration of the user’s emotion
- pedagogy
  - achievement of “higher” goals through entertainment

Game Analysis

- often deterministic
  - the outcome of actions is known
  - sometimes an element of chance is part of the game
    - e.g. dice
- two-player, turn-taking
  - one move for each player
- zero-sum utility function
  - what one player wins, the other must lose
- often perfect information
  - fully observable, everything is known to both players about the state of the environment (game)
  - not for all games
    - e.g. card games with “private” or “hidden” cards
    - Scrabble

Games as Adversarial Search

- many games can be formulated as search problems
- the zero-sum utility function leads to an adversarial situation
  - in order for one agent to win, the other necessarily has to lose
- factors complicating the search task
  - potentially huge search spaces
  - elements of chance
  - multi-person games, teams
  - time limits
  - imprecise rules

Difficulties with Games

- games can be very hard search problems
  - yet reasonably easy to formalize
  - finding the *optimal* solution may be impractical
    - a solution that beats the opponent is "good enough"
  - unforgiving
    - a solution that is "not good enough" leads to higher costs, and to a loss to the opponent
- example: chess
  - size of the search space
    - branching factor around 35
    - about 50 moves per player
    - about 35^{100} or 10^{154} nodes
      - about 10^{40} *distinct* nodes (size of the search graph)

Games and Search

- the actions of an agent playing a game can often be formulated as a search problem
- some factors make the use of search methods challenging
  - multiple players
  - actions of opponents
  - chance events (e.g. dice)
  - consideration of probabilities
  - ...

Search Problem Formulation

- initial state
  - board, positions of pieces
  - whose turn is it
- successor function (operators)
  - list of *(move, state)*
  - defines the legal moves, and the resulting states
- terminal test
  - also called goal test
  - determines when the game is over
  - calculate the result
    - usually win, lose, draw; sometimes a score (see below)
- utility or payoff function
  - numeric value for the outcome of a game

Single-Person Game

- conventional search problem
  - identify a sequence of moves that leads to a winning state
  - examples: Solitaire, dragons and dungeons, Rubik's cube
  - little attention in AI
- some games can be quite challenging
  - some versions of solitaire
  - Rubik's cube
    - a heuristic for this was found by the Absolver theorem prover

Contingency Problem

- uncertainty due to the moves and motivations of the opponent
  - tries to make the game as difficult as possible for the player
    - attempts to maximize its own, and thus minimize the player's utility function value
  - different from contingency due to neutral factors, such as
    - chance
    - outside influence

Two-Person Games

- games with two opposing players
  - often called MIN and MAX
  - usually MAX moves first, then they take turns
  - in game terminology, a *move* comprises two steps (“*plies*”)
    - *one* by MAX and one by MIN
- MAX wants a strategy to find a winning state
  - no matter what MIN does
- MIN does the same
  - or at least tries to prevent MAX from winning
- full information
  - both players know the full state of the environment
- partial information
  - one player only knows part of the environment
  - some aspects may be hidden from the opponent, or from both players

Perfect Decisions

- based on an rational (optimal) strategy for MAX
  - traverse all relevant parts of the search tree
    - this must include possible moves by MIN
  - identify a path that leads MAX to a winning state
- often impractical
  - time and space limitations

MiniMax Strategy

- optimal strategy for MAX
  - not very practical

•generate the whole game tree
•calculate the value of each terminal state
  •based on the utility function
•calculate the utilities of the higher-level nodes
  •starting from the leaf nodes up to the root
•MAX selects the value with the highest node
•MAX assumes that MIN in its move will select the node that minimizes the value

MiniMax Value

- utility of being in the state that corresponds to a node
  - from MAX's perspective: MAX tries to move to a state with the maximum value, MIN to one with the minimum
  - assumes that both players play optimally

```
function MiniMax-Value(state) returns a utility value
  if Terminal-Test(state) then
    return Utility(state)
  else if Max is to move then
      return  the highest MiniMax-Value of Successors(state)
    else
      return the lowest MiniMax-Value of Successors(state)
```

MiniMax Algorithm

- selects the best successor from a given state
  - invokes MINIMAX-VALUE for each successor state

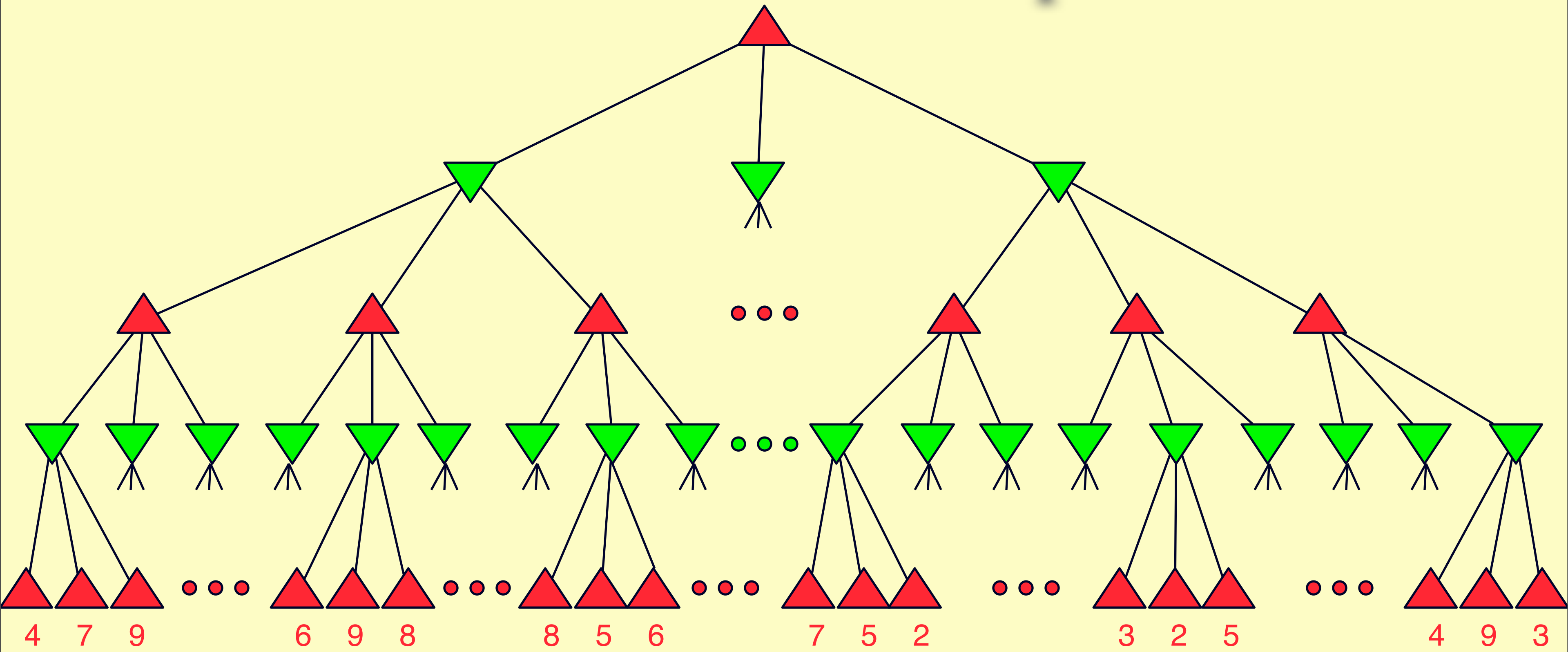
```
function MiniMax-Decision(state) returns action
    for each s  in  Successors[state] do
   Value[s] := MiniMax-Value(s)
    end
   return action  with the highest Value[s]
```

MiniMax Properties

- based on depth-first
  - recursive implementation
- time complexity is $O(b^m)$
  - exponential in the number of moves
- space complexity is $O(b*m)$

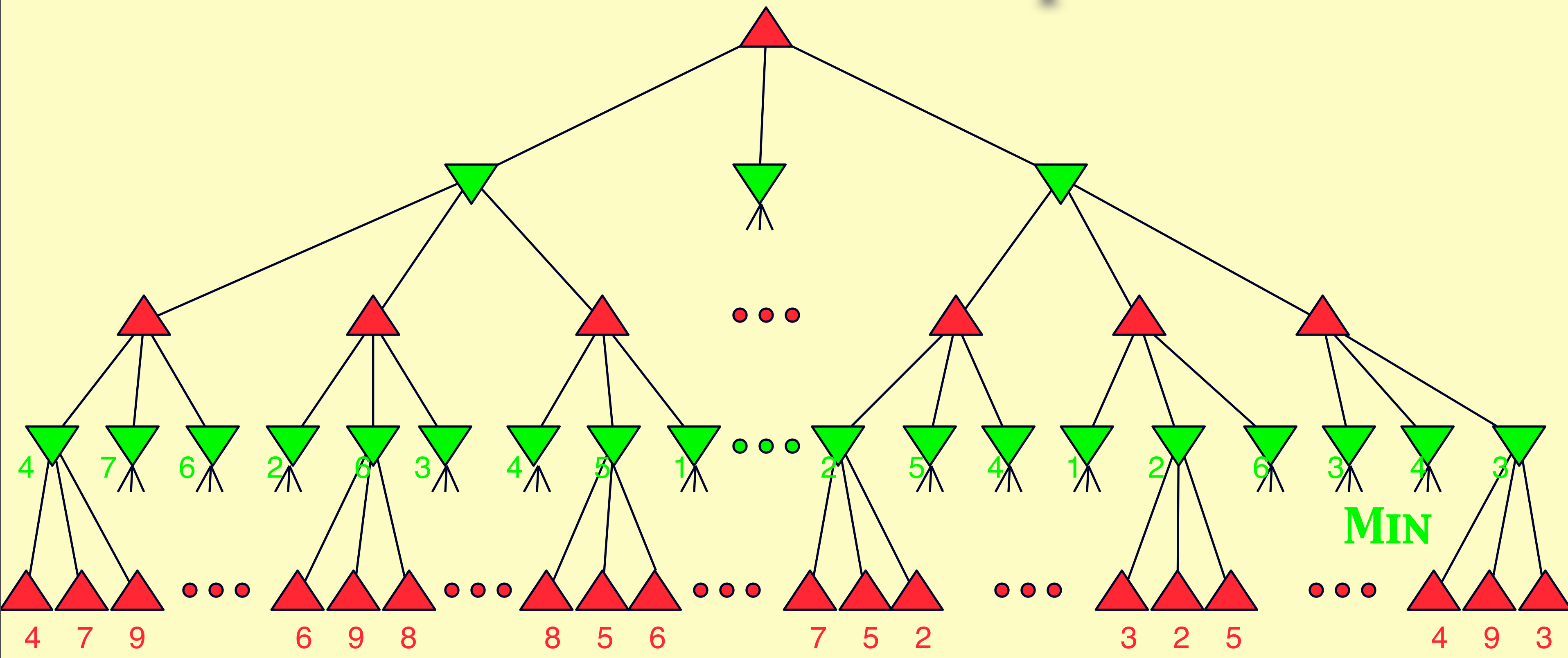
b	branching factor
m	maximum depth of the search tree

