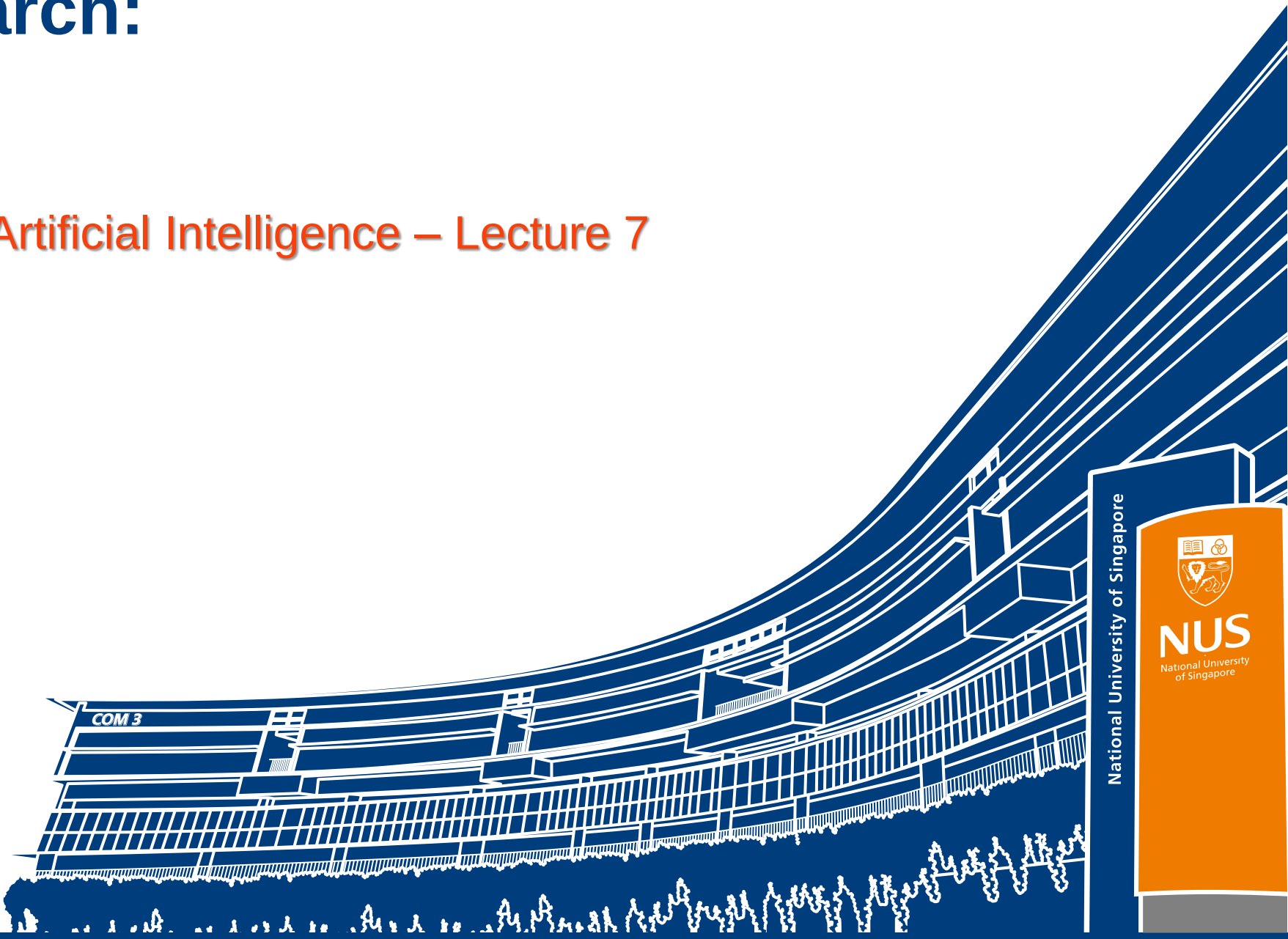
Adversarial Search: Playing Games

CS3243: Introduction to Artificial Intelligence – Lecture 7



NUS | Computing
National University of Singapore

1 Administrative Matters

Midterm Survey Results

- General feedback on course (40 responses; 45% of the class)

	Reasonable Difficulty	Fair (Continual) Assessments	Interesting Content	Course Design
Strongly Agree	4	11	17	12
Agree	31	23	22	24
Disagree	5	6	1	4
Strongly Disagree	0	0	0	0

	Lectures	DQs	Tutorials	Forums	Projects
Voted Useful	37 (92.5%)	19 (47.5%)	28 (70.0%)	12 (30.0%)	30 (75.0%)

Difficult Assessments	Projects: 35 (87.5%)	Tutorials: 6 (15.0%)	DQs: 3 (7.5%)

