Foundations of Artificial Intelligence

G2. Board Games: Formal Definition and Minimax Search

Malte Helmert

University of Basel

May 11, 2026

Board Games: Overview

chapter overview:

- G1. Introduction and State of the Art
- G2. Formal Definition and Minimax Search
- G3. Evaluation Functions
- G4. Alpha-Beta Search
- G5. Stochastic Games
- G6. Monte-Carlo Tree Search Framework
- G7. Monte-Carlo Tree Search Variants

Games

Example: Chess

Example (Chess)

- positions described by:
  - configuration of pieces
  - whose turn it is
  - en-passant and castling rights
- turns alternate
- terminal positions: checkmate and stalemate positions
- utility of terminal position for first player (white):
  - $+1$ if black is checkmated
  - 0 if stalemate position
  - -1 if white is checkmated

Terminology Compared to State-Space Search

Many concepts for board games are similar to state-space search.
Terminology differs, but is often in close correspondence:

- state $\rightsquigarrow$ position
- goal state $\rightsquigarrow$ terminal position
- action $\rightsquigarrow$ move
- search tree $\rightsquigarrow$ game tree

Definition

Definition (game)

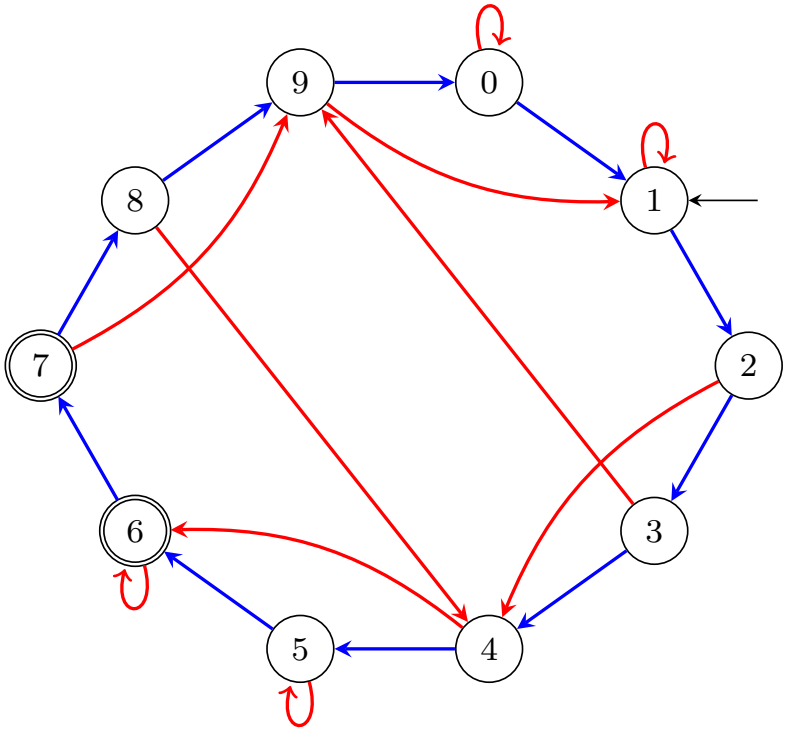
A game is a 7-tuple $\mathcal{S} = \langle S, A, T, s_I, S_G, \mathit{utility}, \mathit{player} \rangle$ with

- finite set of positions S
- finite set of moves A
- deterministic transition relation $T \subseteq S \times A \times S$
- initial position $s_I \in S$
- set of terminal positions $S_G \subseteq S$
- utility function $\mathit{utility} : S_G \to \mathbb{R}$
- player function $\mathit{player} : S \setminus S_G \to \{\mathrm{MAX}, \mathrm{MIN}\}$

Reminder: Bounded Inc-and-Square Search Problem

informal description:

- find a sequence of
  - increment-mod10 (*inc*) and
  - square-mod10 (*sqr*) actions
- on the natural numbers from 0 to 9
- that reaches the number 6 or 7
- starting from the number 1
- assuming each action costs 1.



Running Example: Bounded Inc-and-Square Game

informal description:

- Players alternatingly apply a
  - increment-mod10 (*inc*) or
  - square-mod10 (*sqr*) move
- on the natural numbers from 0 to 9
- starting from the number 1;
- if the game reaches the number 6 or 7
- or after a fixed number of 4 moves
- MAX obtains utility u (MIN: $-u$)
  where u is the current number.

Running Example: Bounded Inc-and-Square Game

informal description:

- Players alternatingly apply a
  - increment-mod10 (*inc*) or
  - square-mod10 (*sqr*) move
- on the natural numbers from 0 to 9
- starting from the number 1;
- if the game reaches the number 6 or 7
- or after a fixed number of 4 moves
- MAX obtains utility u (MIN: $-u$) where u is the current number.

formal model:

- $S = \{s_i^k \mid 0 \leq i \leq 9, 0 \leq k \leq 4\}$
- $A = \{inc, sqr\}$
- for $0 \leq i \leq 9$ and $0 \leq k < 4$:
  - $\langle s_i^k, inc, s^{k+1}_{(i+1) \bmod 10} \rangle \in T$
  - $\langle s_i^k, sqr, s^{k+1}_{i^2 \bmod 10} \rangle \in T$
- $s_{\mathsf{I}} = s_1^0$
- $S_{\mathsf{G}} = \{s_i^k \mid i \in \{6,7\} \vee k = 4\}$
- $utility(s_i^k) = i$ for all $s_i^k \in S_{\mathsf{G}}$
- $player(s_i^k) =$ MAX if k even and $player(s_i^k) =$ MIN otherwise

Why are Board Games Difficult?

As in classical search problems, the number of positions of (interesting) board games is huge:

- Chess: roughly 10^{40} reachable positions;
  game with 50 moves/player and branching factor 35:
  tree size roughly $35^{100} \approx 10^{154}$
- Go: more than 10^{100} positions;
  game with roughly 300 moves and branching factor 200:
  tree size roughly $200^{300} \approx 10^{690}$

In addition, it is not sufficient to find a solution path:

- We need a strategy reacting to all possible opponent moves.
- Usually, such a strategy is implemented as an algorithm that provides the next move on the fly (i.e., not precomputed).

Strategies

Definition (strategy, partial strategy)

Let $\mathcal{S} = \langle S, A, T, s_I, S_G, \mathit{utility}, \mathit{player} \rangle$ be a game and let $S_{\mathrm{MAX}} = \{s \in S \mid \mathit{player}(s) = \mathrm{MAX}\}$.
A partial strategy for player MAX is a function

$$\pi : S'_{\mathrm{MAX}} \mapsto A$$

with $S'_{\mathrm{MAX}} \subseteq S_{\mathrm{MAX}}$ and $\pi(s) = a$ implies that a is applicable in s.

If $S'_{\mathrm{MAX}} = S_{\mathrm{MAX}}$, then π is also called total strategy (or strategy).

We always take the viewpoint of MAX, but S_{MIN} and a (partial/total) strategy for MIN are defined accordingly.

Specific vs. General Algorithms

- We consider approaches that must be tailored to a specific board game for good performance, e.g., by using a suitable evaluation function.

⇝ see chapters on informed search methods

- Analogously to the generalization of search methods to declaratively described problems (automated planning), board games can be considered in a more general setting, where game rules (state spaces) are part of the input.

⇝ general game playing: regular competitions since 2005

Algorithms for Board Games

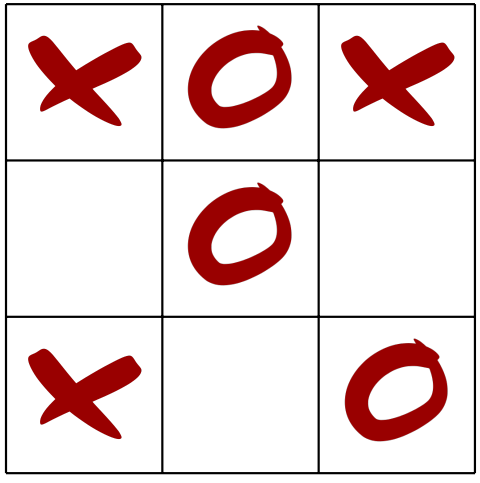
properties of good algorithms for board games:

- look ahead as far as possible (deep search)
- consider only interesting parts of the game tree
  (selective search, analogously to heuristic search algorithms)
- evaluate current position as accurately as possible
  (evaluation functions, analogously to heuristics)

Minimax Search

Example: Tic-Tac-Toe

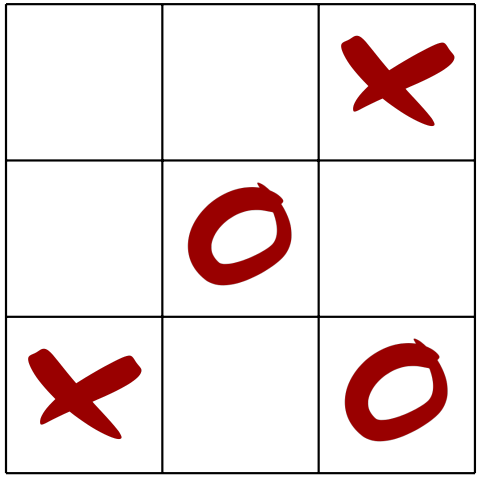
consider it's the turn of player ✕:



If the utility for win/draw/lose for player ✕ is +1/0/-1,
what is an appropriate utility value for the depicted position?