MiniMax Example



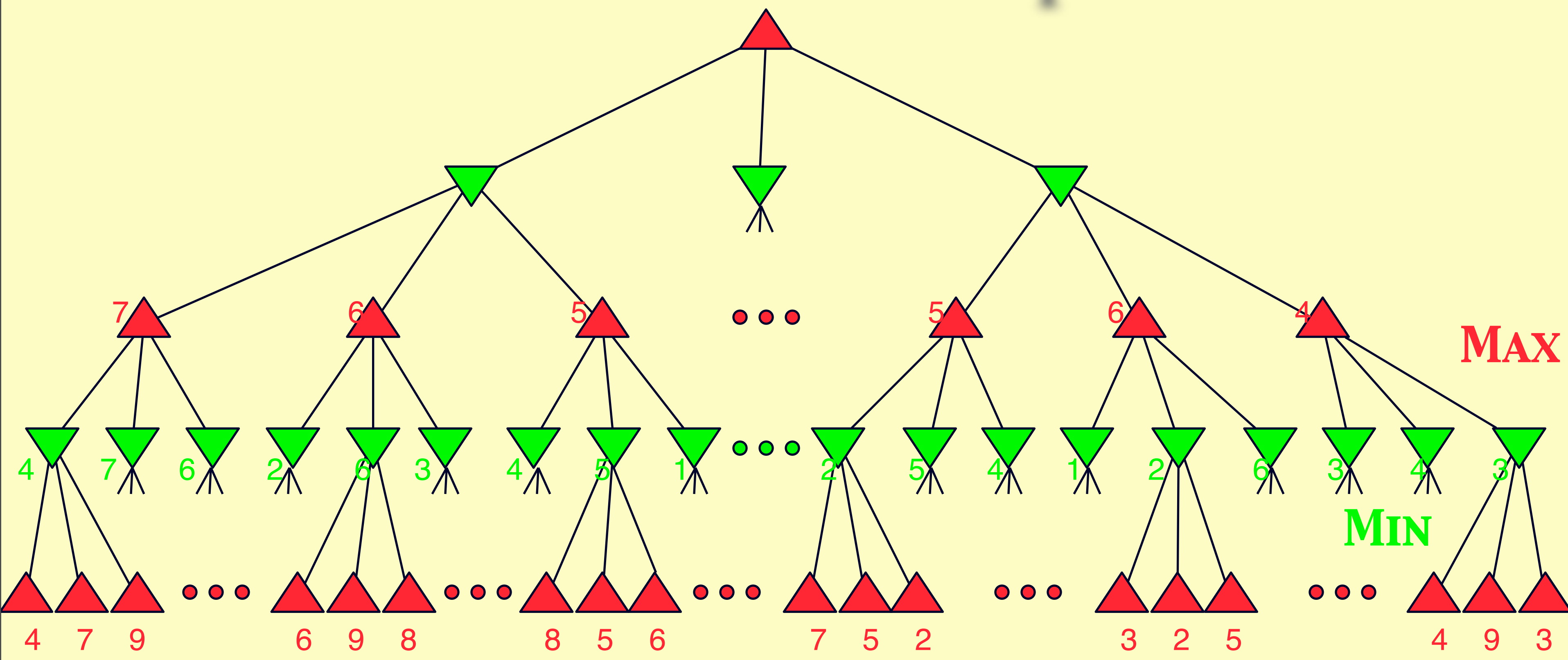
terminal nodes: values calculated from the utility function

MiniMax Example

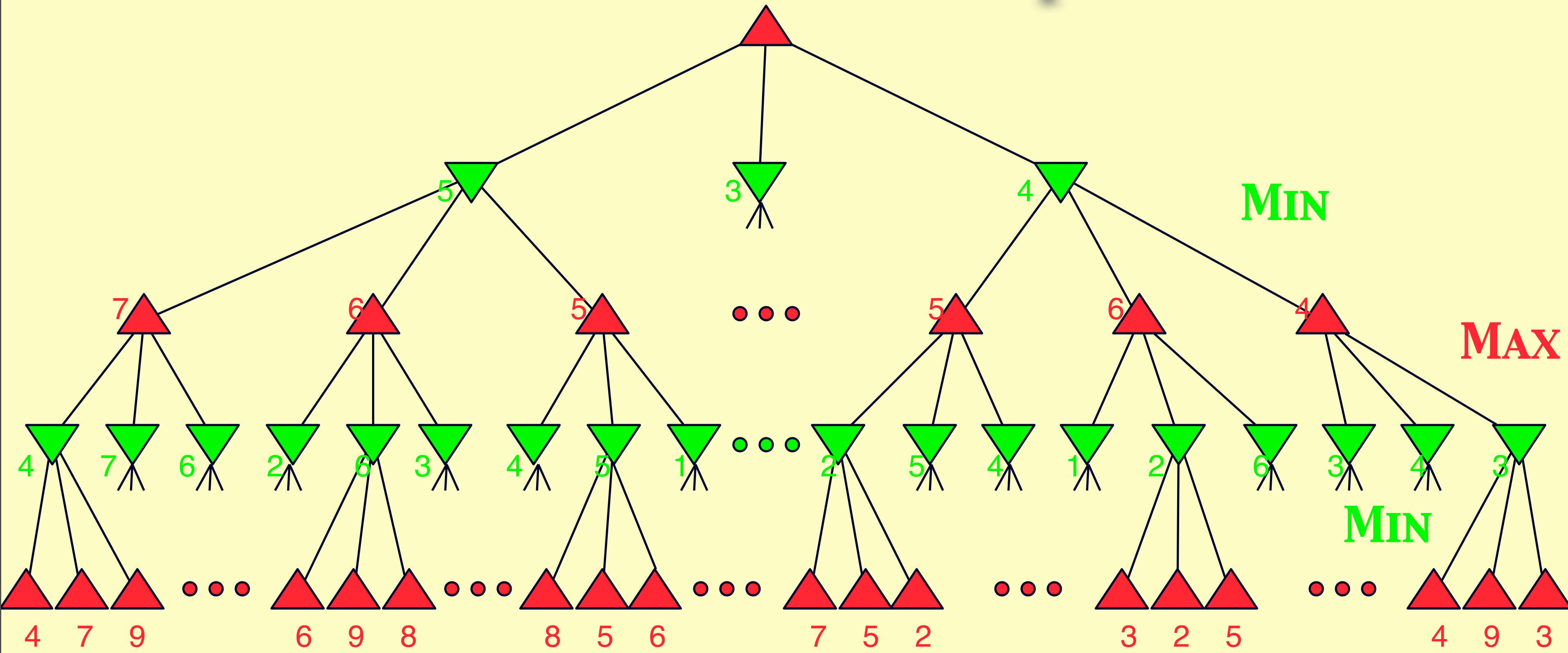


other nodes: values calculated via minimax algorithm

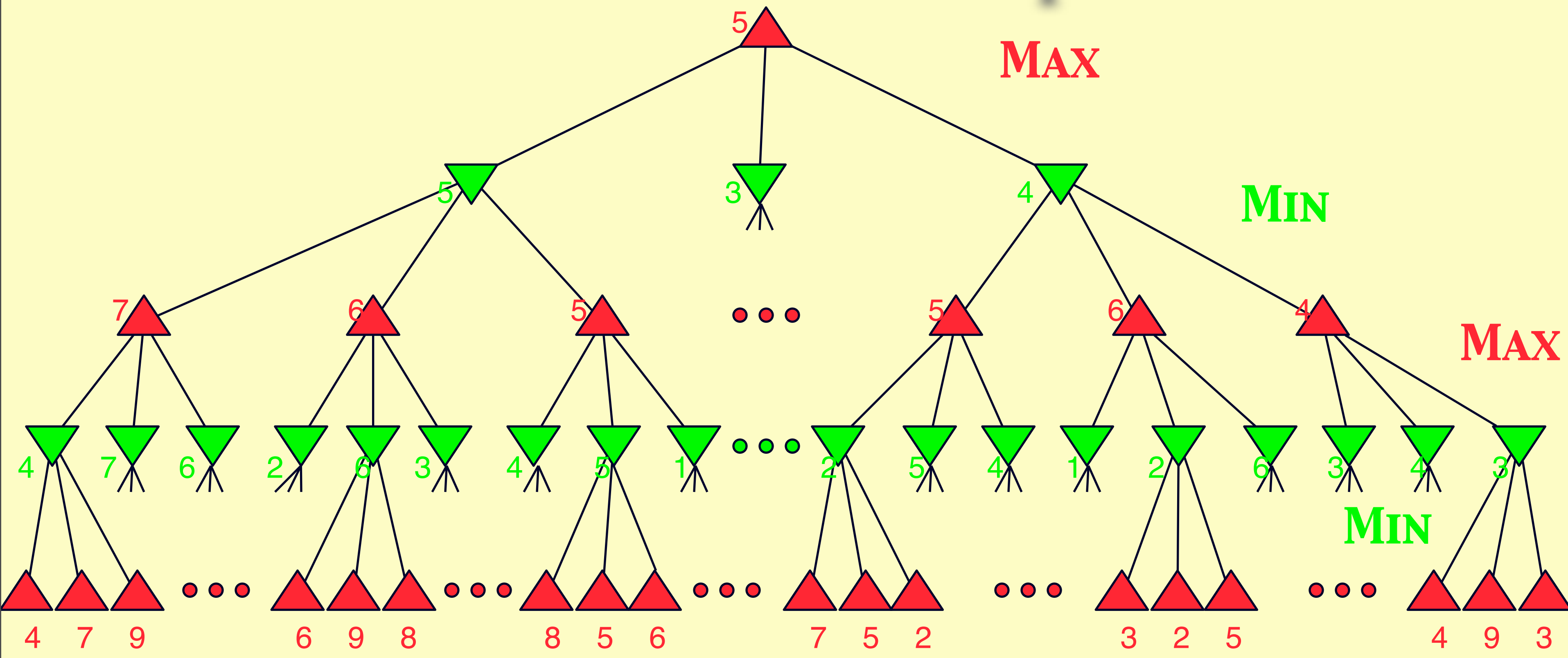
MiniMax Example



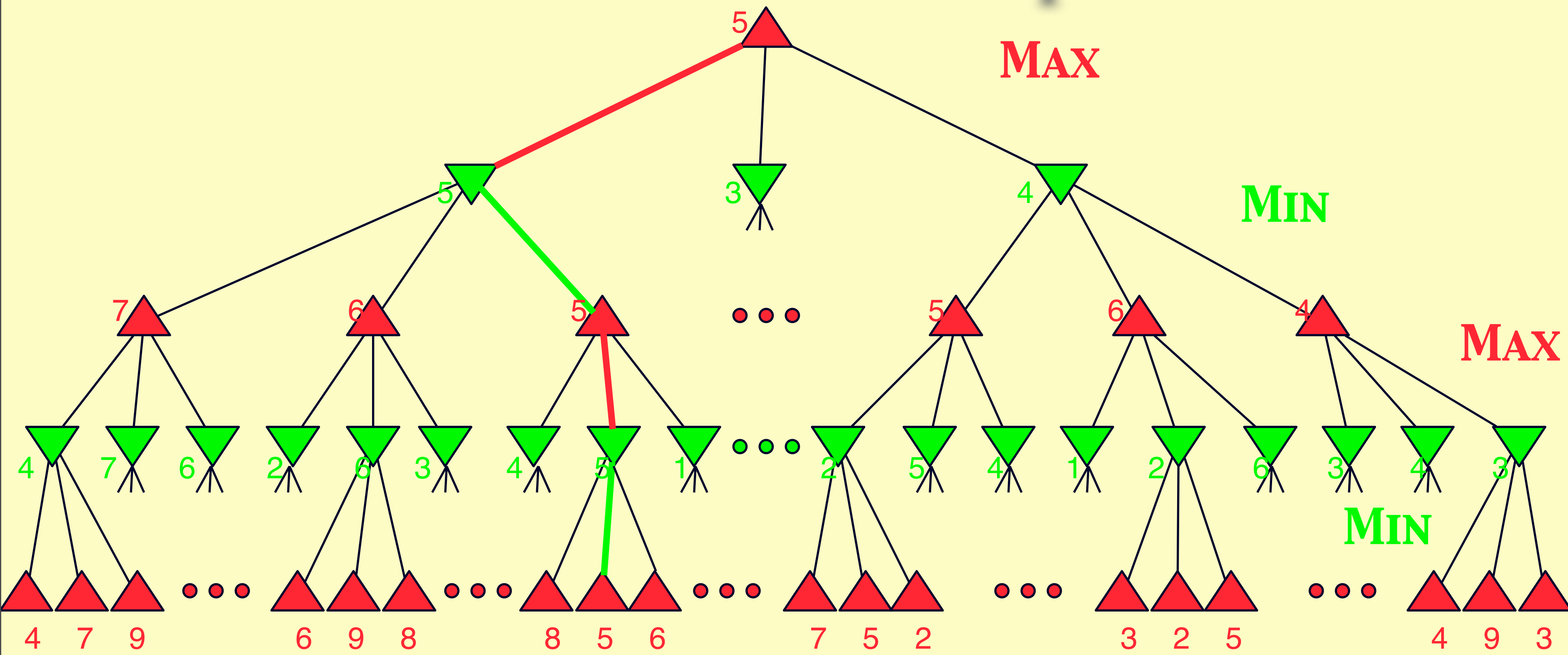
MiniMax Example



MiniMax Example



MiniMax Example



moves by MAX and countermoves by MIN

MiniMax Observations

- the values of some of the leaf nodes are irrelevant for decisions at the next level
- this also holds for decisions at higher levels
- as a consequence, under certain circumstances, some parts of the tree can be disregarded
  - it is possible to still make an optimal decision without considering those parts