Midterm Survey Results

- Difficult topics?
  - Modelling problems [9]
  - Modelling heuristics [9]
  - Algorithm implementations (projects) [6]
  - Complexity analysis [4]
  - Proofs [3]
  - CSPs and Inference [2]
  - Graph and tree search variants
  - A* Search

2D Bloom's Taxonomy image source: https://pressbooks.cuny.edu/leh250/chapter/understanding-blooms-taxonomy/

Midterm Survey Comments – Projects

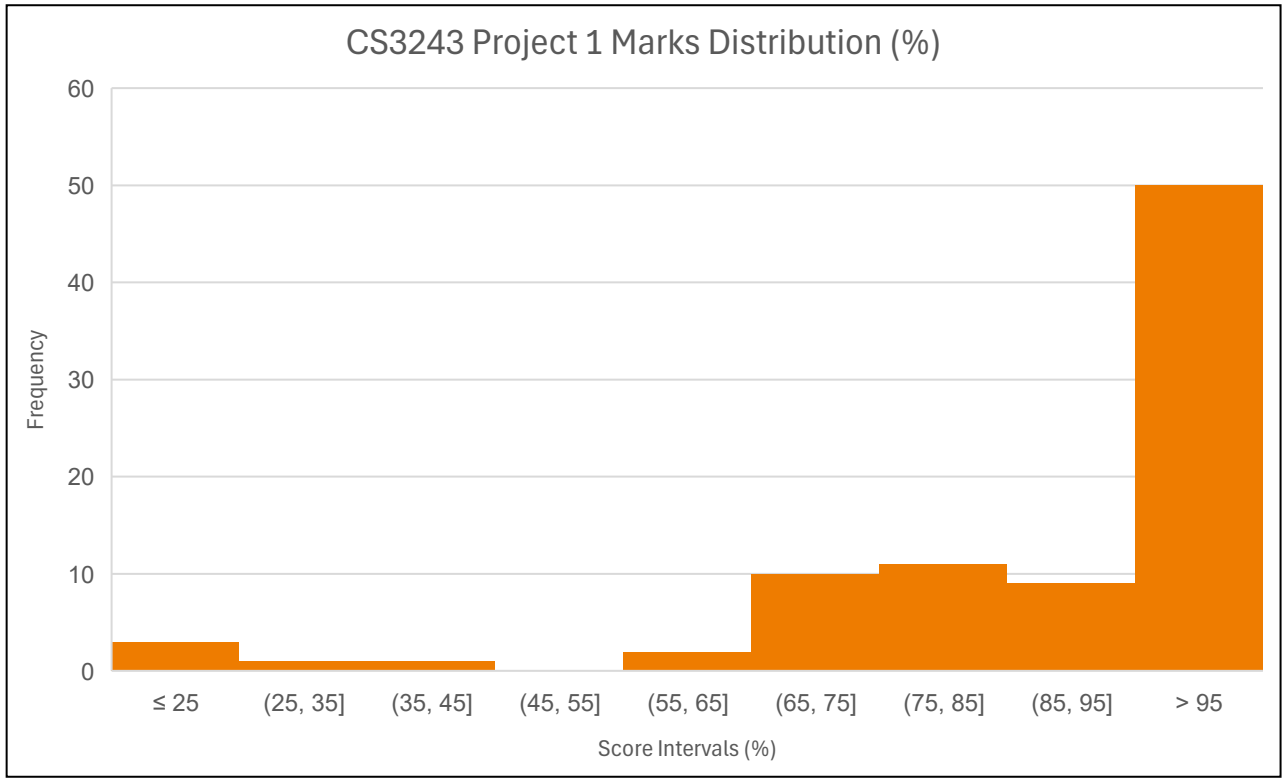
- ... not the leetcoder type ... which have made the projects so far more difficult. I would love to be directed to any resources that would help with this.
- ... give more tips on the projects ... like a nudge in the right direction 1/2 a week before the deadline.
- ... I think maybe more hints could be dropped along the way as promised ...

  ...I thought it would be nice if the pseudo code could be released after projects as well because right now some of us still could not figure out what is causing the code to fail certain test cases even after using the initial hints ... It could even just be more of worded answers in the general direction of the pseudo code ...

- ... relaxing the requirements of the efficiency test cases a little ... increase the proportion of public test cases.
- We might need more public test cases in projects.
- ... It would make work quicker if the public test (cases) would also have a solution(s) included.
- A lot of time was spend trying to print out the maze to debug. Maybe provide template code for printing the maze etc. to help debugging so the focus is more on the algo to solve the problem.
- ... could deadlines from Coursemology be added to Canvas assignments? I believe they can be no-submission but would appear on student's dashboards with their other course assignments regardless.