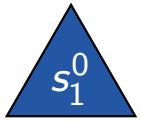
Example: Tic-Tac-Toe

consider it's the turn of player ✕:



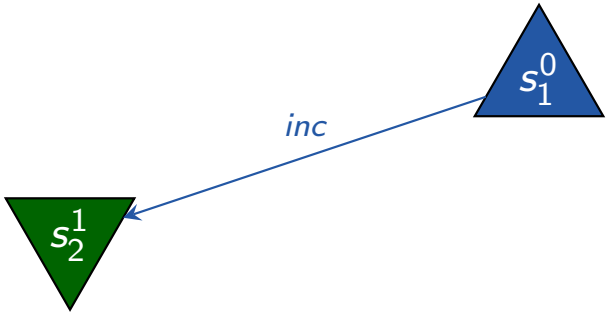
And what about this one?

Idea and Example



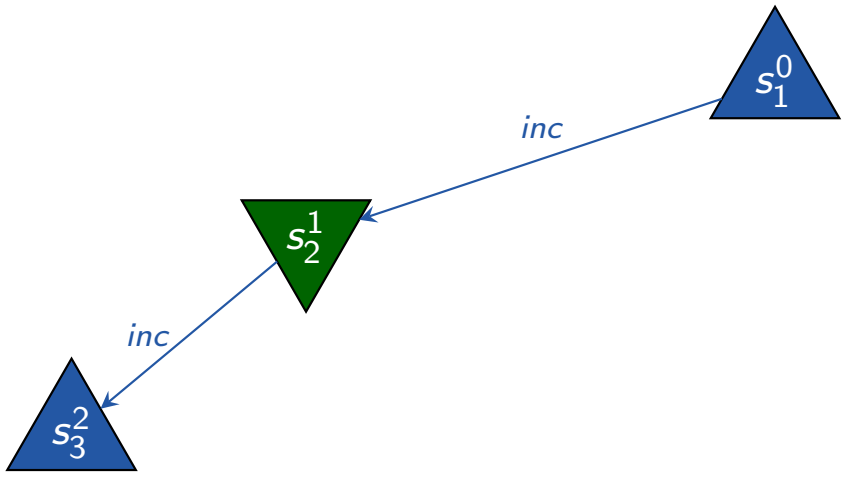
- depth-first search in game tree

Idea and Example



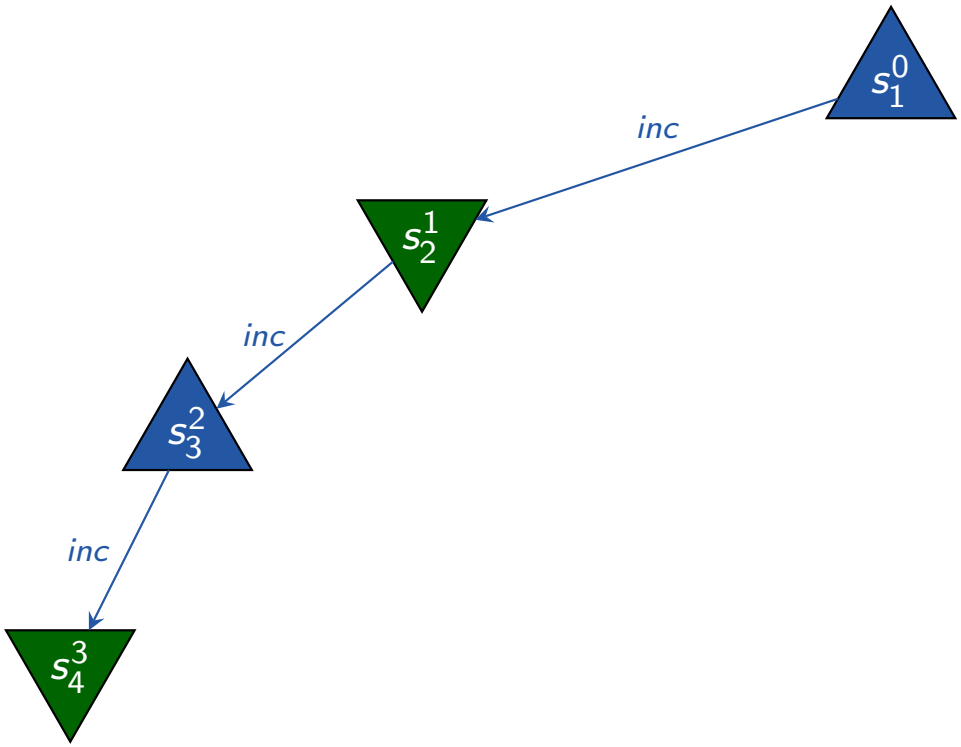
- depth-first search in game tree

Idea and Example



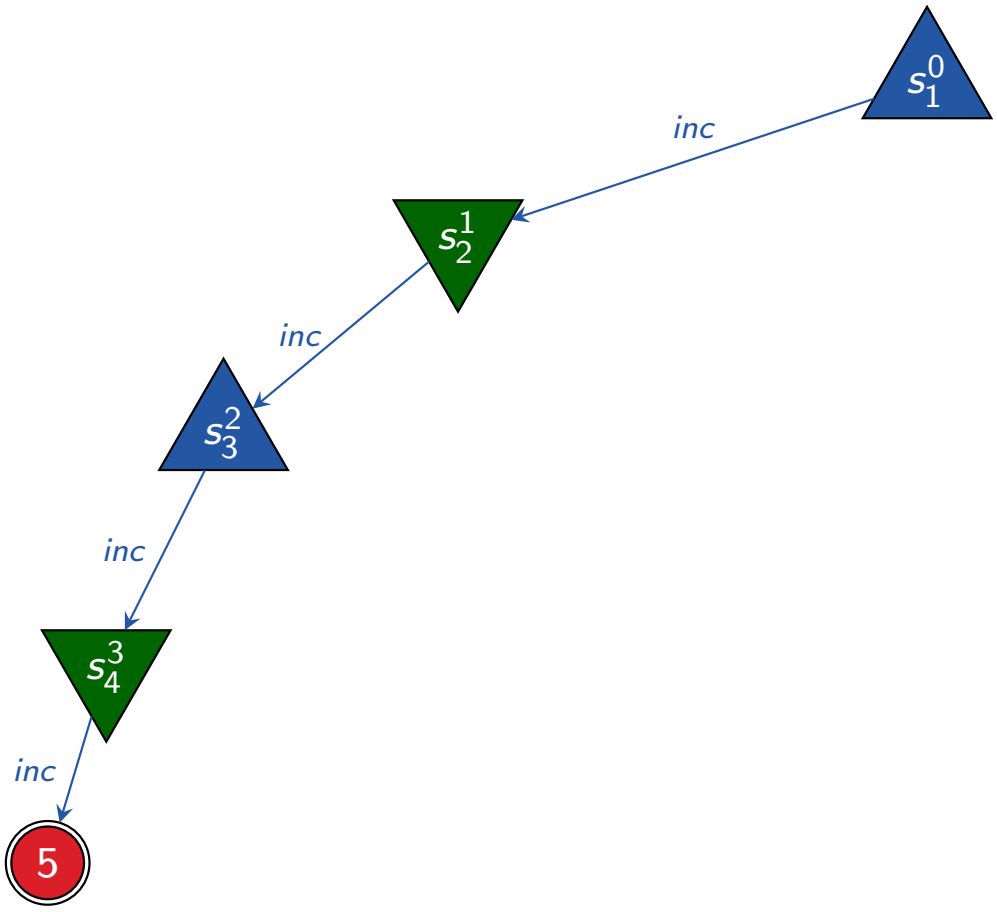