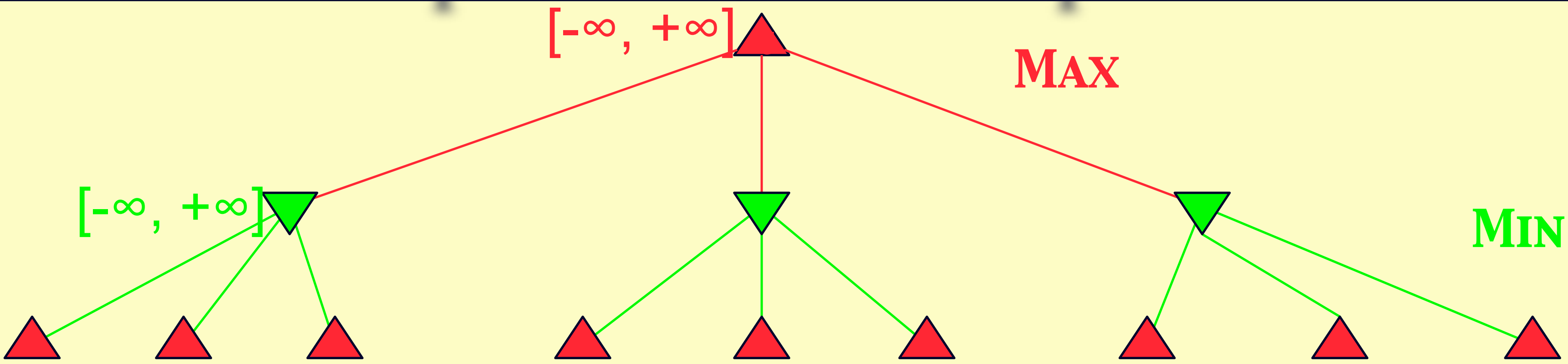
Pruning

- discards parts of the search tree
  - guaranteed not to contain good moves
  - guarantee that the solution is not in that branch or sub-tree
    - if both players make optimal (rational) decisions, they will never end up in that part of the search tree
    - sub-optimal moves by the opponent may lead into that part
      - may increase the amount of calculations for the player, but does not change the outcome of the game
- results in substantial time and space savings
  - as a consequence, longer sequences of moves can be explored
  - the leftover part of the task may still be exponential, however

Alpha-Beta Pruning

- certain moves are not considered
  - won't result in a better evaluation value than a move further up in the tree
  - they would lead to a less desirable outcome
- applies to moves by both players
  - α indicates the best choice for **MAX** so far never decreases
  - β indicates the best choice for **MIN** so far never increases
- extension of the minimax approach
  - results in the same sequence of moves as minimax, but with less overhead
  - prunes uninteresting parts of the search tree

Alpha-Beta Example 1

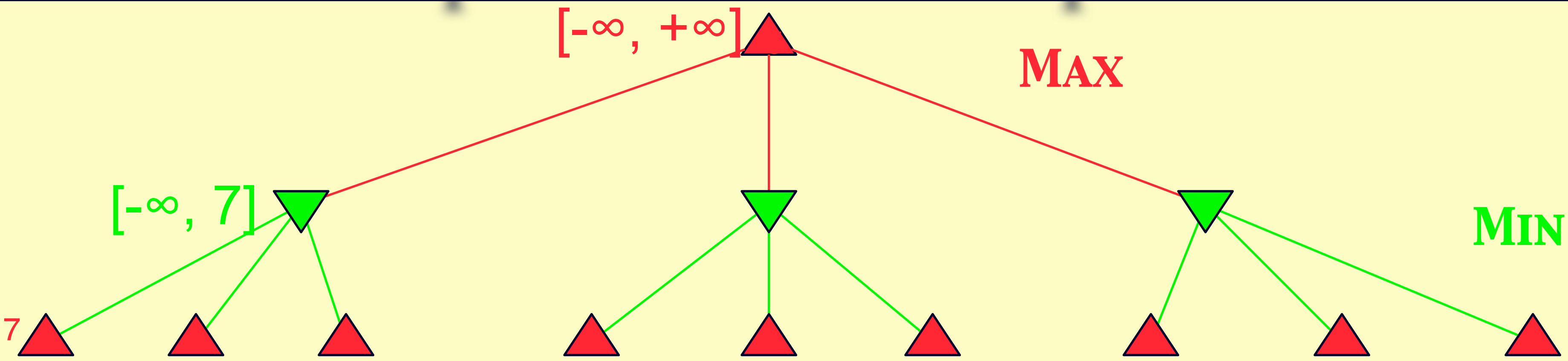


α best choice for MAX ?

β best choice for MIN ?

- we assume a depth-first, left-to-right search as basic strategy
- the range of the possible values for each node are indicated
  - initially [-∞, +∞]
  - from MAX's or MIN's perspective
  - these *local* values reflect the values of the sub-trees in that node; the *global* values α and β are the best overall choices so far for MAX or MIN

Alpha-Beta Example 2

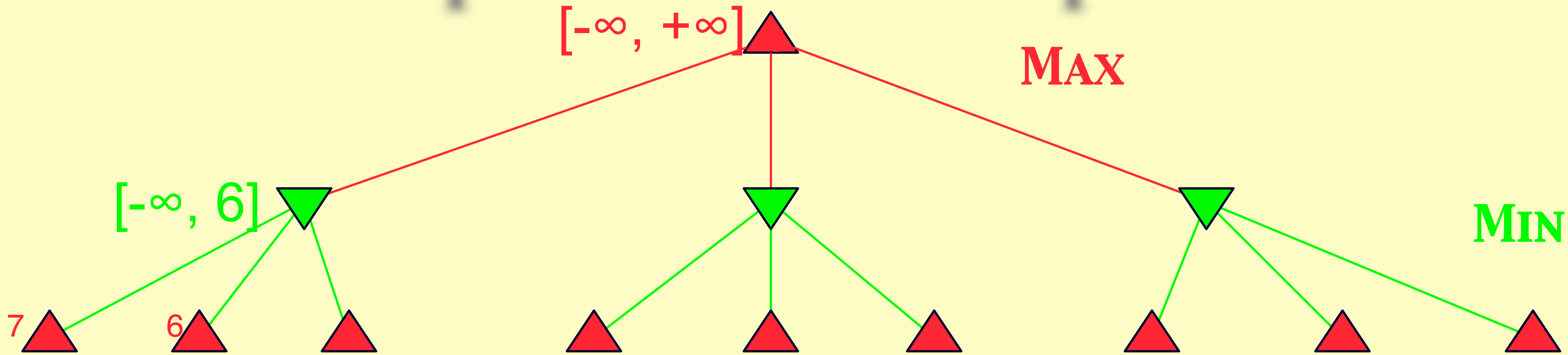


α best choice for **MAX** ?

β best choice for **MIN** 7

- **MIN** obtains the first value from a successor node

Alpha-Beta Example 3

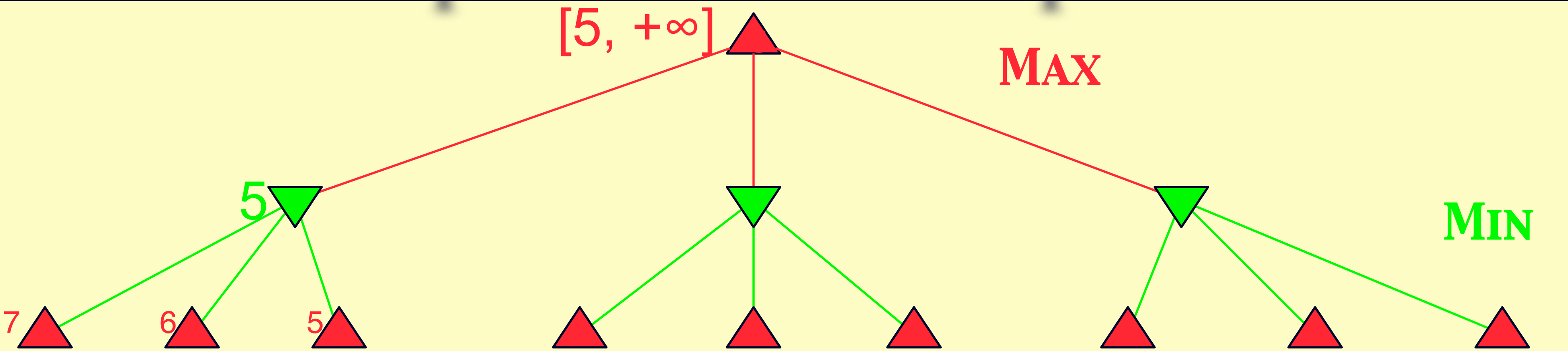


α best choice for MAX ?

β best choice for MIN 6

- MIN obtains the second value from a successor node

Alpha-Beta Example 4

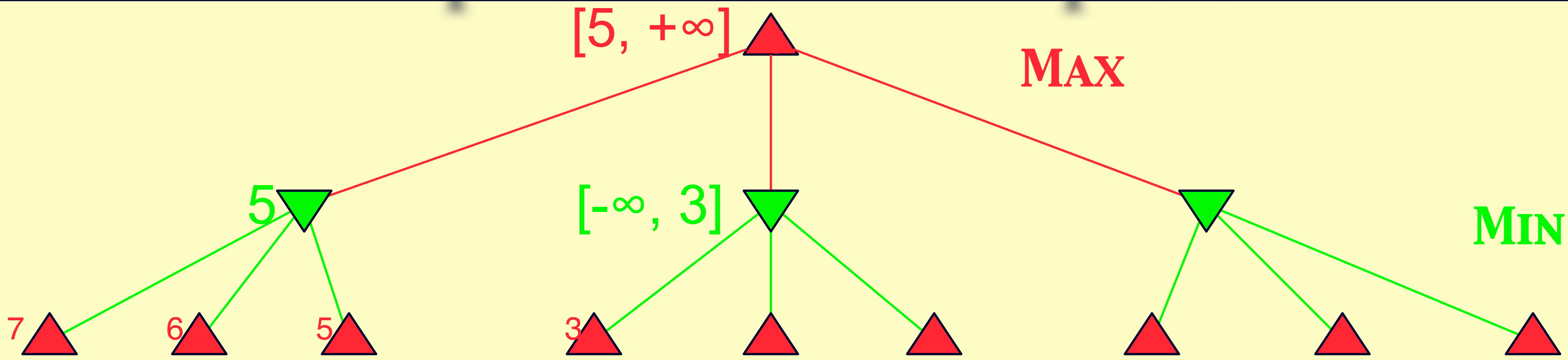


α best choice for **MAX** 5

β best choice for **MIN** 5

- **MIN** obtains the third value from a successor node
- this is the last value from this sub-tree, and the exact value is known
- **MAX** now has a value for its first successor node, but hopes that something better might still come

Alpha-Beta Example 5

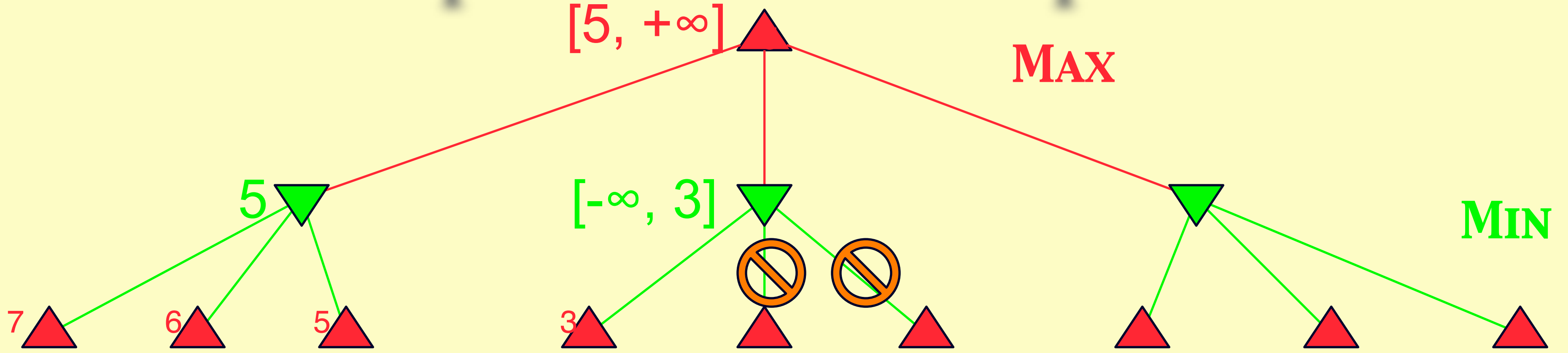


α best choice for MAX 5

β best choice for MIN 3

- MIN continues with the next sub-tree, and gets a better value
- MAX has a better choice from its perspective, however, and will not consider a move in the sub-tree currently explored by MIN
  - initially [-∞, +∞]