Project 1 Results

- Project 1.1 (MAX = 3)
  - Mean: 2.89
  - Median: 3
- Project 1.2 (MAX = 7)
  - Mean: 5.83
  - Median: 6.89
- Project 1 (%)
  - Mean: 87.27
  - Median: 97.8



Midterm Survey Comments – Non-Project Matters

- Do not like having to keep track of 2 tutorial submissions every week, in addition to all else in this course (Projects, Midterms, etc). Would be much better without this tutorial submission stuff.

- Having both pre and post tutorial assignments including the tutorials themselves feels redundant. Could do with just tutorial or tutorial and pre tutorial.

- Coming from a university abroad, I really enjoyed the well-defined structure of the course. The double tutorial is also a really well implemented idea, making sure that the students come in having already pre-read the material.

- ... please release the score of the tutorial as soon as possible.

- I wish the tutorials were more interactive and collaborative, but they're basically like a second lecture.

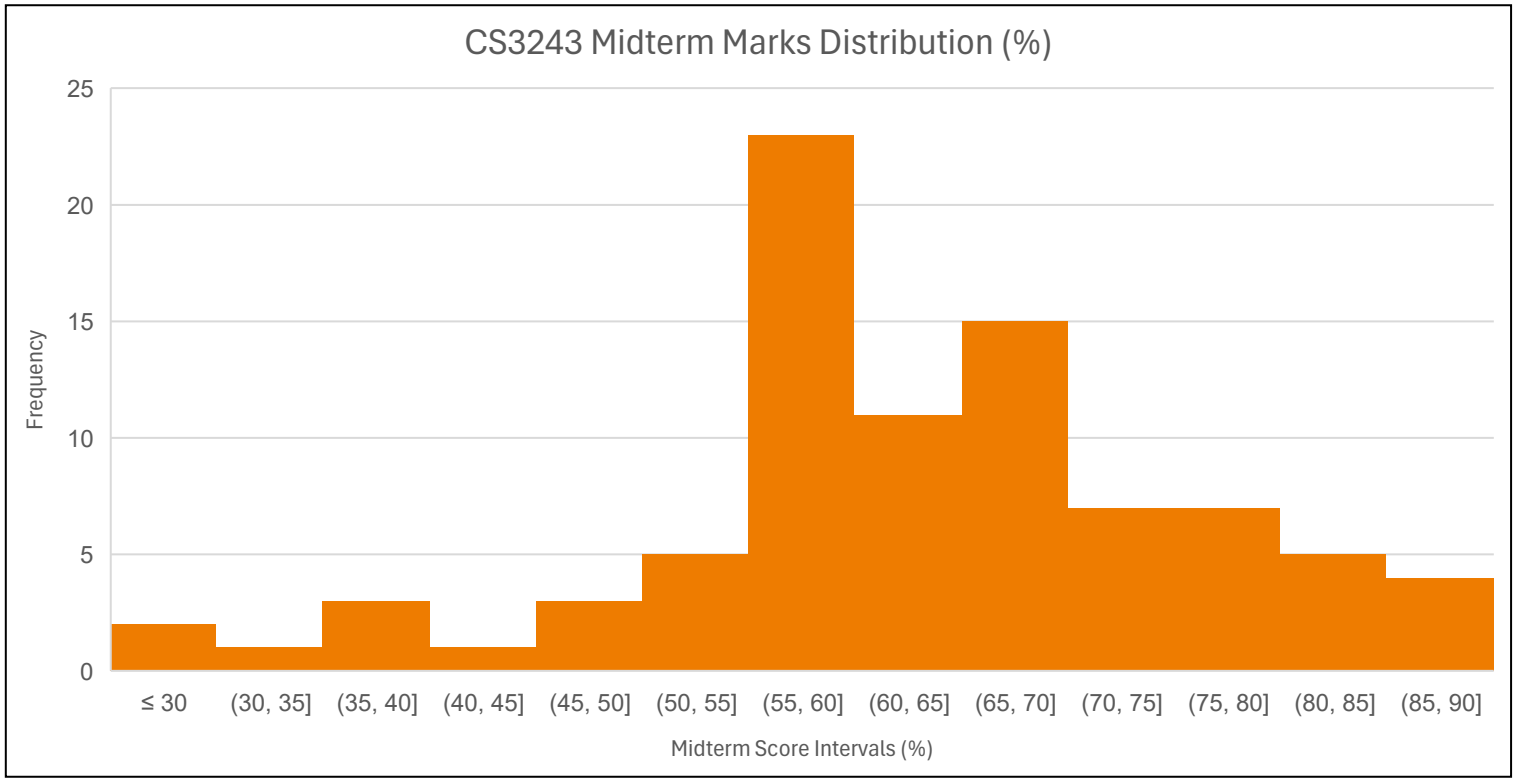
- I didn't particularly enjoy some of the assignments for search space formulation, when the state is already predefined - it sometimes helps, if your idea for the state of the solution is similar but can be a hurdle if you're thinking of some alternative approach.