- depth-first search in game tree

Idea and Example



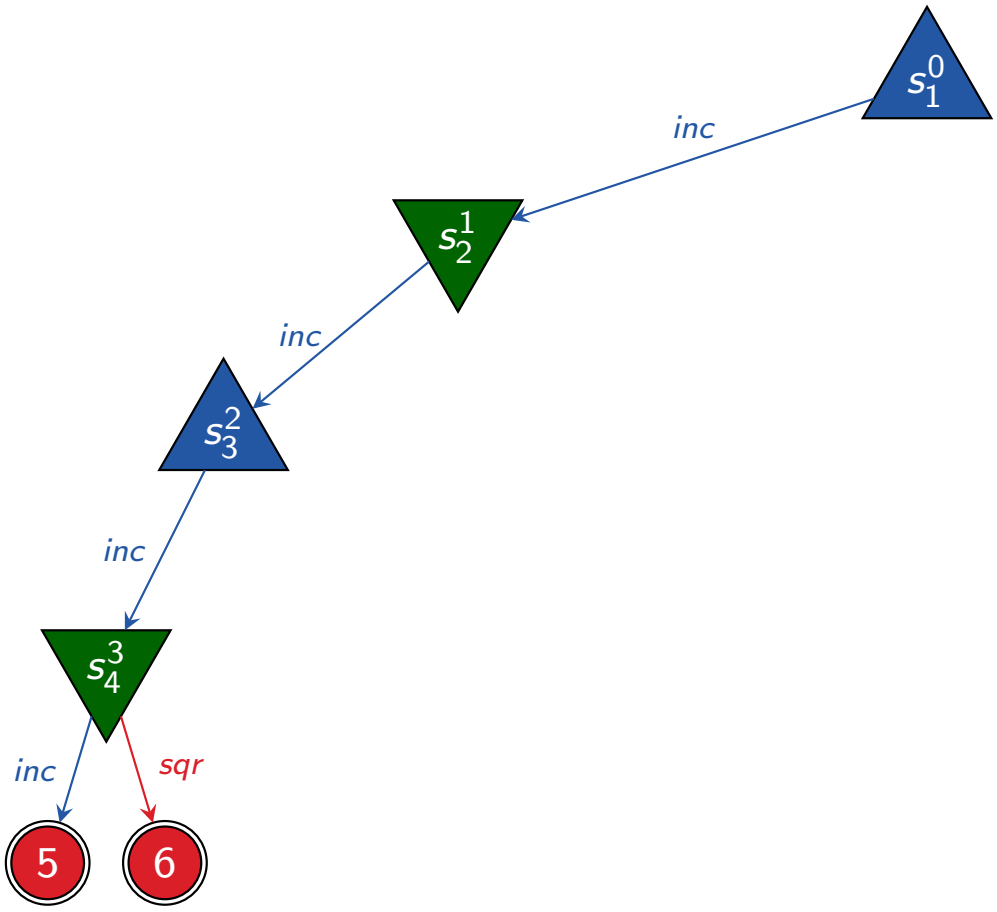
- depth-first search in game tree

Idea and Example



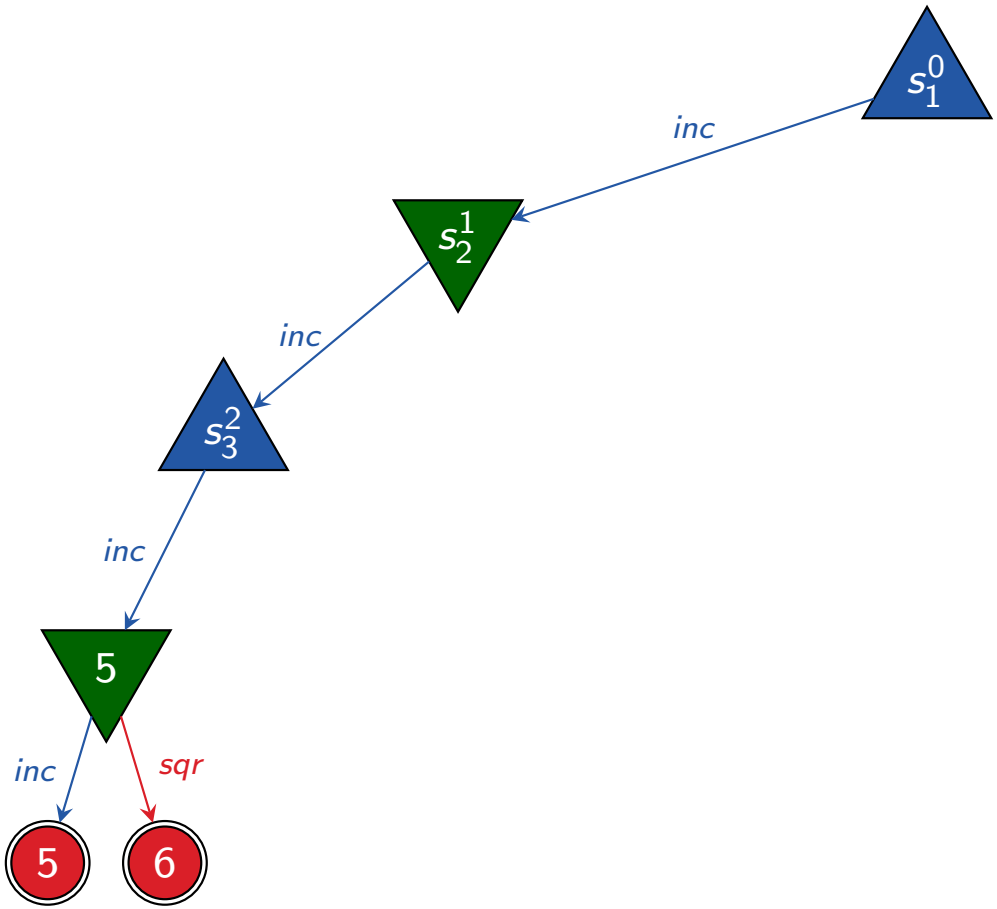
- depth-first search in game tree
- determine utility value of terminal position with utility function

Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function

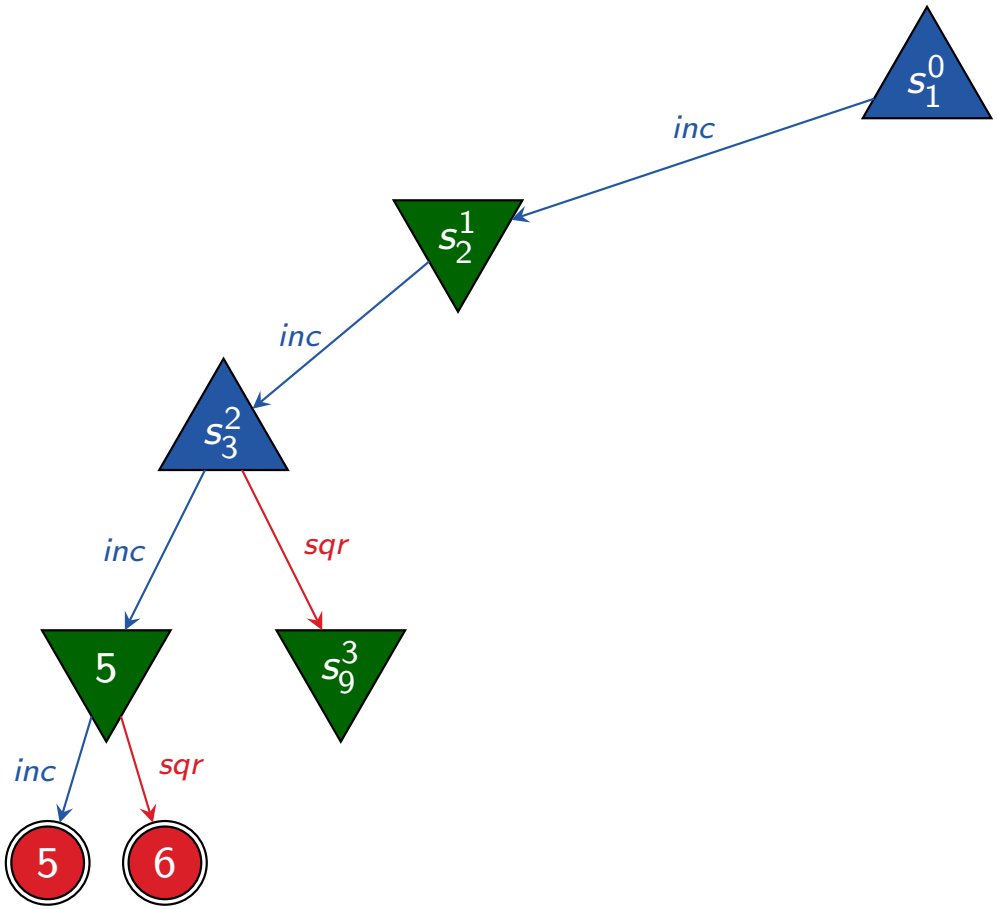
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