Alpha-Beta Example 6

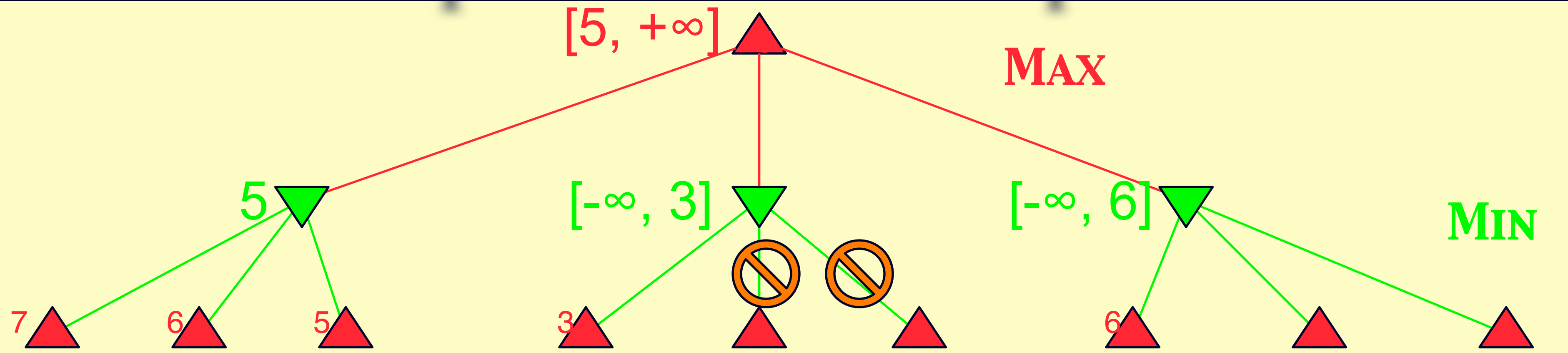


α best choice for **MAX** 5

β best choice for **MIN** 3

- **MIN** knows that **MAX** won't consider a move to this sub-tree, and abandons it
- this is a case of *pruning*, indicated by 

Alpha-Beta Example 7

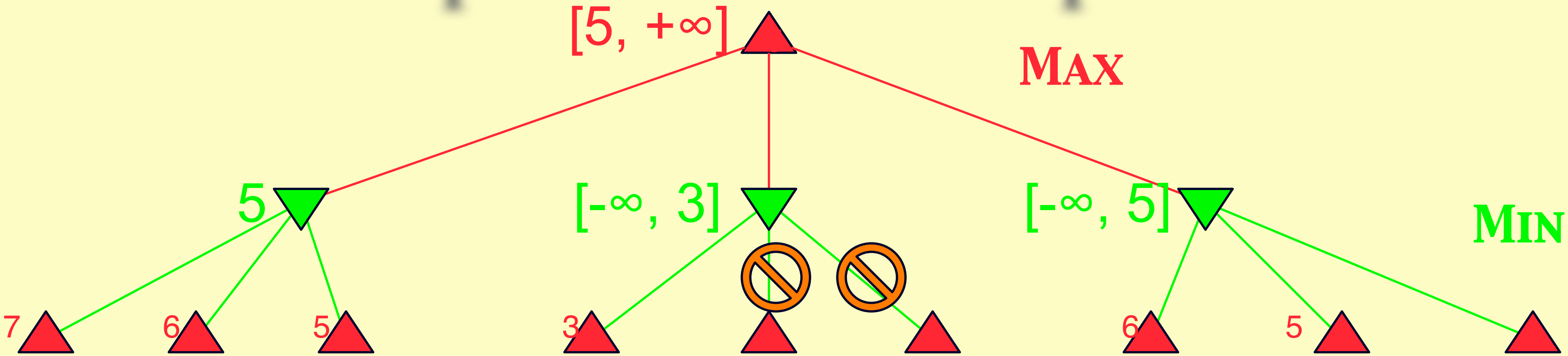


α best choice for **MAX** 5

β best choice for **MIN** 3

- **MIN** explores the next sub-tree, and finds a value that is worse than the other nodes at this level
- if **MIN** is not able to find something lower, then **MAX** will choose this branch, so **MIN** must explore more successor nodes

Alpha-Beta Example 8

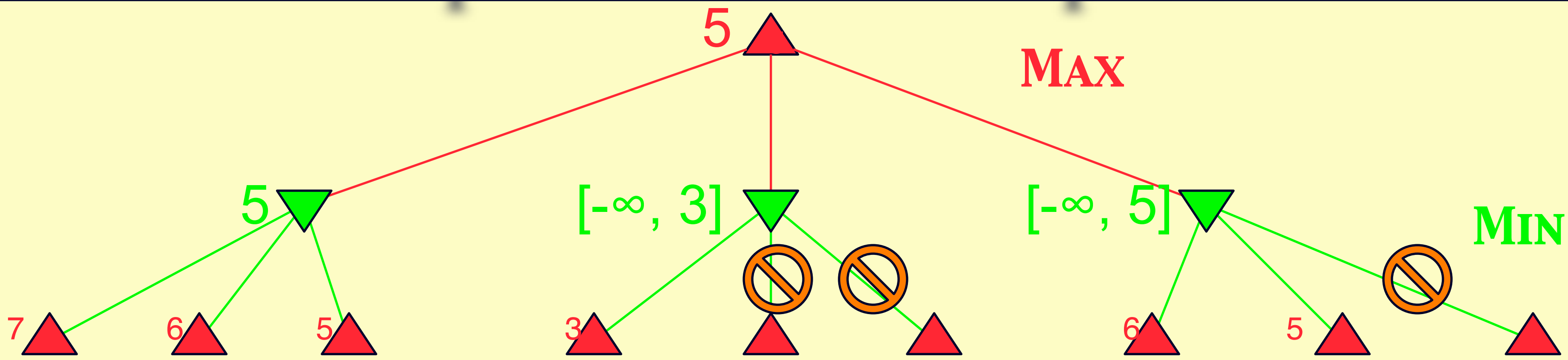


α best choice for MAX 5

β best choice for MIN 3

- MIN is lucky, and finds a value that is the same as the current worst value at this level
- MAX can choose this branch, or the other branch with the same value

Alpha-Beta Example 9



α best choice for MAX 5

β best choice for MIN 3

- MIN could continue searching this sub-tree to see if there is a value that is less than the current worst alternative in order to give MAX as few choices as possible
  - this depends on the specific implementation
- MAX knows the best value for its sub-tree

Alpha-Beta Algorithm

```
function Max-Value(state, alpha, beta) returns a utility value
  if Terminal-Test (state) then return Utility(state)
    for each s in  Successors(state) do
    alpha := Max (alpha, Min-Value(s, alpha, beta))
       if alpha  >= beta then return beta
    end
    return alpha


function Min-Value(state, alpha, beta) returns a utility value
  if Terminal-Test (state) then return Utility(state)
    for each s in  Successors(state) do
   beta := Min (beta, Max-Value(s, alpha, beta))
       if beta  <= alpha then return alpha
    end
    return beta
```

Properties of Alpha-Beta Pruning

- in the ideal case, the best successor node is examined first
  - results in $O(b^{d/2})$ nodes to be searched instead of $O(b^d)$
  - alpha-beta can look ahead twice as far as minimax
  - in practice, simple ordering functions are quite useful
- assumes an idealized tree model
  - uniform branching factor, path length
  - random distribution of leaf evaluation values
- transpositions tables can be used to store permutations
  - sequences of moves that lead to the same position
- requires additional information for good players
  - game-specific background knowledge
  - empirical data

Logistics - Oct. 30, 2012

- **AI Nugget presentations scheduled**
  - Section 1:
    - Joseph Hain: Linux MCE - Home Automation (delayed from Oct. 23)
    - William Dugger: Object Recognition
    - Erik Sandberg: Traffic Ground Truth Estimation Using Multisensor Consensus Filter
    - Daniel Gilliland: Autopilot
  - Section 3:
    - Bryan Stoll: Virtual Composer (delayed from Oct. 25)
    - Spencer Lines: What IBM's Watson has been up to since it won in 2011
    - Mathew Cabutage
    - Evolution of Robots by Darwinian Selection
- **Lab 7 Wumpus World Agent available**
  - paper-based exercise to get familiar with the Wumpus World
- **A2 Wumpus World**
  - Part 1: Knowledge Representation and Reasoning
    - Web form, no programming required
    - Due: Nov. 8
  - Part 2: Implementation
    - Due: Nov. 15
- **A3 Competitions**
  - current interest level
- **Project**
  - use feedback from mid-quarter project displays to revise project materials

Imperfect Decisions

- complete search is impractical for most games
- alternative: search the tree only to a certain depth
  - requires a cutoff-test to determine where to stop
    - replaces the terminal test
    - the nodes at that level effectively become terminal leave nodes
  - uses a heuristics-based evaluation function to estimate the expected utility of the game from those leave nodes

Evaluation Function

- determines the performance of a game-playing program
- must be consistent with the utility function
  - values for terminal nodes (or at least their order) must be the same
- tradeoff between accuracy and time cost
  - without time limits, minimax could be used
- should reflect the actual chances of winning
- frequently weighted linear functions are used
  - $E = w_1 f_1 + w_2 f_2 + \dots + w_n f_n$
  - combination of features, weighted by their relevance

Example: Tic-Tac-Toe

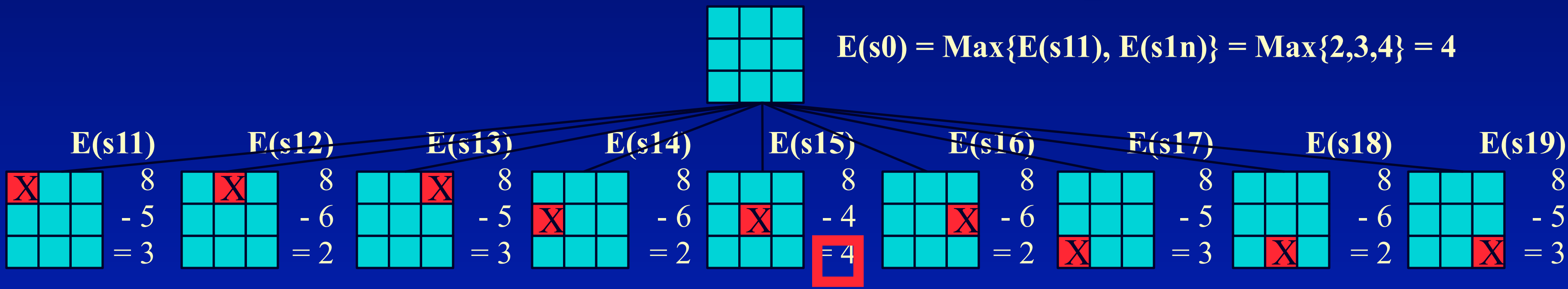
- simple evaluation function

```
E(s) = (rx + cx + dx) - (ro + co + do)
```