- I'm struggling with the support options between project consultations, the discussion forums, and texting my TA on telegram. I'm more used to having a wider range of consultations (even just 2 separate times per week) for both general concepts and projects - has this been considered before? I also find that neither discussion forums, nor texting my TA is conducive to my learning style I tend to ask a lot of clarifying/follow-up questions ...

  In terms of using Archipelago ... an unintended effect of Archipelago is guilt that if I ask a question, I'm causing lecture to be released late.

Midterm Results

- Midterm Q1 (%)
  - Mean: 32.33
  - Median: 30.0
- Midterm Q2 (%)
  - Mean: 76.16
  - Median: 80.0
- Moderated Midterm (%)
  - Mean: 63.51
  - Median: 64.0



Midterm Appeals

- Rubrics
  - Will be available on Canvas later today
- Appeal procedure
  - Email or message your tutor on Telegram
  - For each appeal, include the following
    - Question number and part - e.g., Q2a(i)
    - Screenshot of your solution
    - Desired mark change (must be specific) – e.g., +1 mark
    - Succinct rationale for change (no essays)
- Appeals deadline
  - Week 9 (TUE), OCT 15, 2359 hrs

Upcoming...

- Deadlines
  - No Post-tutorial Assignment due this week!
  - Tutorial Assignment 6 (released today)
    - Pre-tutorial Submission: Due this Sunday!
    - Post-tutorial Submission: Due next Friday!
  - Project 2.1 (released 16 September)
    - Due THIS Sunday (i.e., Week 8)!

      13 October 2024
  - Project 2.2 (released 16 September)
    - Due NEXT Sunday (i.e., Week 9)!

      20 October 2024

Late Penalties
- < deadline +24 hours = 80% of score
- < deadline +48 hours = 50% of score
- ≥ deadline +48 hours = 0% of score

Project Consultations:
Thursdays via Zoom
Canvas > Announcements

Contents

- Adversarial Search Problems
- Optimal Decisions via Minimax
- α-β Pruning
- Heuristic Minimax

2 Formulating Adversarial Search Problems

Thus Far...

- Path search (path planning)
  - Search for a path from start to goal
    - Complete: finds a solution or says when there isn't one
    - Optimal: total cost of path returned is minimal
  - Uninformed
    - Systematically search all paths via general search problem formulation
  - Informed
    - Uses a heuristic to search less of the search space

- Goal search (find legal goal state)
  - Focus on goal and ignore path
    - Completeness consideration only
  - Local search
    - Uses heuristic to guide search to goal (uses restarts; many variants)
  - Constraint satisfaction problem
    - Uses specific search problem formulation and shrinks search space via inference

Games and Search

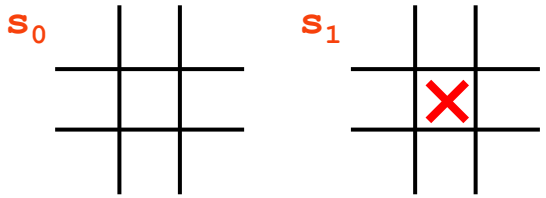
- Can we solve games using existing methods?
  - In our searching thus far, we control all actions
    - All actions taken are determined by our agent
  - With games, your opponent decides actions too...
    - Multi-agent problem
    - Conventional planning ⇒ wasted computation
      since opponent can spoil your plans

Games and Search

- What is a game anyway?
  - Assume two players
  - Zero-sum game
    - Winner gets paid, and loser pays
  - We define
    - MAX player – player 1, who wants to maximise value (our agent)
    - MIN player – player 2, who want to minimise value (opponent who wants our agent to lose)
- General idea behind the adversarial search problem
  - Simulate play against utility-maximising opponent
  - Find a strategy – i.e., define a move for every possible opponent response

Formulating Games

- State representation
  - As per general search formulation



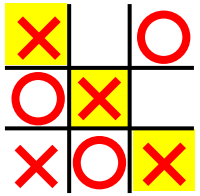
- `TO-MOVE(s)`
  - Returns `p`, the player to move in state `s`
- `ACTIONS(s)`
  - Legal moves in state `s`
- `RESULT(s, a)`
  - The transition model;
    returns the resultant intermediate state
    when taking action `a` at state `s`
- `IS-TERMINAL(s)`
  - Returns `True` when the game is over
    and `False` otherwise
  - States where game has ended are
    called terminal states
- `UTILITY(s, p)`
  - Defines a numeric value (score) for player `p`
    when the game ends at terminal state `s`

Note on Utility

- Given zero-sum games
  - At terminal state `s`
    - `utility(MAX, s)` + `utility(MIN, s)` = 0