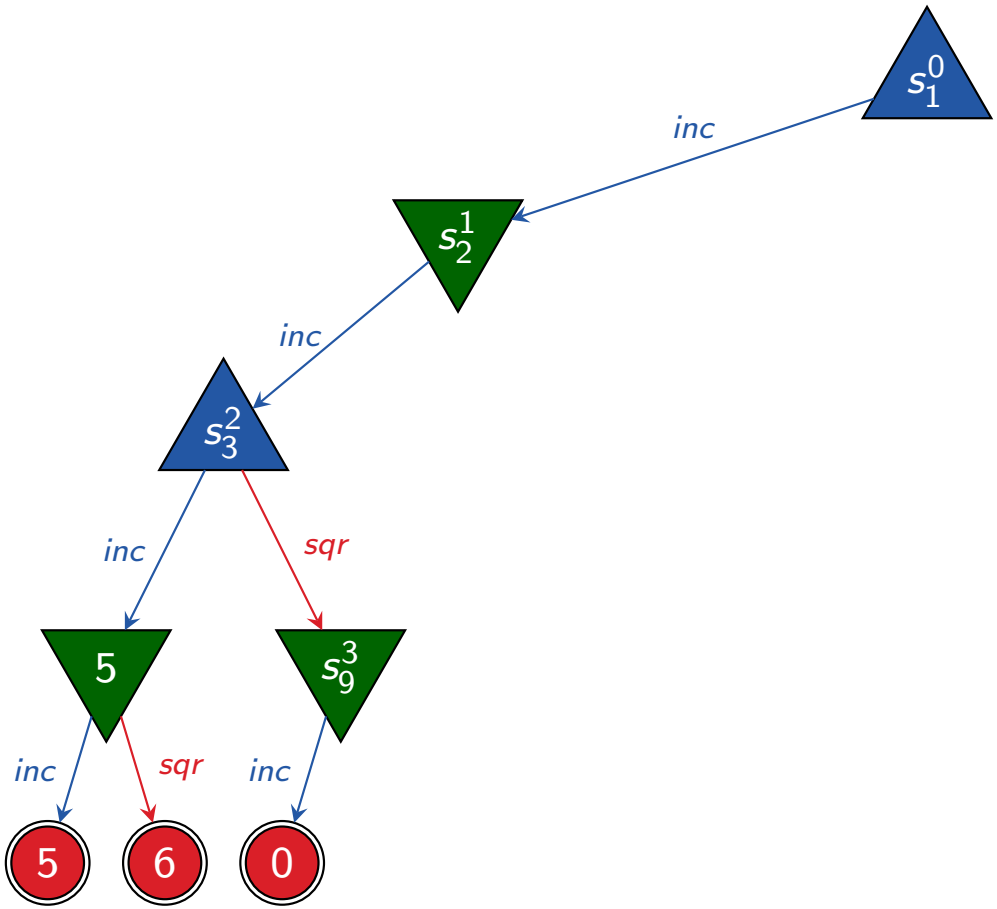
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

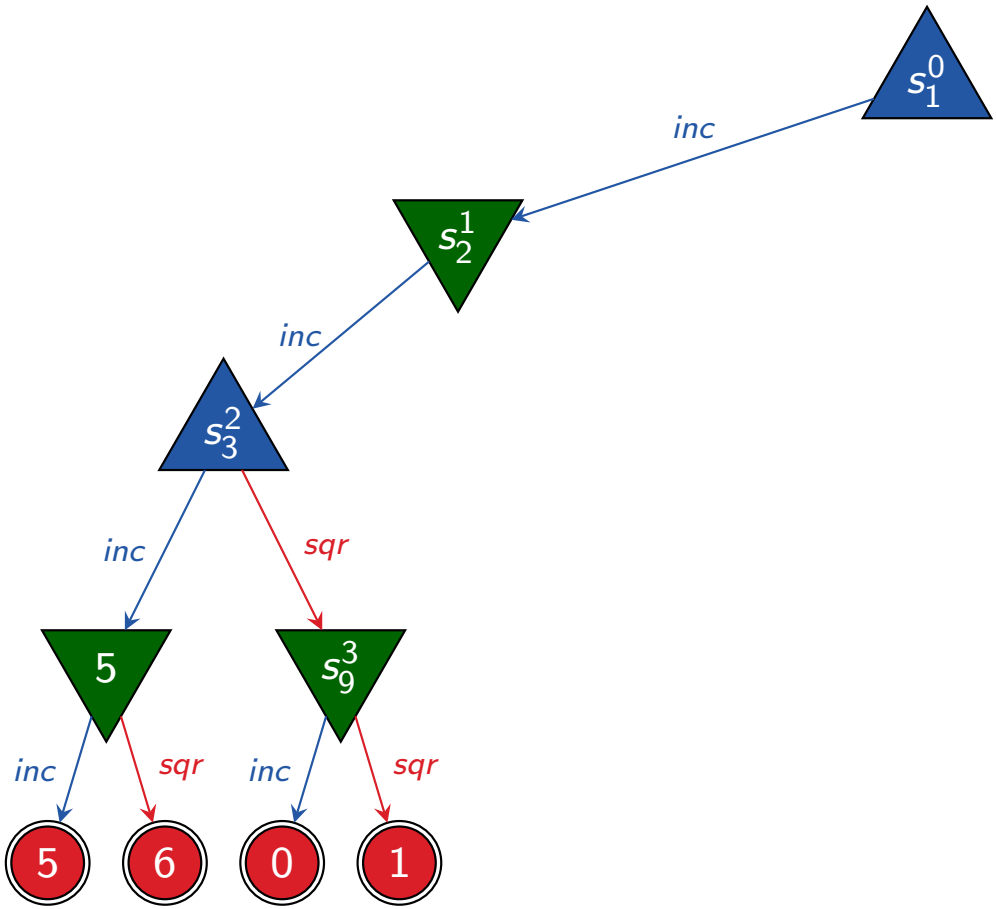
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

Idea and Example

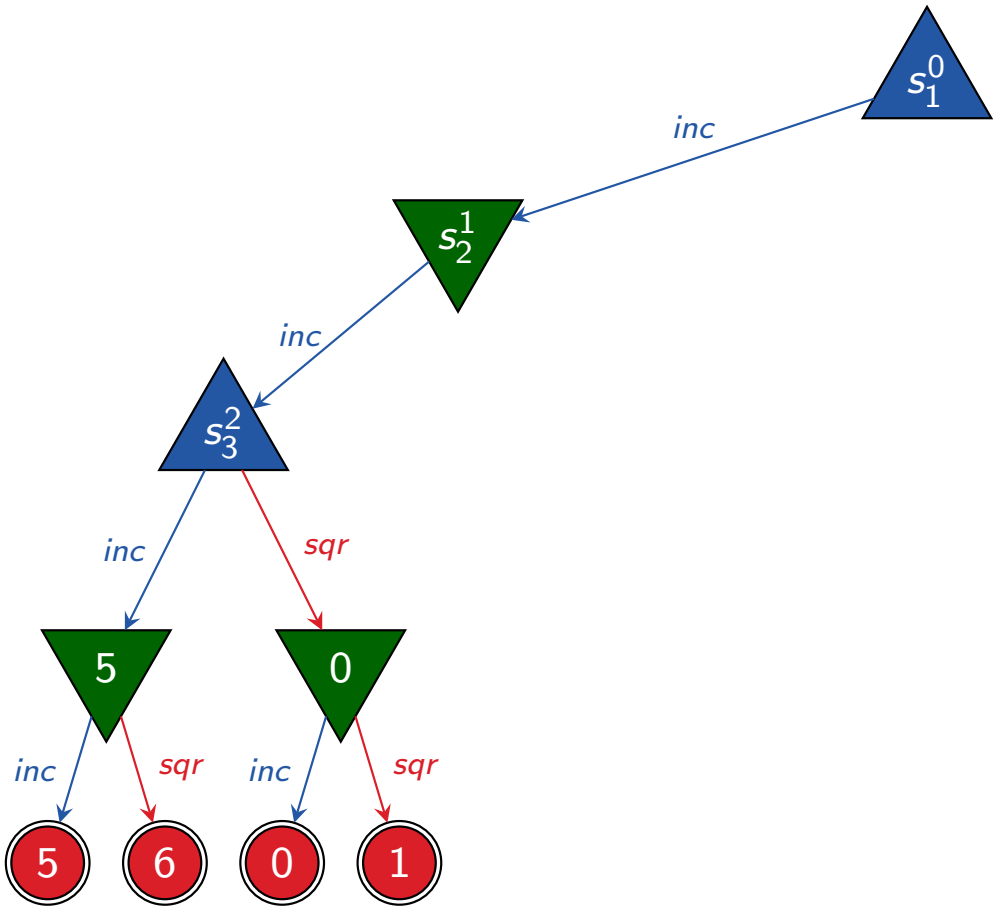


- depth-first search in game tree
- determine utility value of terminal position with utility function

- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

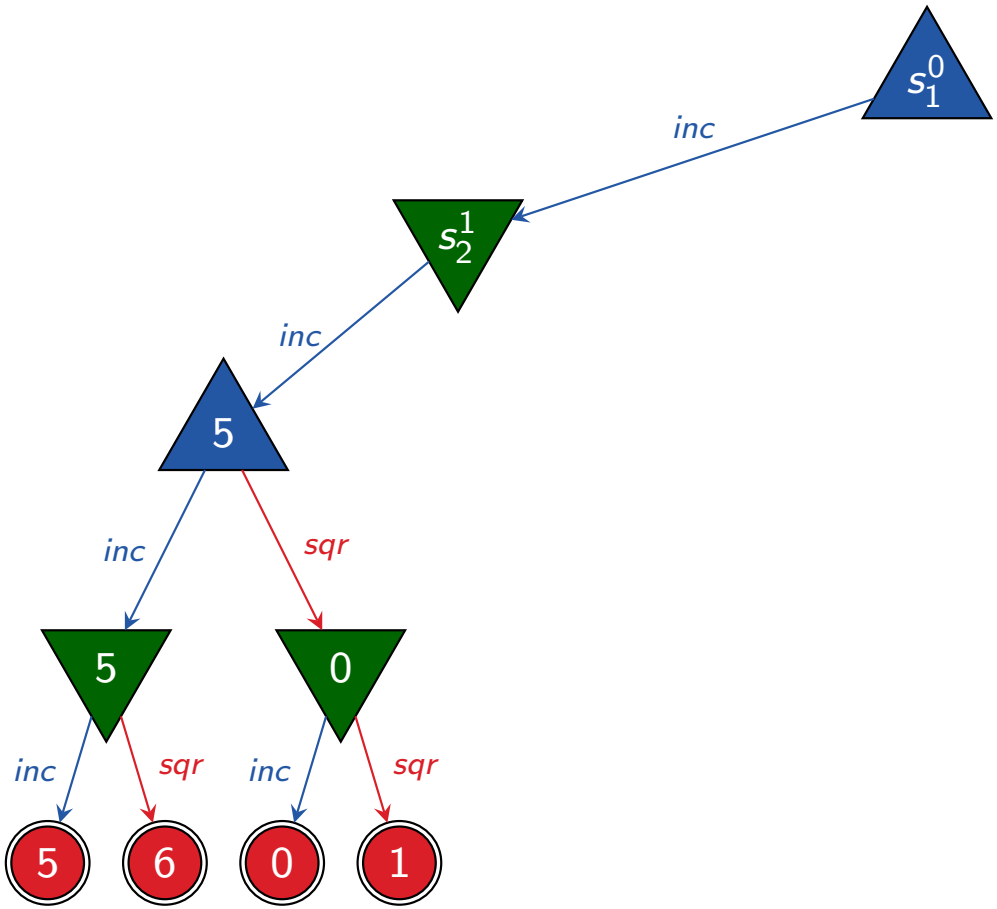
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

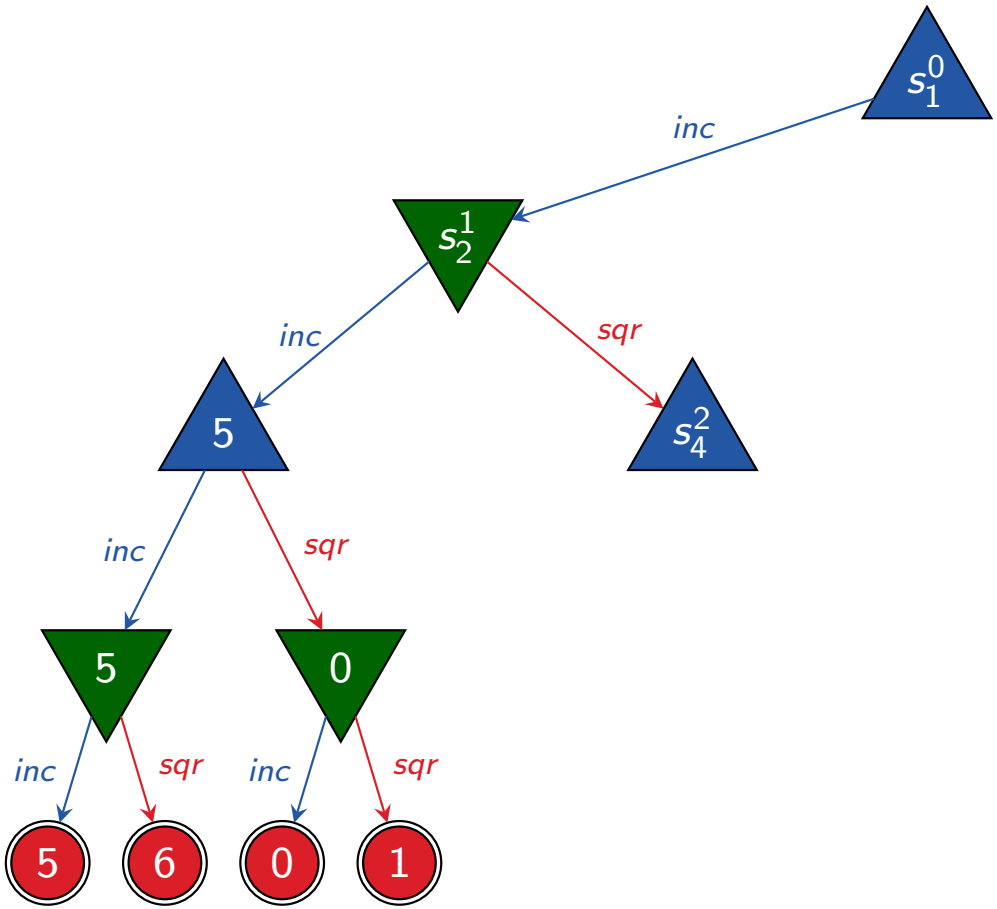
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

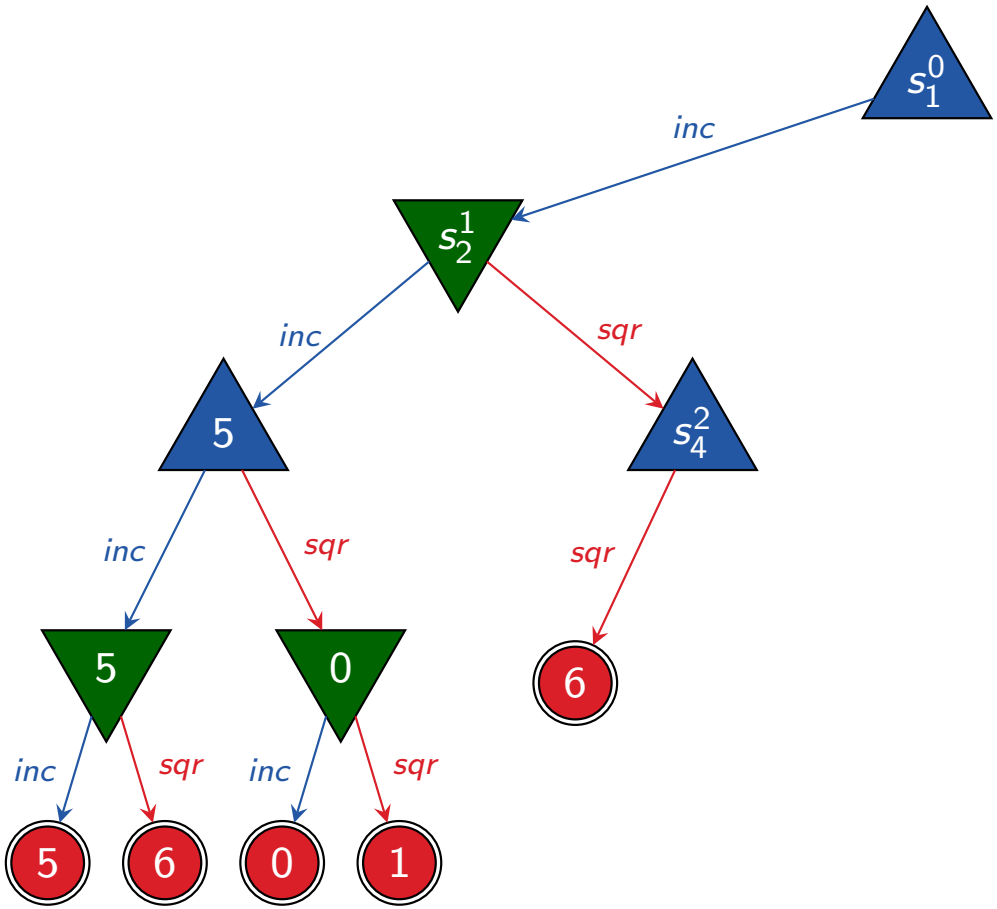
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