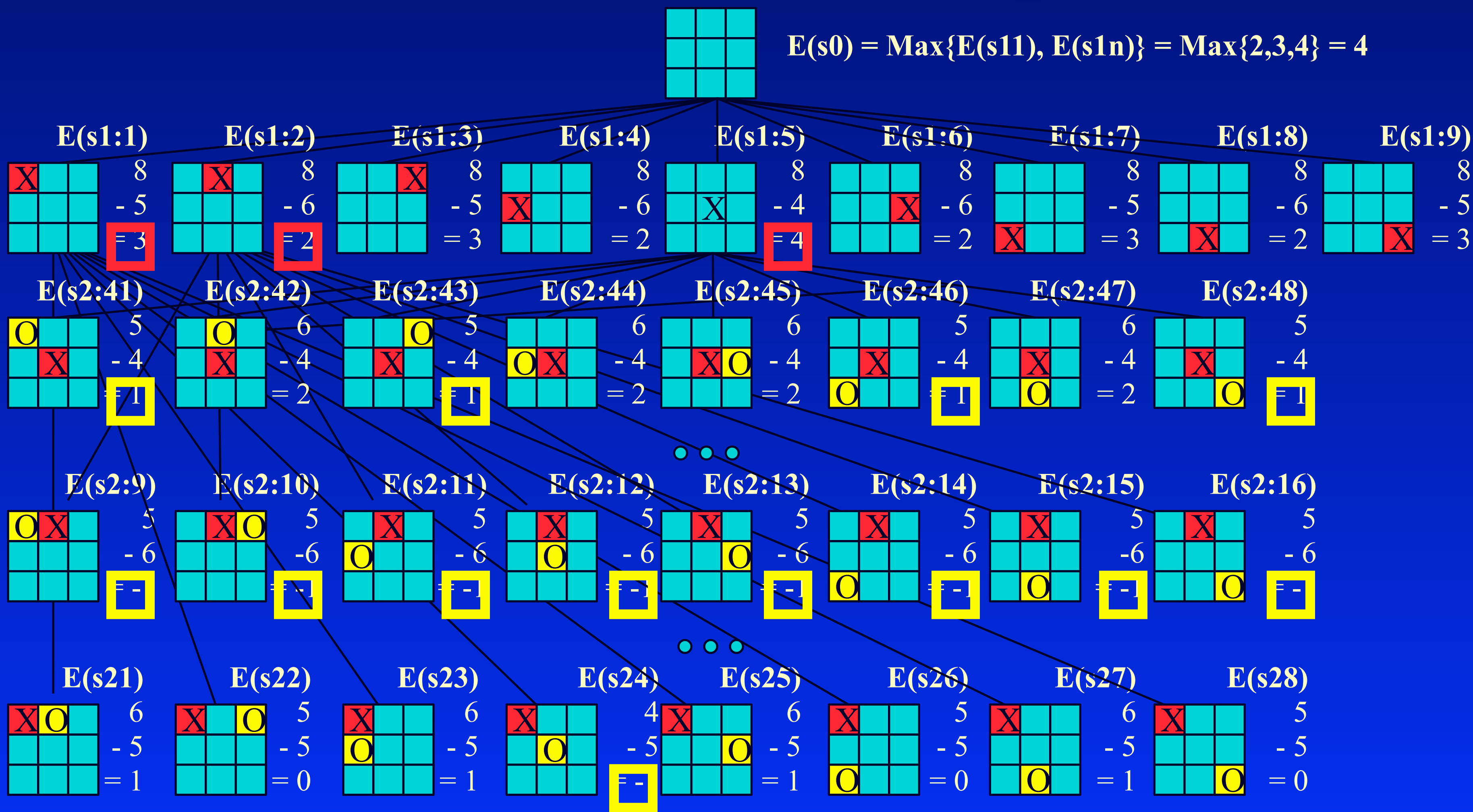
where r,c,d are the numbers of row, column and diagonal lines still available; x and o are the pieces of the two players

- 1-ply lookahead
  - start at the top of the tree
  - evaluate all 9 choices for player 1
  - pick the maximum E-value
- 2-ply lookahead
  - also looks at the opponents possible move
    - assuming that the opponents picks the minimum E-value

Tic-Tac-Toe 1-Ply

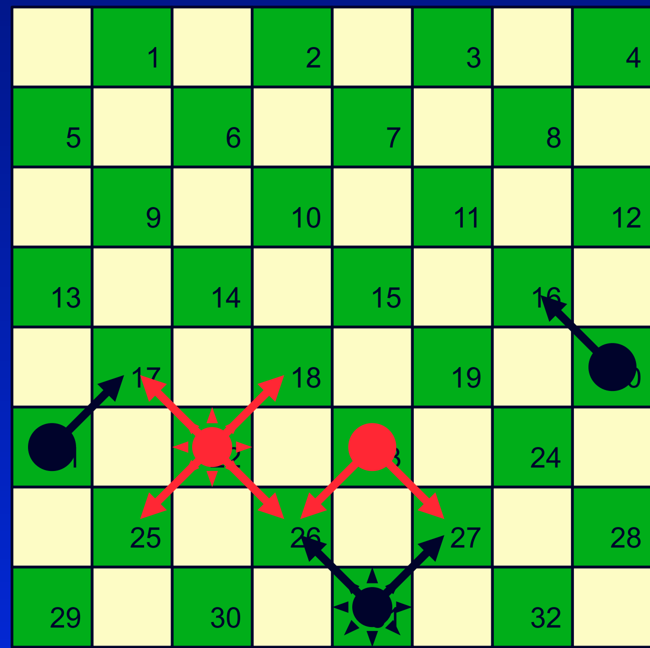


Tic-Tac-Toe 2-Ply

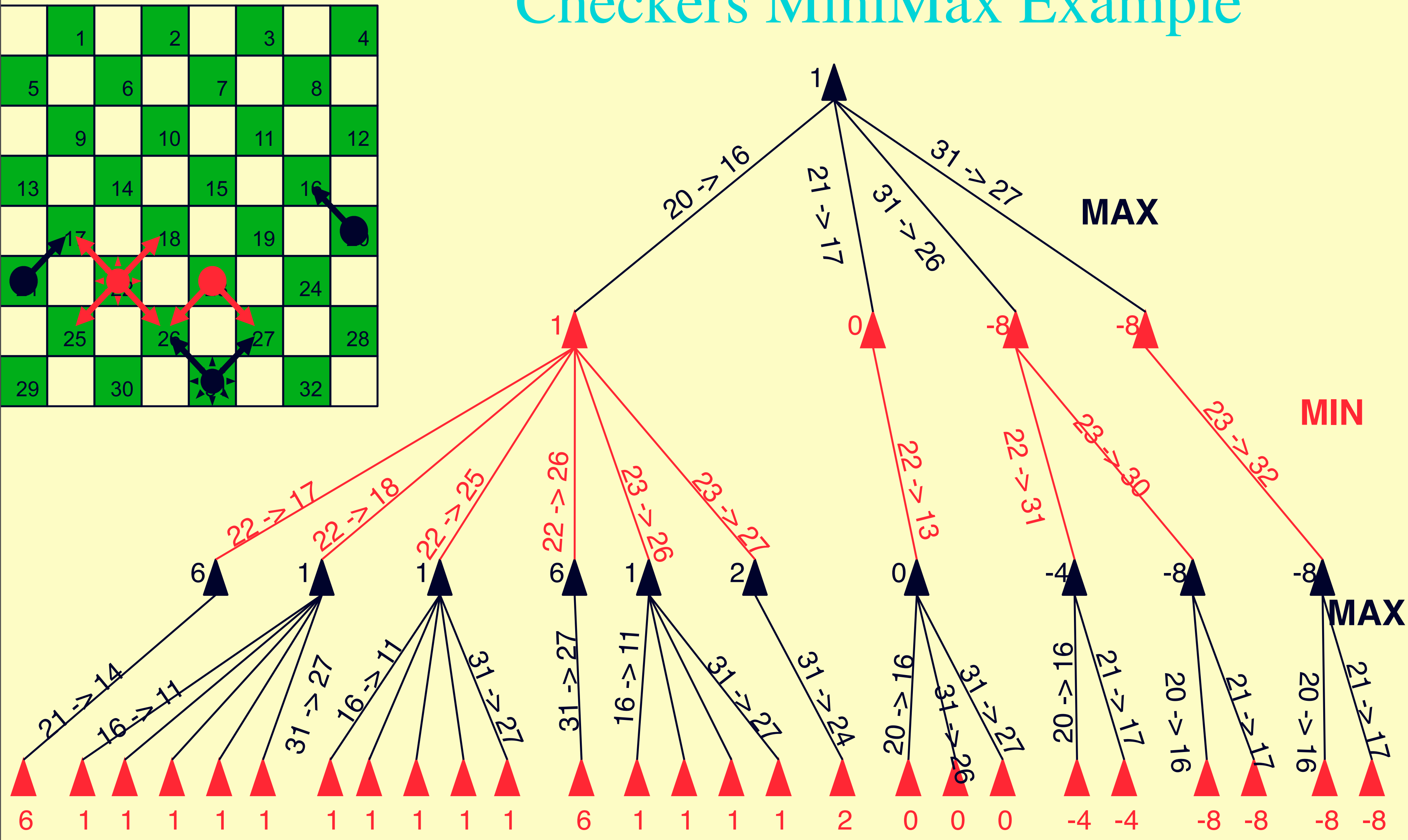


Checkers Case Study

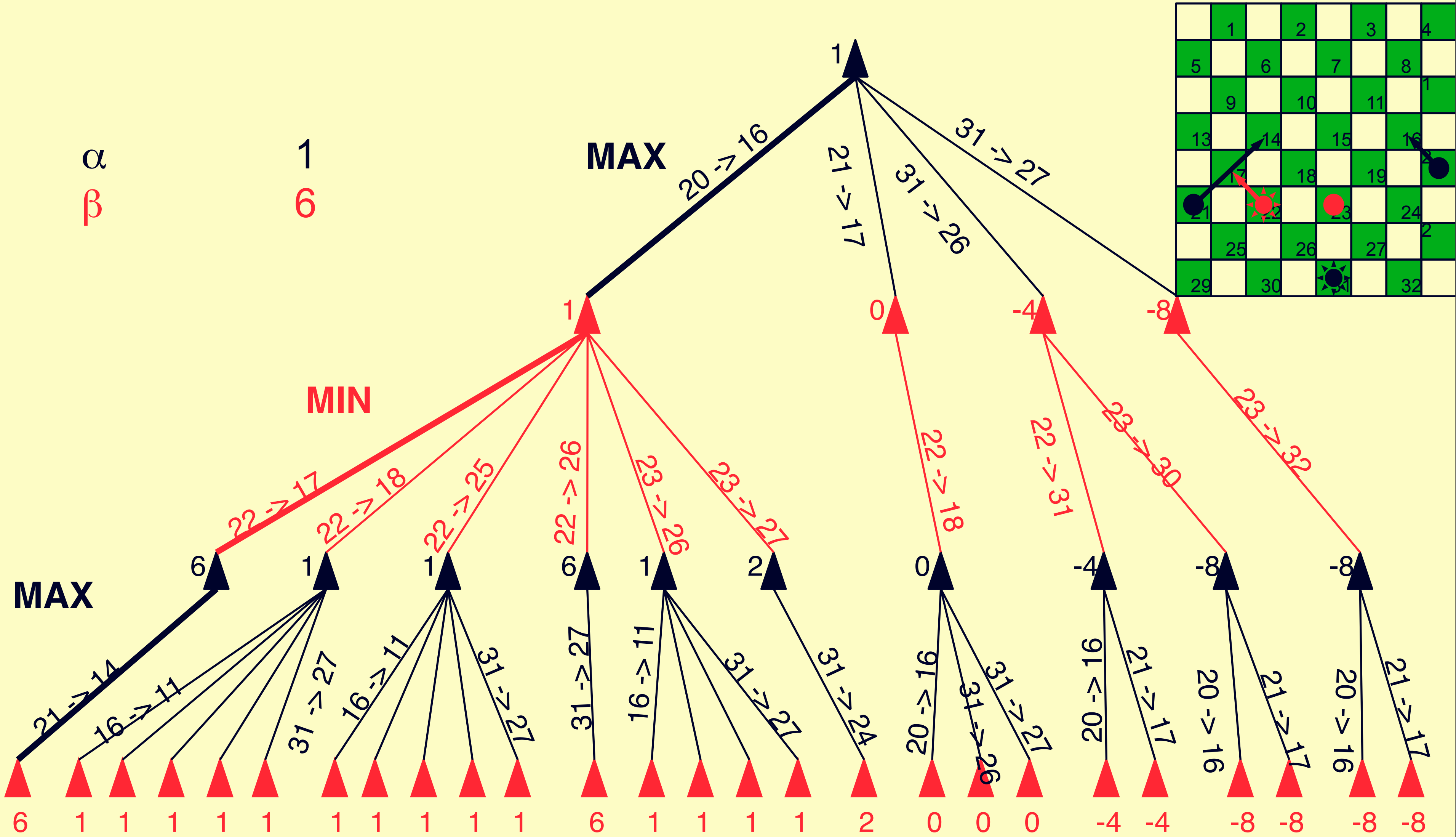
- initial board configuration
  - **BLACK** single on 20 single on 21 king on 31
  - **RED** single on 23 king on 22
  - evaluation function
    $E(s) = (5\,x_1 + x_2) - (5r_1 + r_2)$
    where
    x_1 = black king advantage,
    x_2 = black single advantage,
    r_1 = red king advantage,
    r_2 = red single advantage