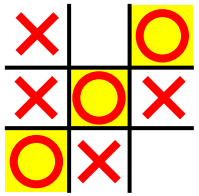
- Tic-Tac-Toe (Noughts and Crosses) example
  - `X` (agent) wins
    - `UTILITY(s_i, MAX) = 1`
    - `UTILITY(s_i, MIN) = -1`

s_i:



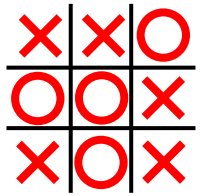
  - `O` (opponent) wins
    - `UTILITY(s_j, MAX) = -1`
    - `UTILITY(s_j, MIN) = 1`

s_j:



  - Draw
    - `UTILITY(s_k, MAX) = 0`
    - `UTILITY(s_k, MIN) = 0`

s_k:

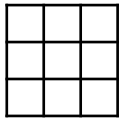


Notice that the given utilities are relative to the player – however, we will standardise the scores such that they reflect only the `MAX` player (i.e., our agent) scores from now onwards.

Just use `utility(MAX)` = `-utility(MIN)`

Game Trees

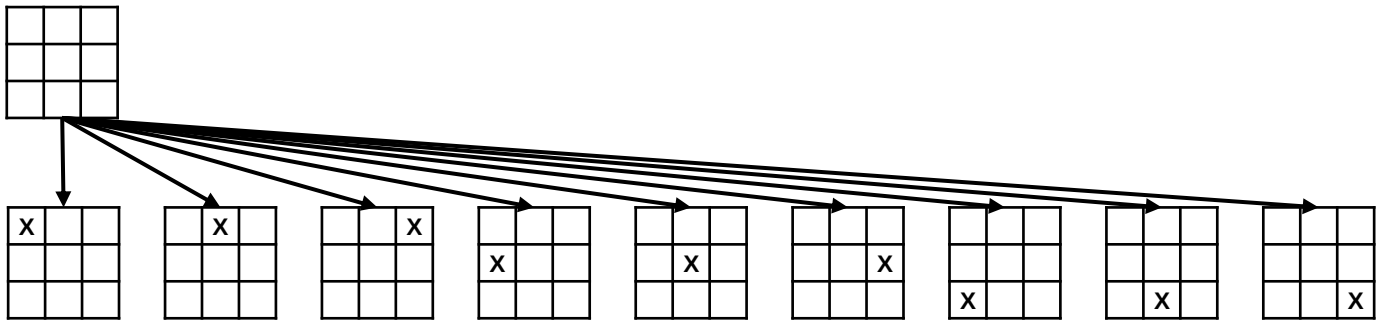
- Example (Tic-Tac-Toe)



- Environment characteristics
  - 2-player
  - Deterministic
  - Turn-taking
  - Zero-sum
- Zero-sum implications
  - Loser exists for every winner
  - Completely adversarial game
  - `utility(MAX) = -utility(MIN)`
  - May also consider constant-sum game

Game Trees

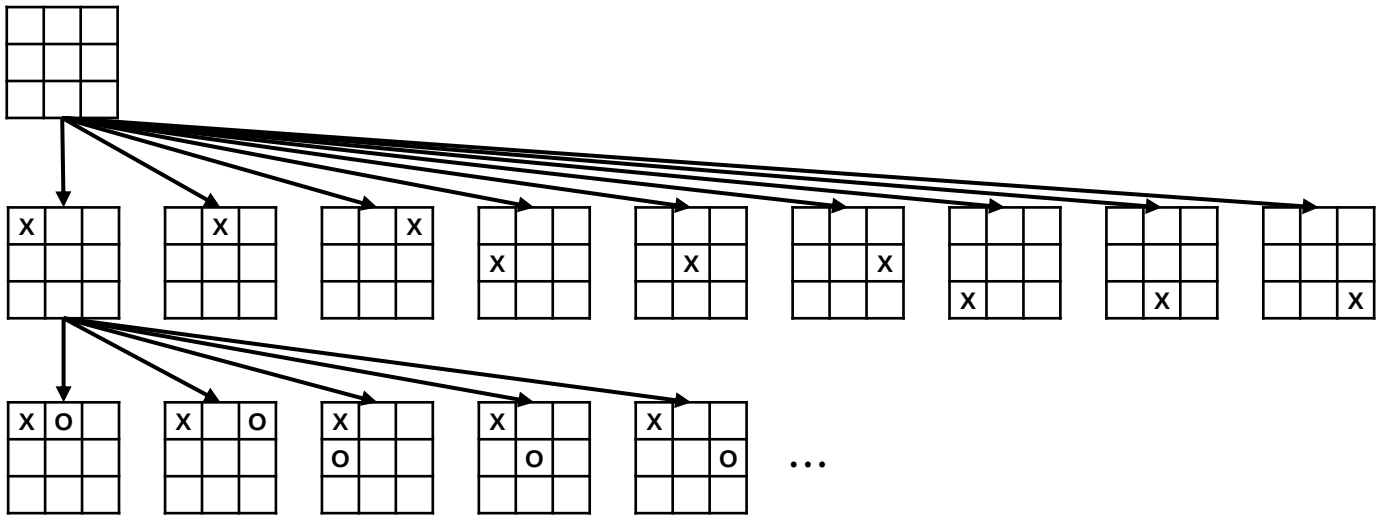
- Example (Tic-Tac-Toe)