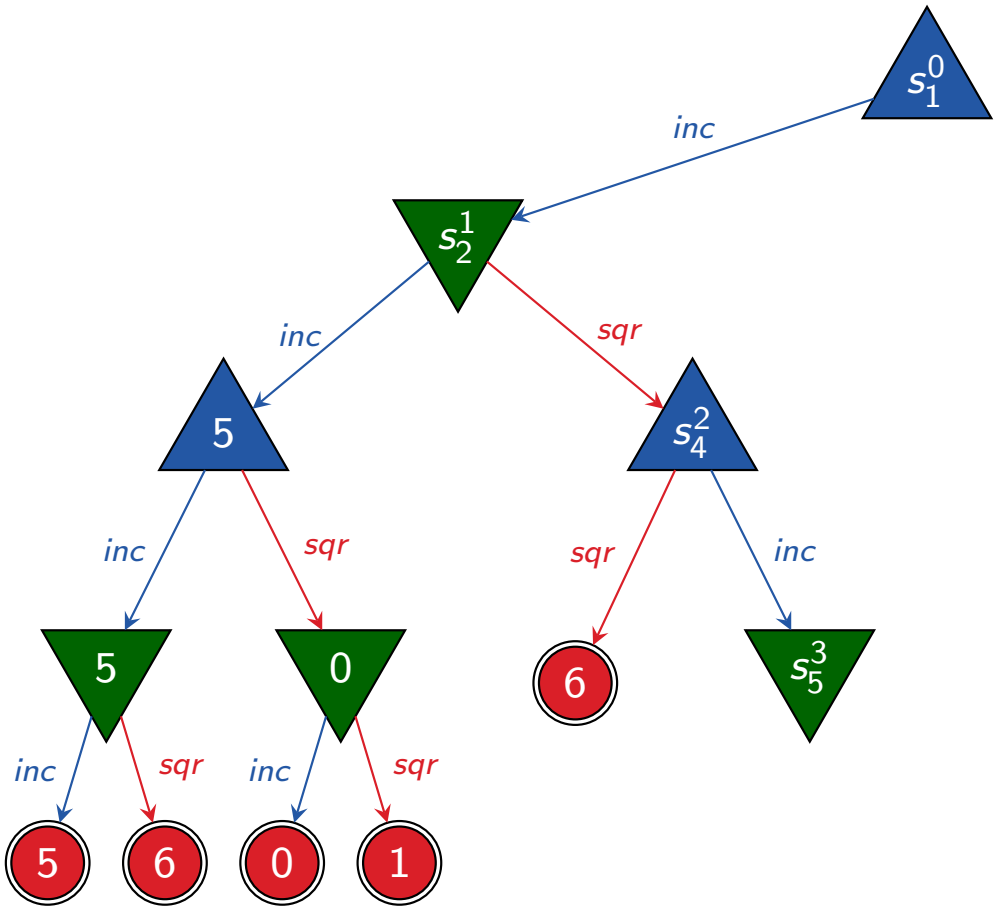
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

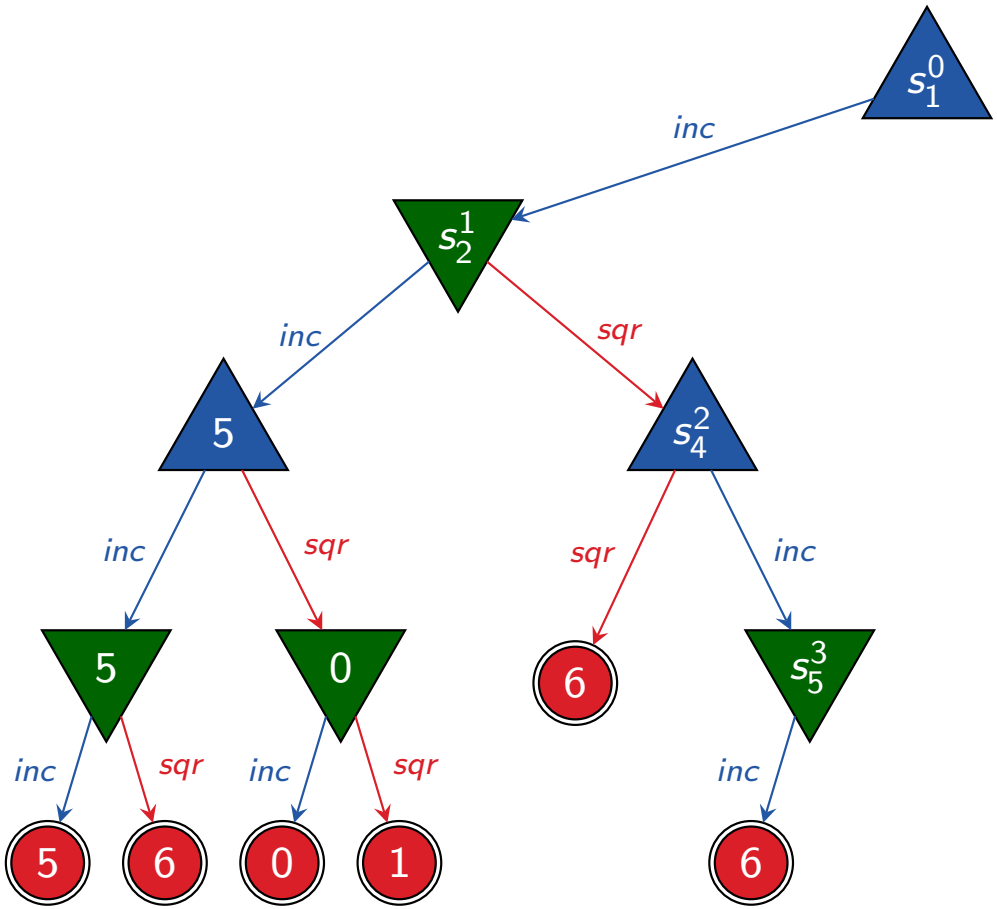
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

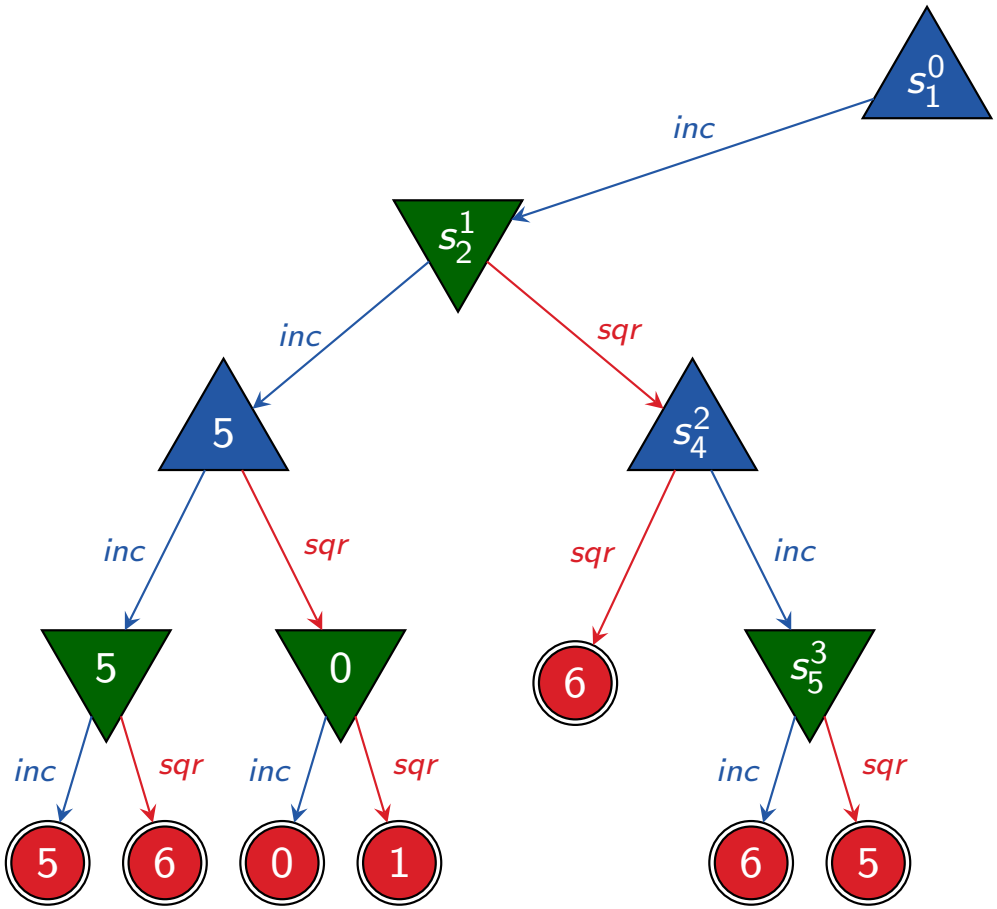
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