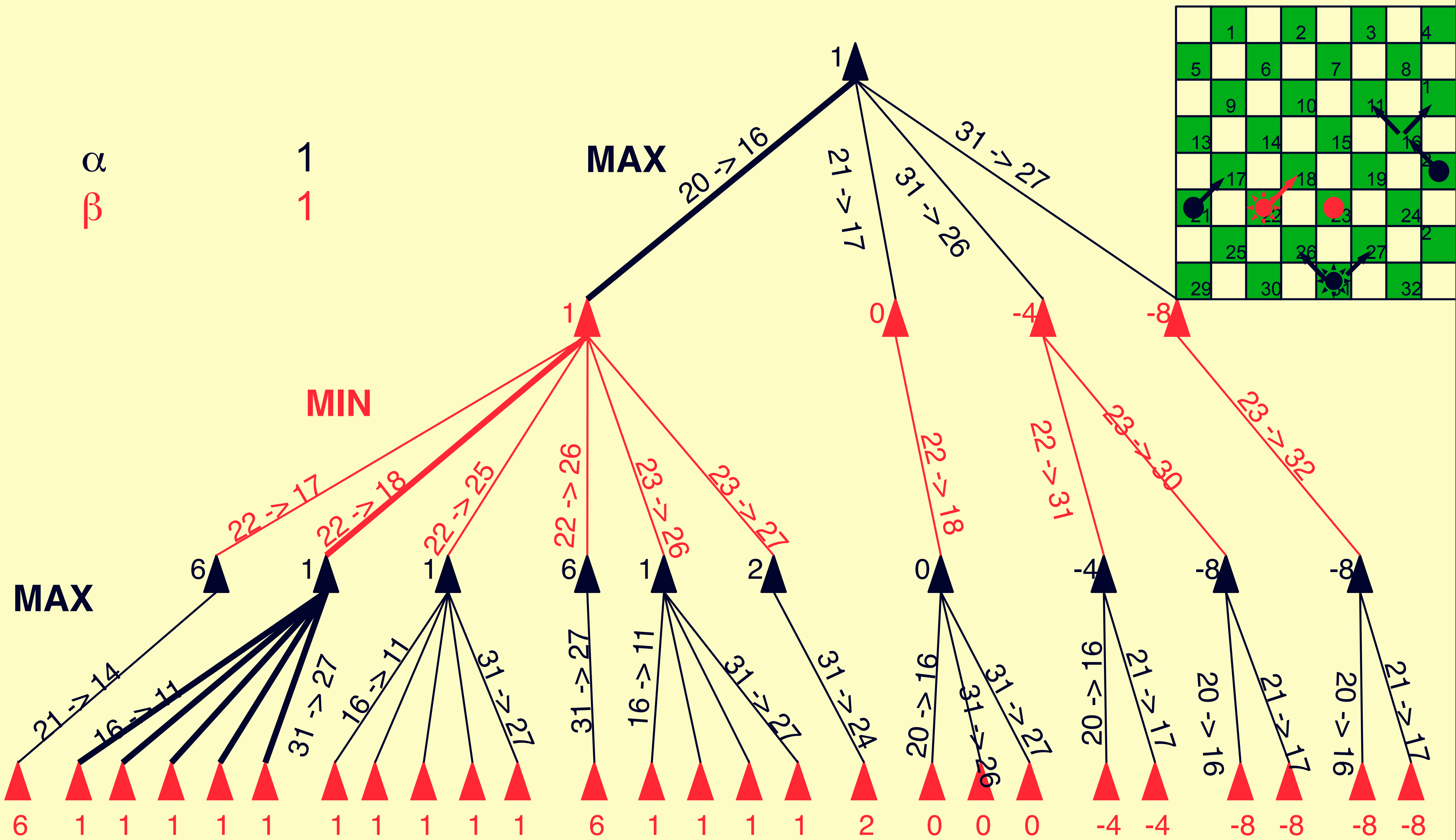
Checkers MiniMax Example



Checkers Alpha-Beta Example



Checkers Alpha-Beta Example

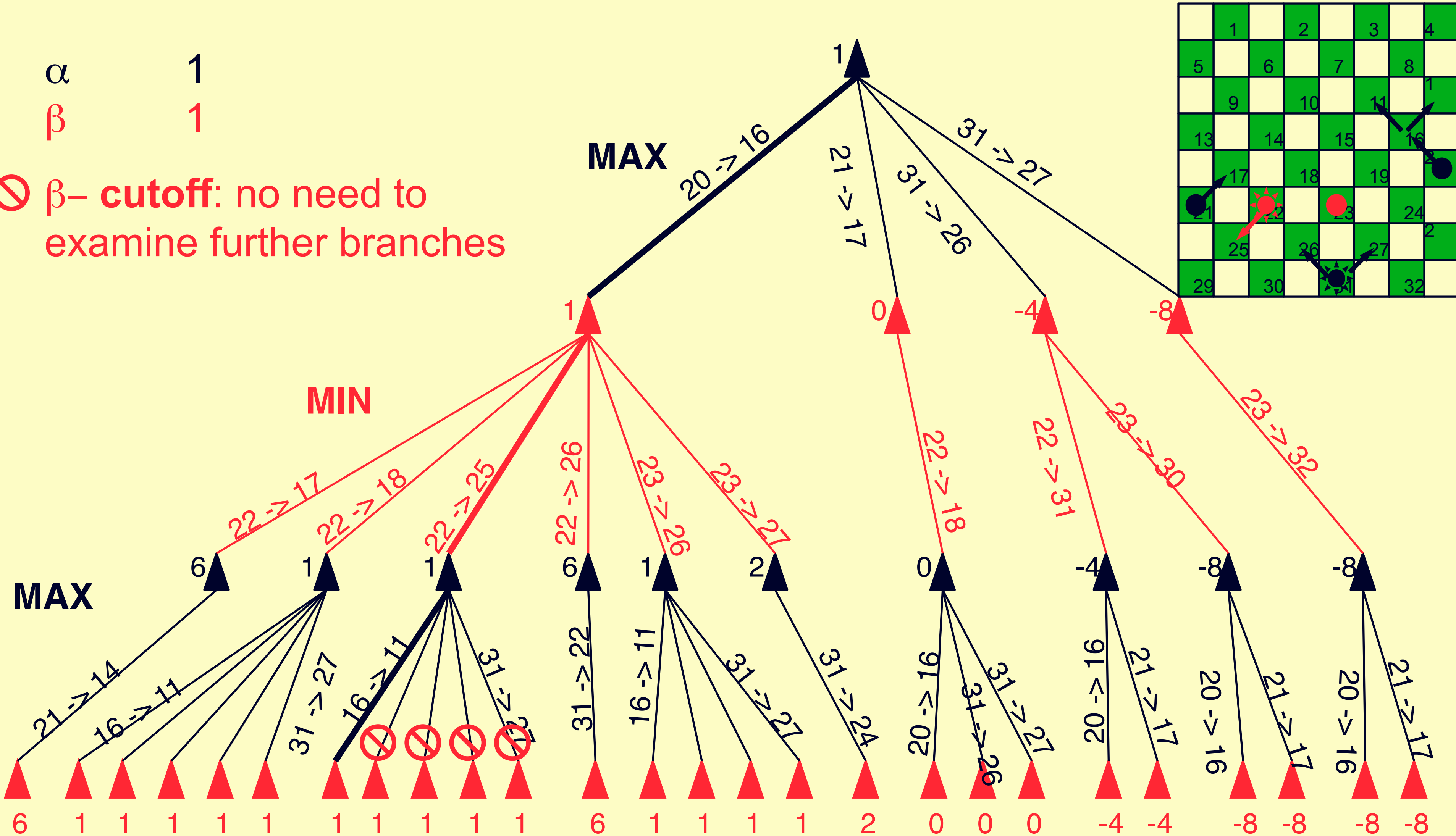


Checkers Alpha-Beta Example

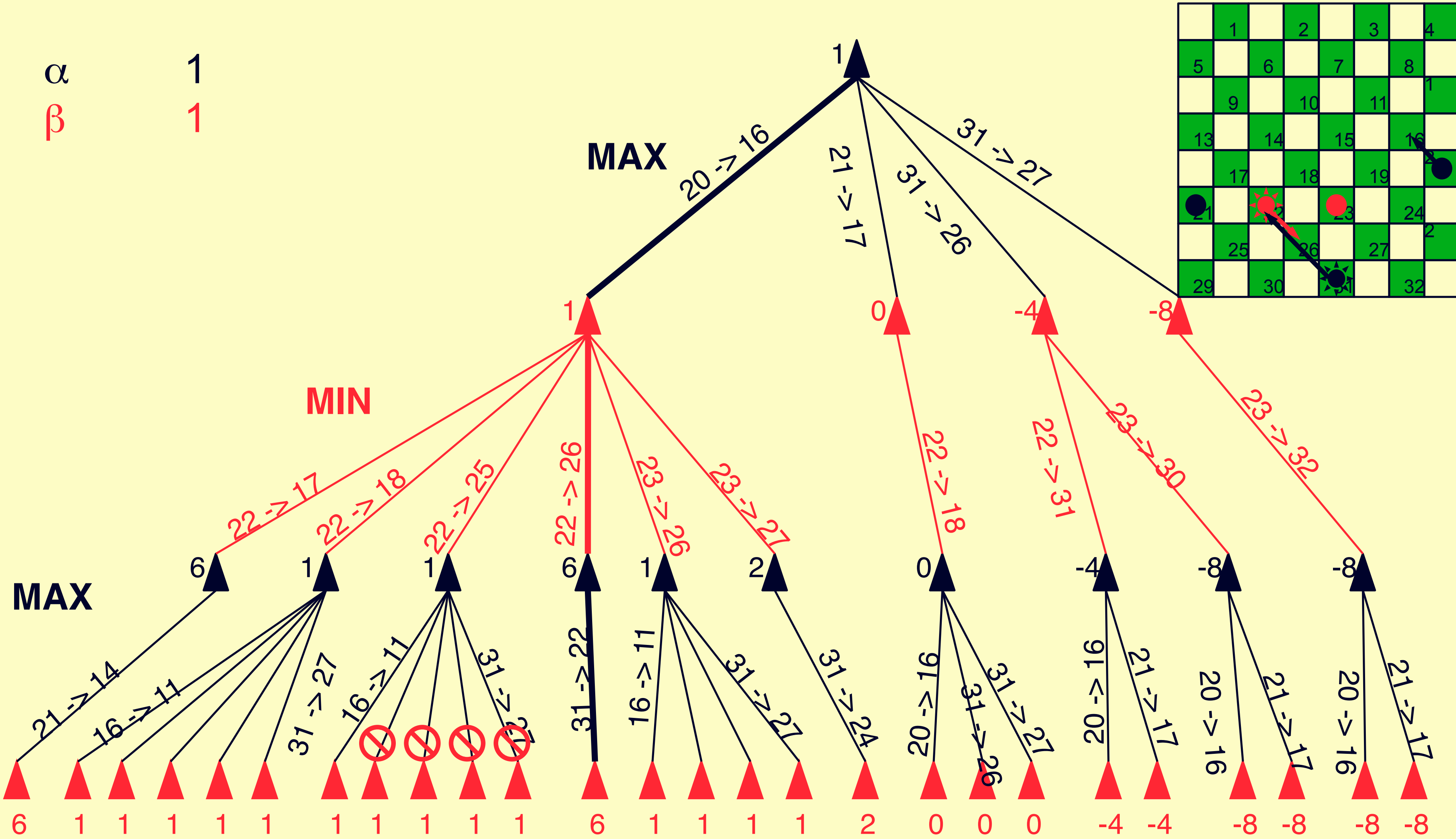
α 1

β 1

β– **cutoff**: no need to examine further branches



Checkers Alpha-Beta Example

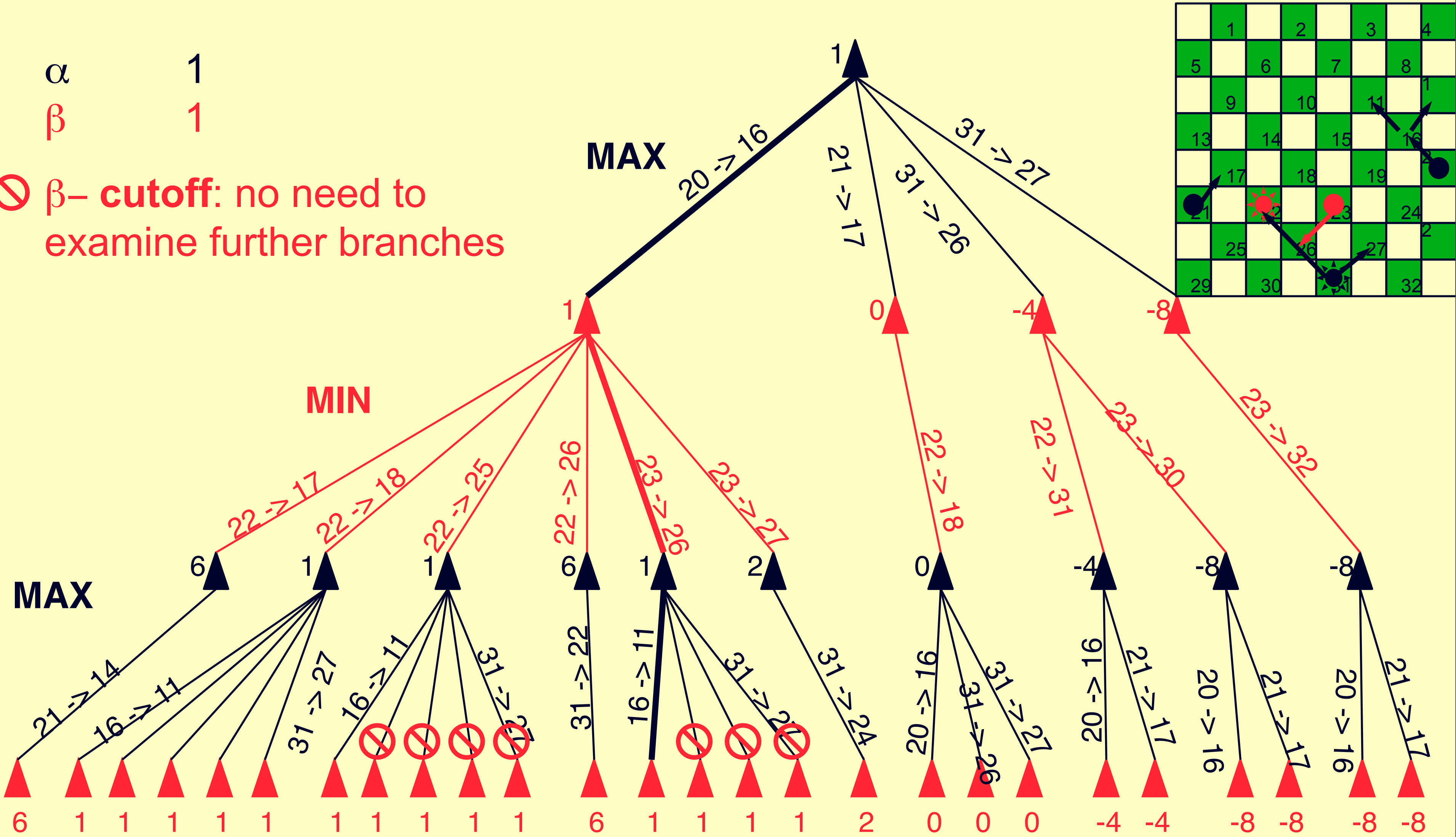


Checkers Alpha-Beta Example

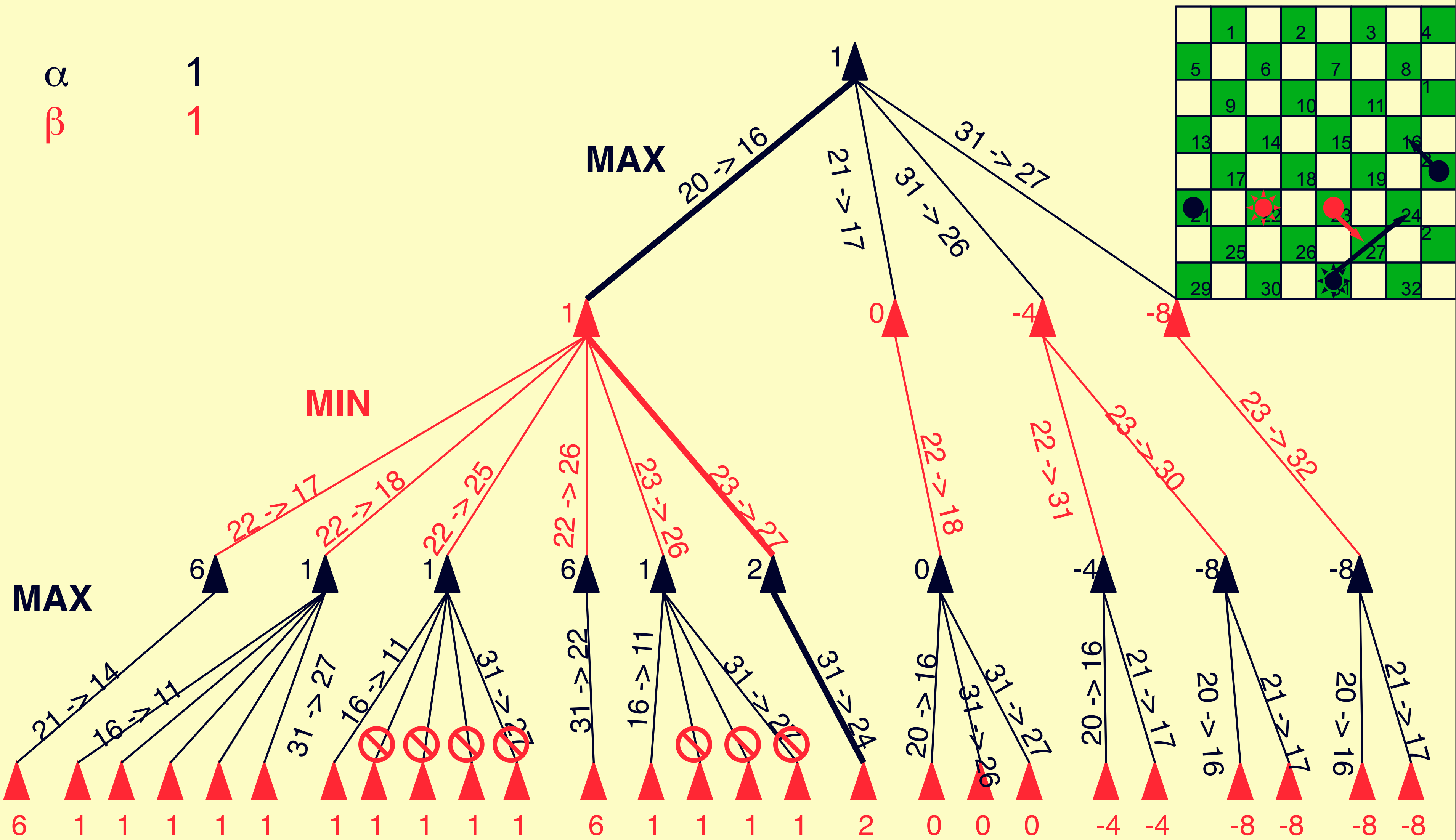
α 1

β 1

β– **cutoff**: no need to examine further branches

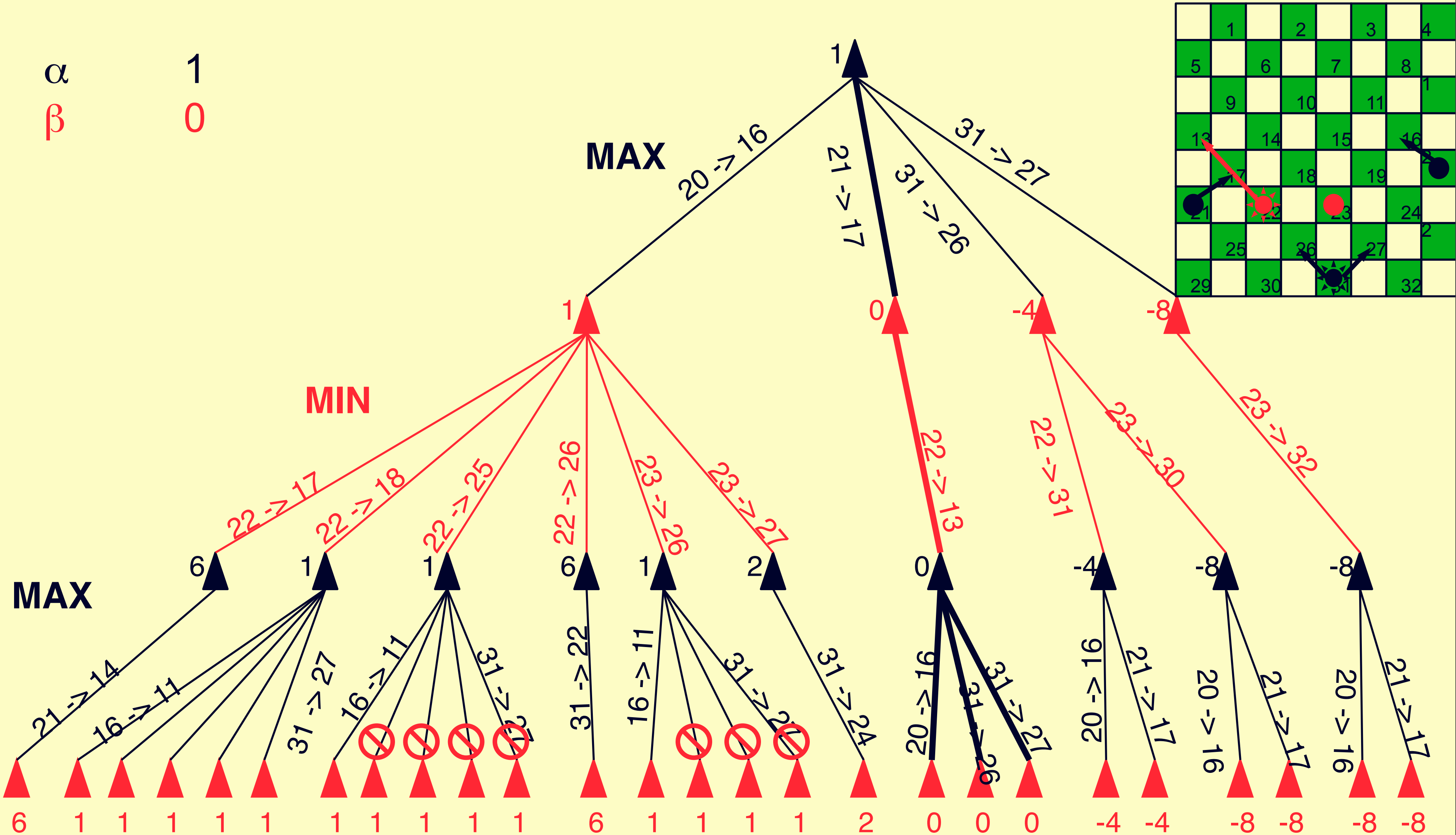


Checkers Alpha-Beta Example



Checkers Alpha-Beta Example

α 1
β 0

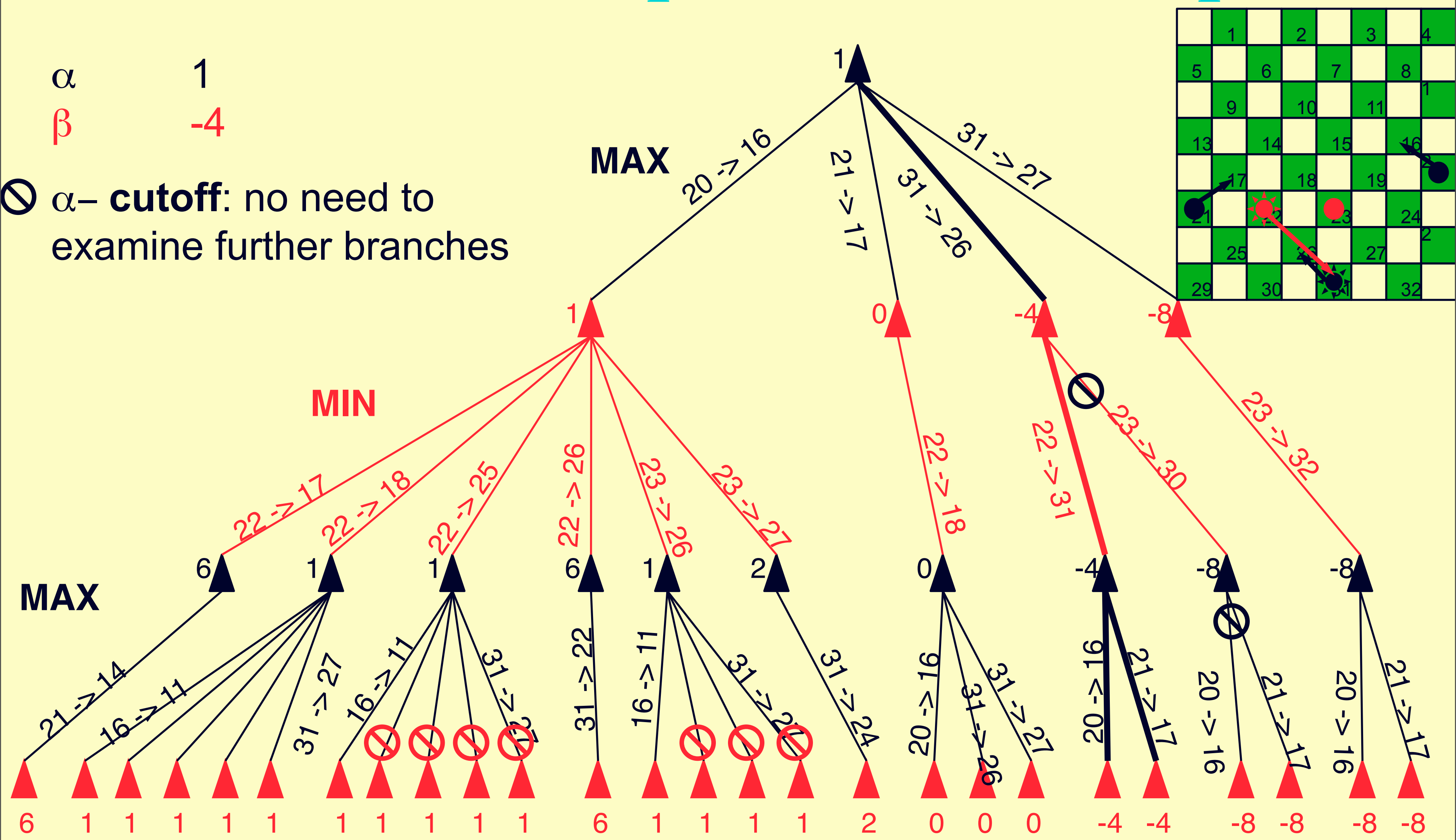


Checkers Alpha-Beta Example

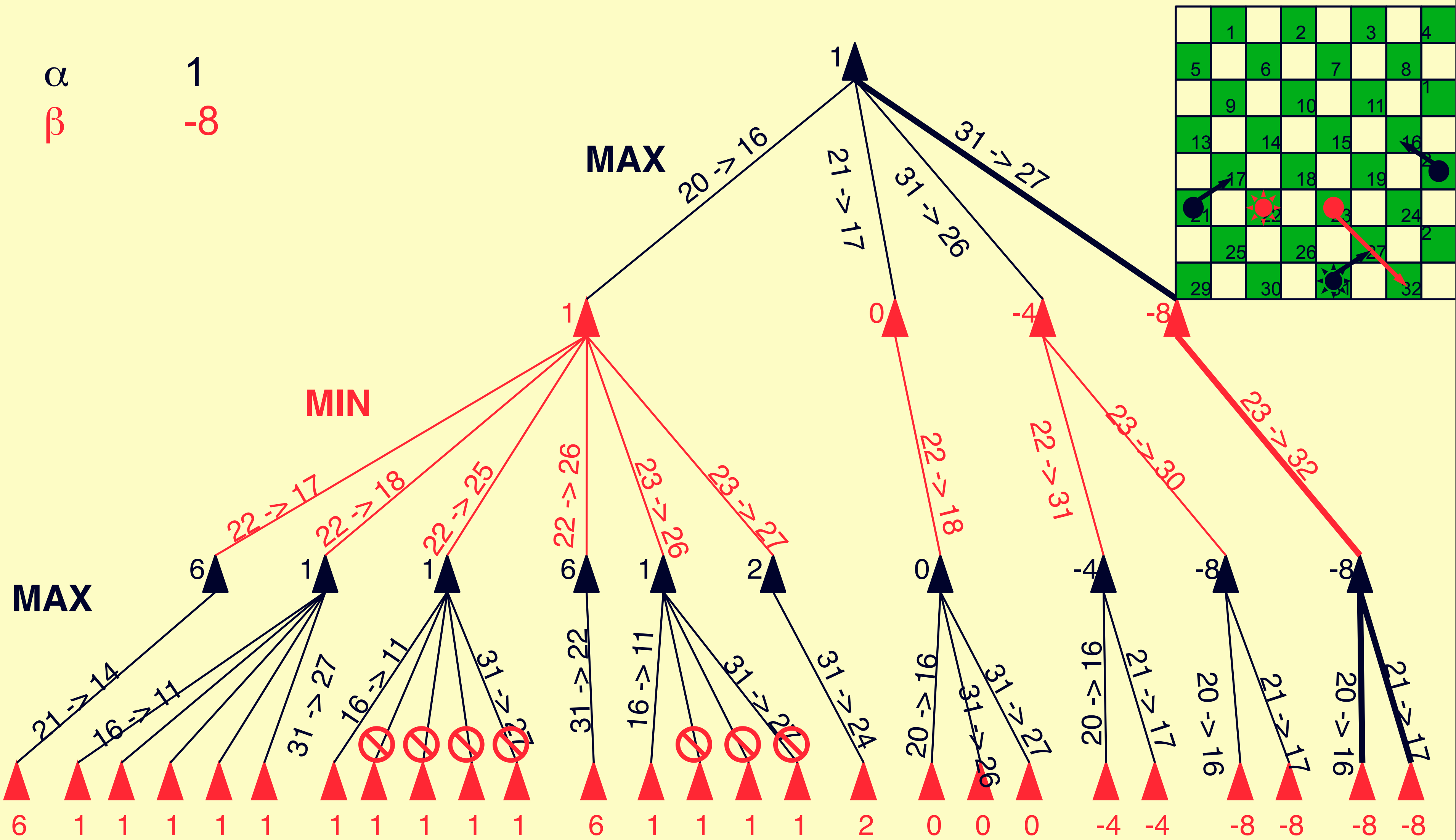
α 1

β -4

⃠ α– **cutoff**: no need to examine further branches



Checkers Alpha-Beta Example



Search Limits

- search must be cut off because of time or space limitations
- strategies like depth-limited or iterative deepening search can be used
  - don’t take advantage of knowledge about the problem
- more refined strategies apply background knowledge
  - quiescent search
    - cut off only parts of the search space that don’t exhibit big changes in the evaluation function

Horizon Problem

- moves may have disastrous consequences in the future, but the consequences are not visible
  - the corresponding change in the evaluation function will only become evident at deeper levels
    - they are “beyond the horizon”
- determining the horizon is an open problem without a general solution
  - only some pragmatic approaches restricted to specific games or situation

Games with Chance

- in many games, there is a degree of unpredictability through random elements
  - throwing dice, card distribution, roulette wheel, …
- this requires *chance nodes* in addition to the MAX and MIN nodes
  - branches indicate possible variations
  - each branch indicates the outcome and its likelihood

Rolling Dice

- 36 ways to roll two dice
  - the same likelihood for all of them
  - due to symmetry, there are only 21 distinct rolls
  - six doubles have a 1/36 chance
  - the other fifteen have a 1/18 chance

Decisions with Chance

- the utility value of a position depends on the random element
  - the definite minimax value must be replaced by an expected value
- calculation of expected values
  - utility function for terminal nodes