- Environment characteristics
  - 2-player
  - Deterministic
  - Turn-taking
  - Zero-sum
- Zero-sum implications
  - Loser exists for every winner
  - Completely adversarial game
  - `utility(MAX)` = `-utility(MIN)`
  - May also consider constant-sum game

Game Trees

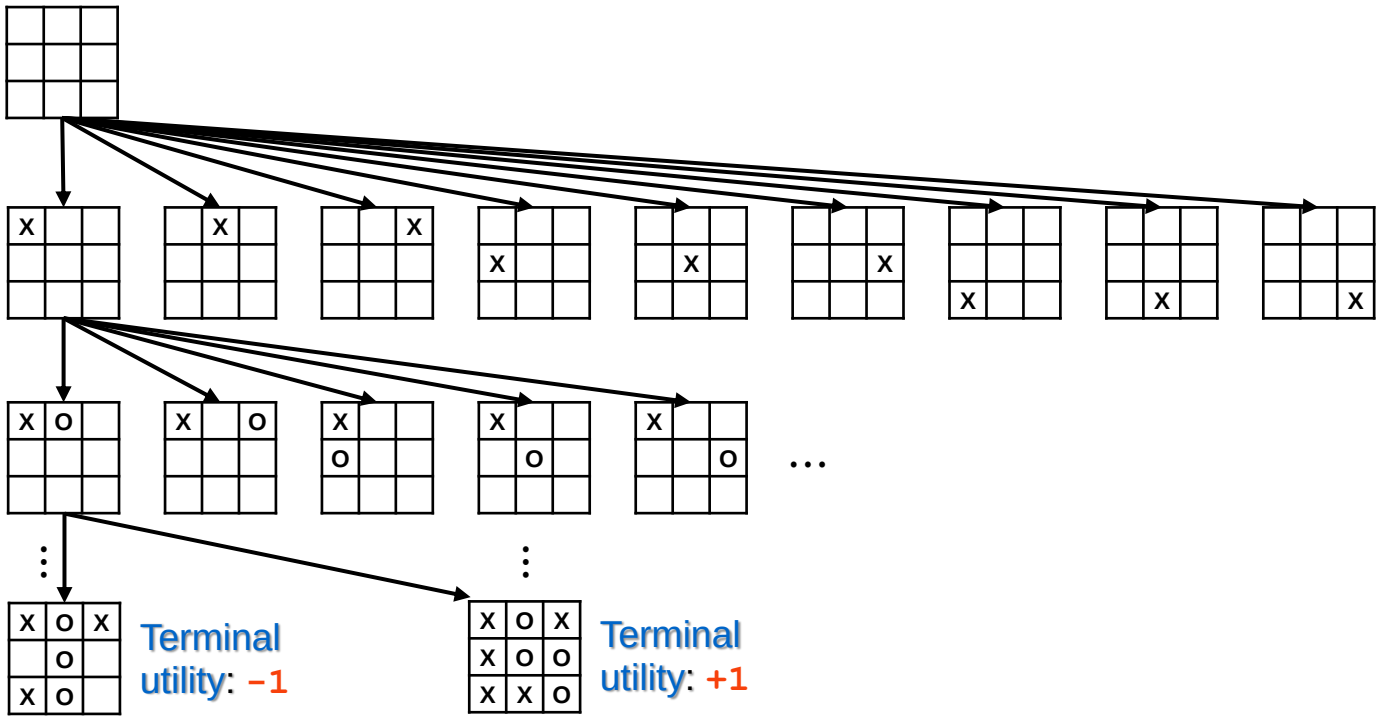
- Example (Tic-Tac-Toe)



- Environment characteristics
  - 2-player
  - Deterministic
  - Turn-taking
  - Zero-sum
- Zero-sum implications
  - Loser exists for every winner
  - Completely adversarial game
  - `utility(MAX)` = `-utility(MIN)`
  - May also consider constant-sum game

Game Trees

- Example (Tic-Tac-Toe)



- Environment characteristics
  - 2-player
  - Deterministic
  - Turn-taking
  - Zero-sum
- Zero-sum implications
  - Loser exists for every winner
  - Completely adversarial game
  - `utility(MAX)` = `-utility(MIN)`
  - May also consider constant-sum game