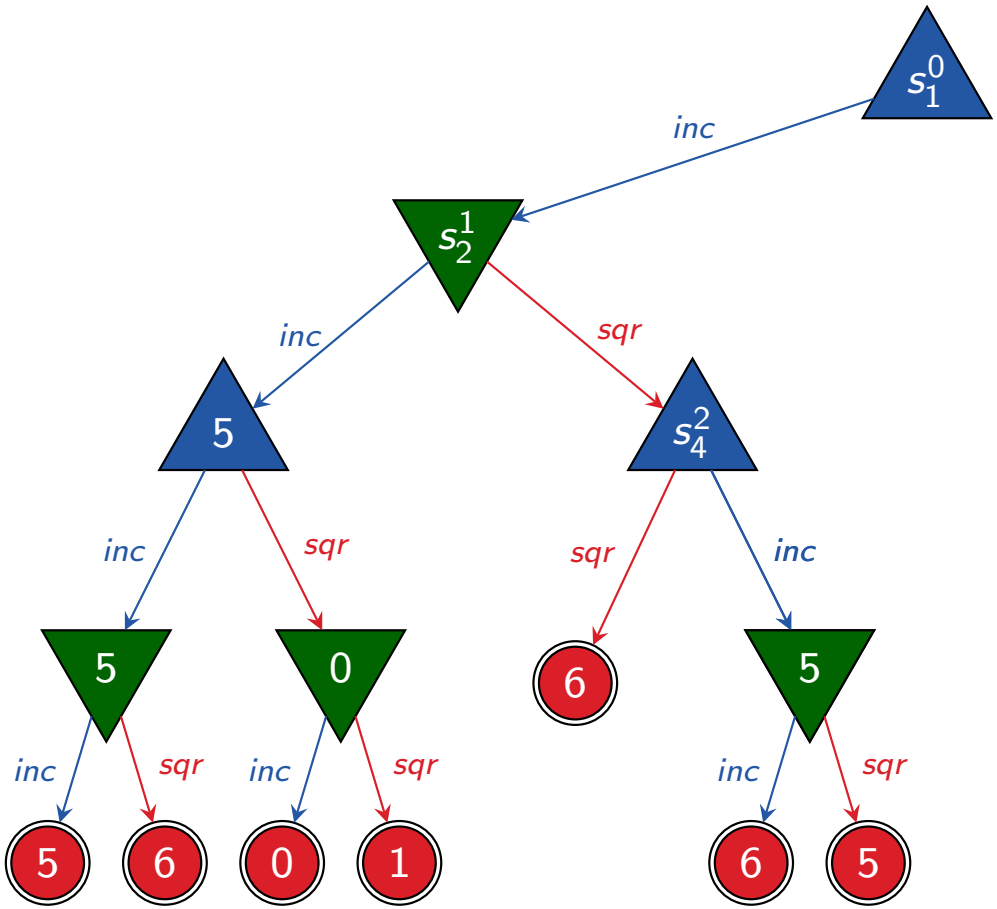
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

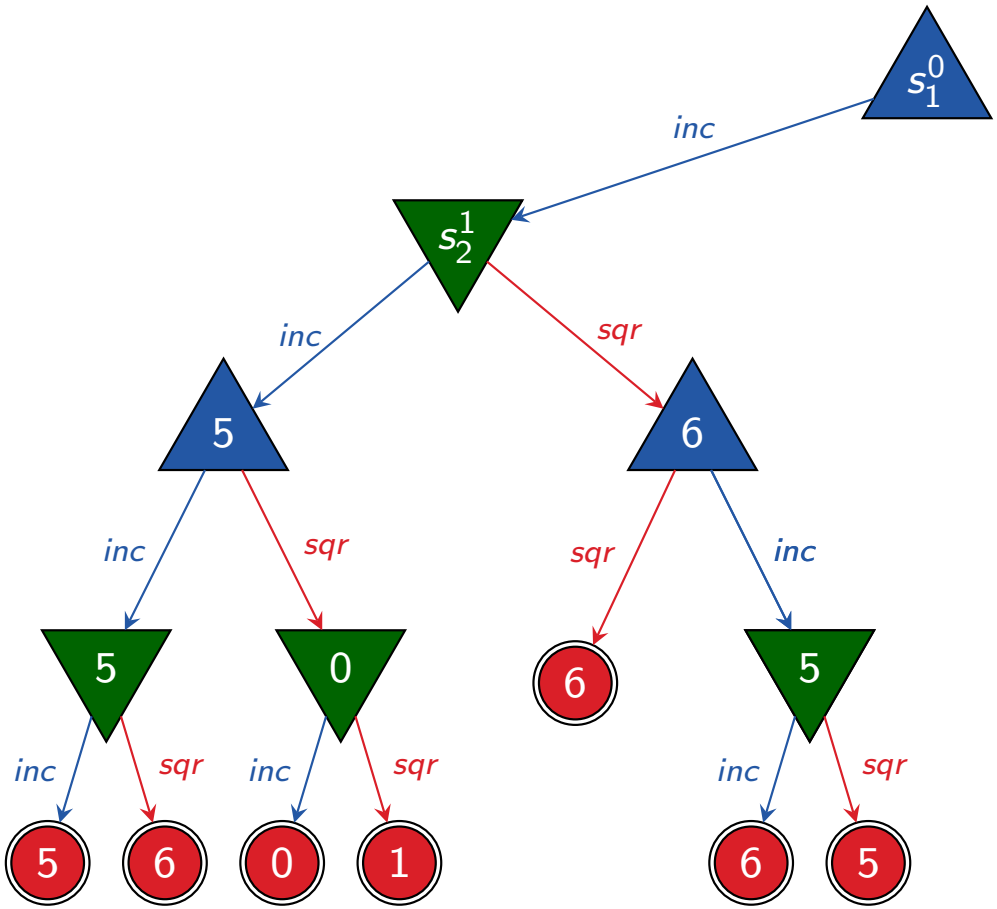
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

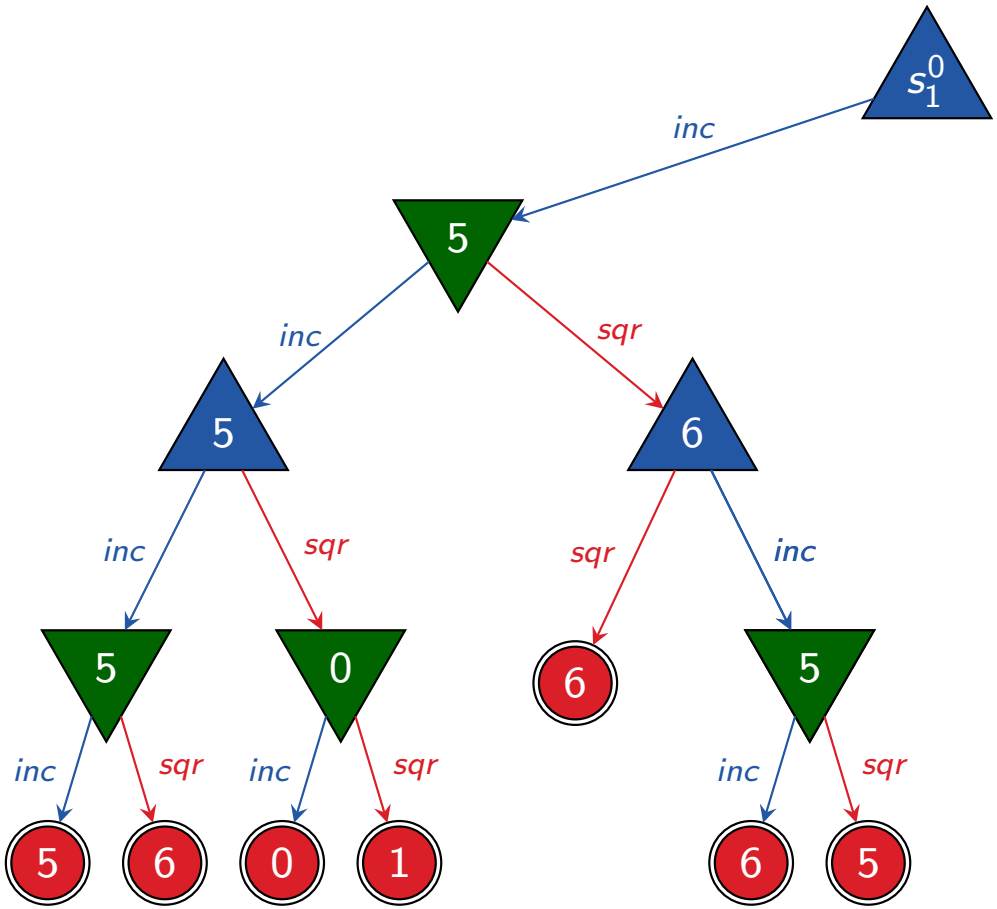
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