  - for all other nodes
    - calculate the utility for each chance event
    - weigh by the chance that the event occurs
    - add up the individual utilities

Expectiminimax Algorithm

- calculates the utility function for a particular position based on the outcome of chance events
- utilizes an additional pair of functions to assess the utility values of chance nodes

$$expectimin(C) = \Sigma_I\ P(d_i)\ min_{s \in S(C,di)}(utility(s))$$

$$expectimax(C) = \Sigma_I\ P(d_i)\ max_{s \in S(C,di)}(utility(s))$$

where *C* are chance nodes,

$P(d_i)$ is the probability of a chance event d_i, and $S(C,d_i)$ the set of positions resulting from the event d_i, occurring at position C

Limiting Search with Chance

- similar to alpha-beta pruning for minimax
  - search is cut off
  - evaluation function is used to estimate the value of a position
  - must put boundaries on possible values of the utility function
- somewhat more restricted
  - the evaluation function is influenced by some aspects of the chance events

Properties of Expectiminimax

- complexity of $O(b^m n^m)$
  - n - number of distinct chance events
  - b - branching factor
  - m - maximum path length (number of moves in the game)
- example backgammon:
  - $n = 21$, $b \approx 20$ (but may be as high as 4000)

Games and Computers

- state of the art for some game programs
  - Chess
  - Checkers
  - Othello
  - Backgammon
  - Go

Chess

- Deep Blue, a special-purpose parallel computer, defeated the world champion Gary Kasparov in 1997
  - the human player didn't show his best game
    - some claims that the circumstances were questionable
  - Deep Blue used a massive data base with games from the literature
- Fritz, a program running on an ordinary PC, challenged the world champion Vladimir Kramnik to an eight-game draw in 2002
  - top programs and top human players are roughly equal

Checkers

- Arthur Samuel develops a checkers program in the 1950s that learns its own evaluation function
  - reaches an expert level stage in the 1960s
- Chinook becomes world champion in 1994
  - human opponent, Dr. Marion Tinsley, withdraws for health reasons
    - Tinsley had been the world champion for 40 years
  - Chinook uses off-the-shelf hardware, alpha-beta search, end-games data base for six-piece positions

Othello

- Logistello defeated the human world champion in 1997