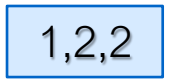
Another Example: Game of NIM

- Several piles of sticks are given
  - Represent the configuration of piles by a *monotone* sequence of integers
    - Example: (1,3,5)
  - With each turn, a player may remove any number of sticks from ONE pile
    - Example:
      - Remove 4 sticks from last pile (of 5 sticks) ⇒ (1,3,5) becomes (1,1,3)
    - The player who takes the last stick loses

- Let's try...
  - Represent the NIM game (1,2,2) as a game tree

Game of NIM: (1,2,2) Game Tree

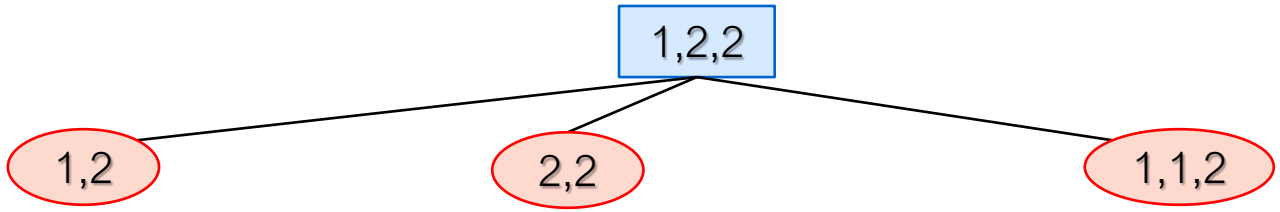
MAX (agent)



Game of NIM: (1,2,2) Game Tree

MAX (agent)

MIN (opponent)

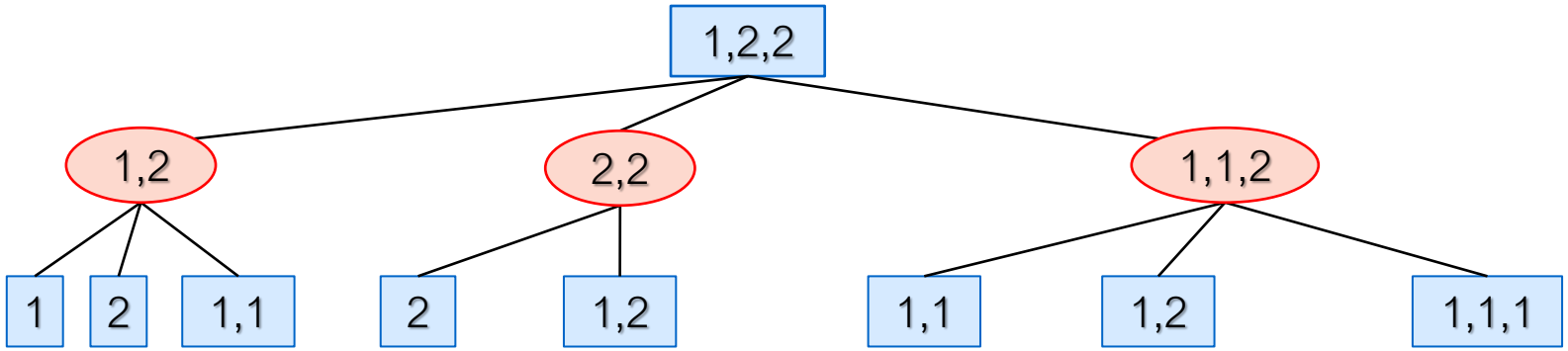


Game of NIM: (1,2,2) Game Tree

MAX (agent)

MIN (opponent)

MAX (agent)