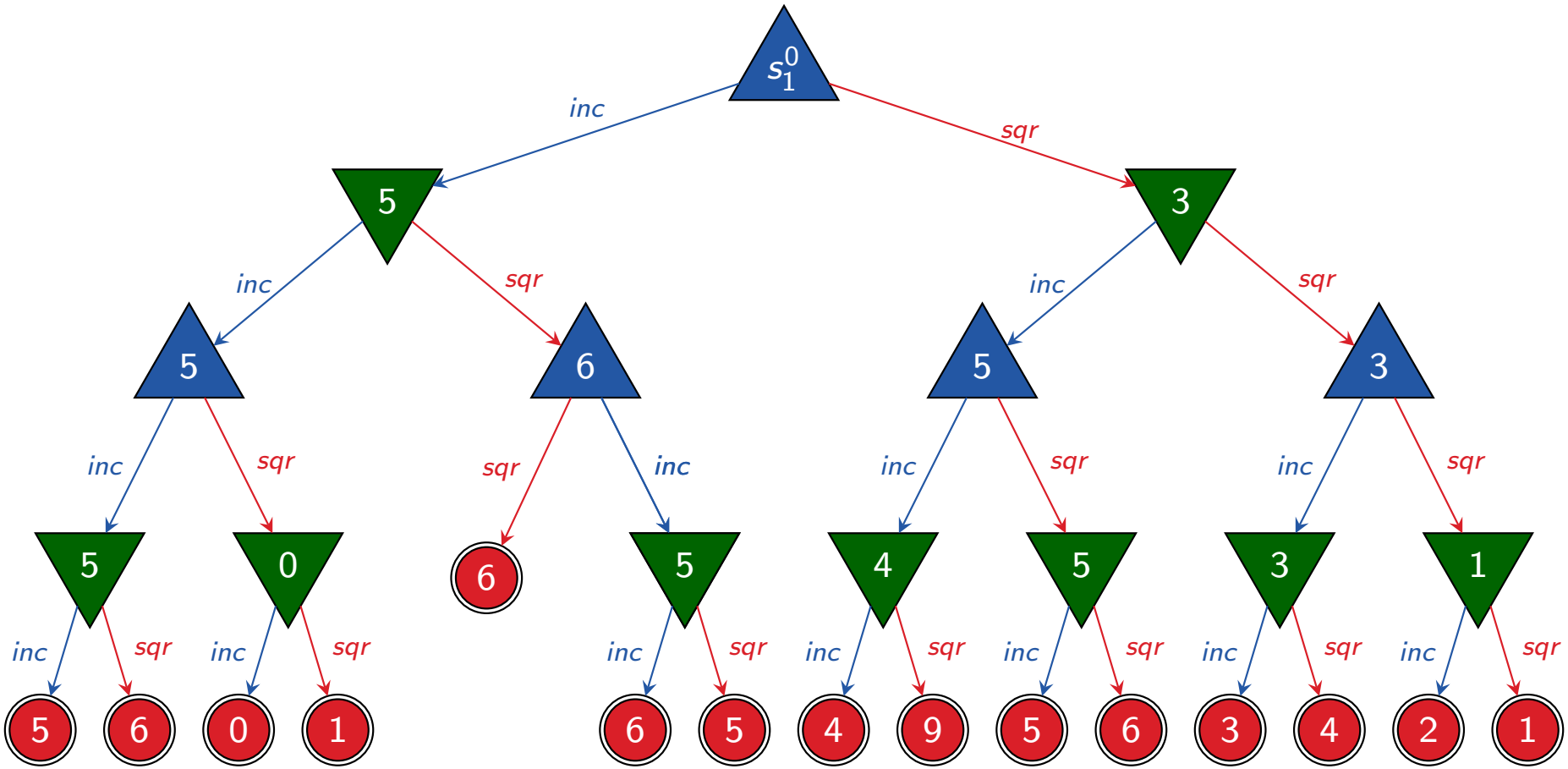
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

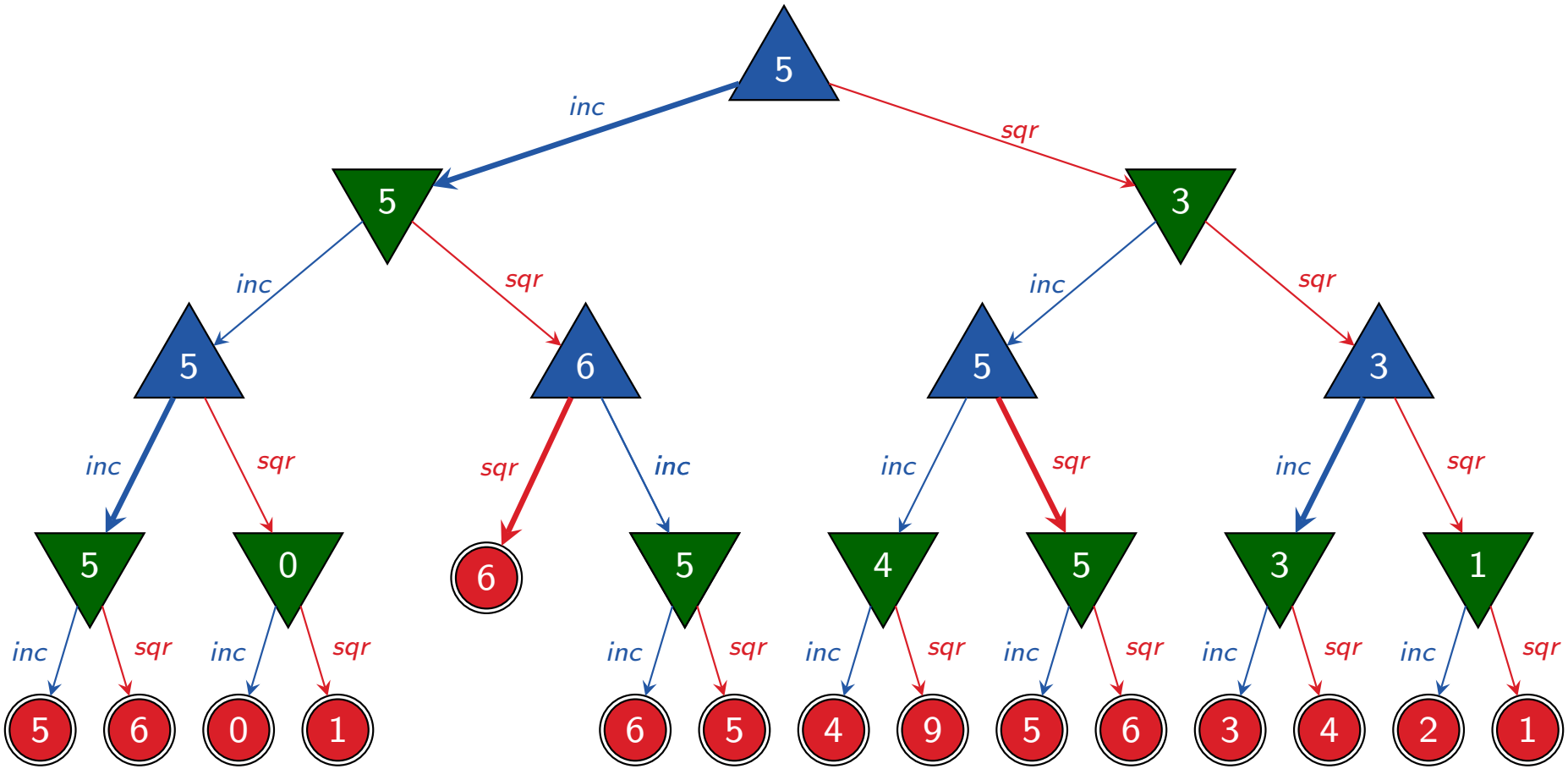
Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- strategy: action that maximizes utility value (minimax decision)
- compute utility value of inner nodes

  from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

Minimax: Pseudo-Code

function minimax(p)

if p is terminal position:
    return $\langle utility(p), \textbf{none} \rangle$
$best\_move := \textbf{none}$
if $player(p) = \text{MAX}$:
    $v := -\infty$
else:
    $v := \infty$
for each $\langle move, p' \rangle \in \text{succ}(p)$:
    $\langle v', best\_move' \rangle := minimax(p')$
    if ($player(p) = \text{MAX}$ **and** $v' > v$) **or**
       ($player(p) = \text{MIN}$ **and** $v' < v$):
        $v := v'$
        $best\_move := move$
return $\langle v, best\_move \rangle$

Discussion

- minimax is the simplest (decent) search algorithm for games
- yields optimal strategy (in the game-theoretic sense, i.e., under the assumption that the opponent plays perfectly)
- MAX obtains at least the utility value computed for the root, no matter how MIN plays
- if MIN plays perfectly, MAX obtains exactly the computed value

Limitations of Minimax

What if the size of the game tree is too big for minimax?

$\rightsquigarrow$ heuristic alpha-beta search

- heuristics (evaluation functions): Chapter G3
- alpha-beta search: Chapter G4

Summary

Summary

- Board games can be considered as classical search problems extended by an opponent.
- Both players try to reach a terminal position with (for the respective player) maximal utility.
- Minimax is a tree search algorithm that plays perfectly (in the game-theoretic sense).
- Its time complexity is $O(b^d)$ (branching factor b, search depth d), which is too big for most interesting games.