- many programs play far better than humans
  - smaller search space than chess
  - little evaluation expertise available

Backgammon

- TD-Gammon, neural-network based program, ranks among the best players in the world
  - improves its own evaluation function through learning techniques
  - search-based methods are practically hopeless
    - chance elements, branching factor

Go

- humans play far better
  - large branching factor (around 360)
    - search-based methods are hopeless
- rule-based systems play at amateur level
- the use of pattern-matching techniques can improve the capabilities of programs
  - difficult to integrate
- $2,000,000 prize for the first program to defeat a top-level player

Jeopardy

- in 2010, IBM announced that its Watson system will participate in a Jeopardy contest
- Watson beat two of the best Jeopardy participants

Beyond Search?

- search-based game playing strategies have some inherent limitations
  - high computational overhead
  - exploration of uninteresting areas of the search space
  - complicated heuristics
- utility of node expansion
  - consider the trade-off between the costs for calculations, and the improvement in traversing the search space
- goal-based reasoning and planning
  - concentrate on possibly distant, but *critical* states instead of complete paths with lots of intermediate states
- meta-reasoning
  - observe the reasoning process itself, and try to improve it
  - alpha-beta pruning is a simple instance

Important Concepts and Terms

- action
- alpha-beta pruning
- Backgammon
- chance node
- Checkers
- Chess
- contingency problem
- evaluation function
- expectiminimax algorithm
- Go
- heuristic
- horizon problem
- initial state
- minimax algorithm
- move
- operator
- Othello
- ply
- pruning
- quiescent
- search
- search tree
- state
- strategy
- successor
- terminal state
- utility function

Chapter Summary

- many game techniques are derived from search methods
- the minimax algorithm determines the best move for a player by calculating the complete game tree
- alpha-beta pruning dismisses parts of the search tree that are provably irrelevant
- an evaluation function gives an estimate of the utility of a state when a complete search is impractical
- chance events can be incorporated into the minimax algorithm by considering the weighted probabilities of chance events