MIN

MAX

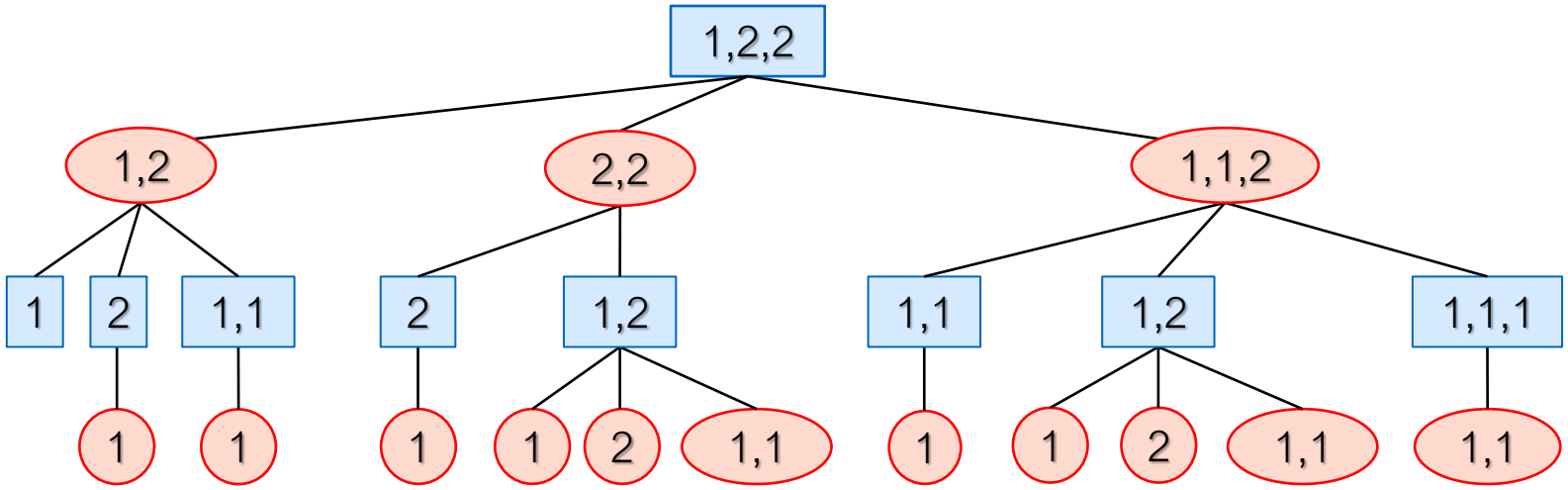
Game of NIM: (1,2,2) Game Tree

MAX (agent)

MIN (opponent)

MAX (agent)

MIN (opponent)



Game of NIM: (1,2,2) Game Tree

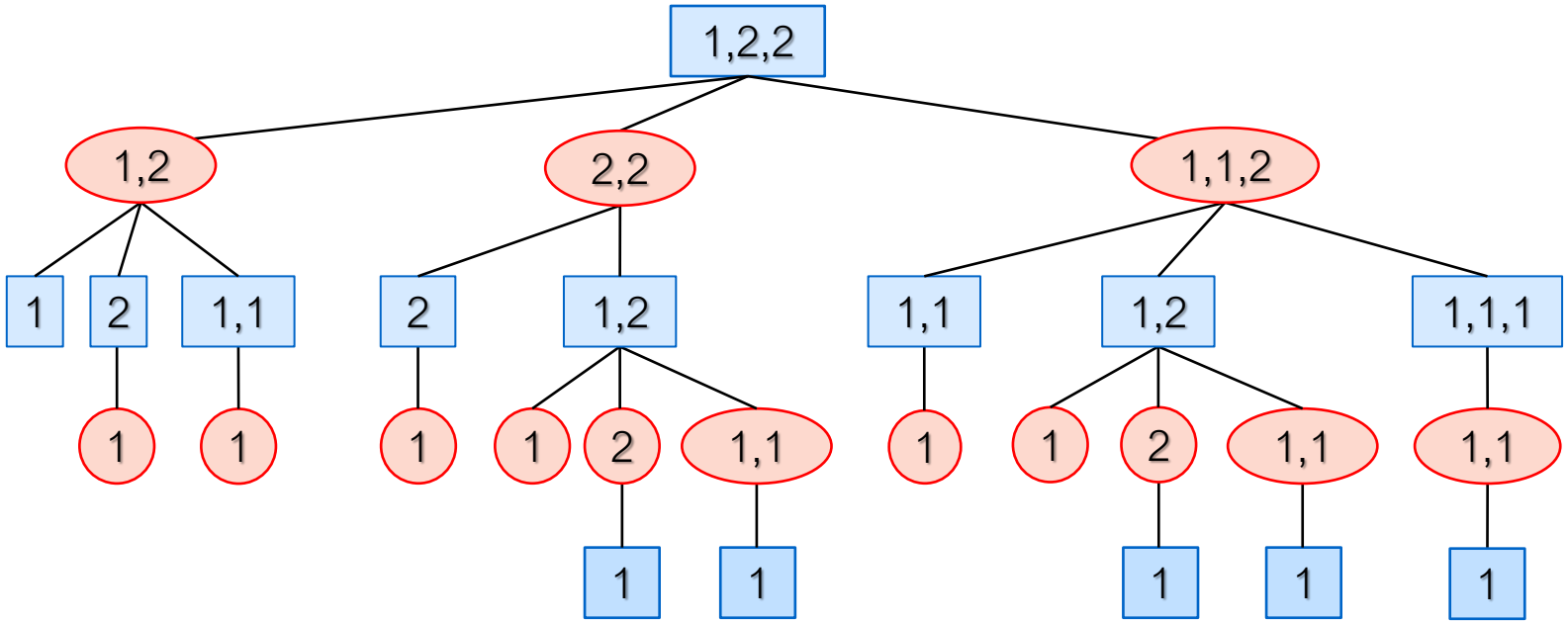
MAX (agent)

MIN (opponent)

MAX (agent)

MIN (opponent)

MAX (agent)



MIN

MAX

Game of NIM: (1,2,2) Game Tree

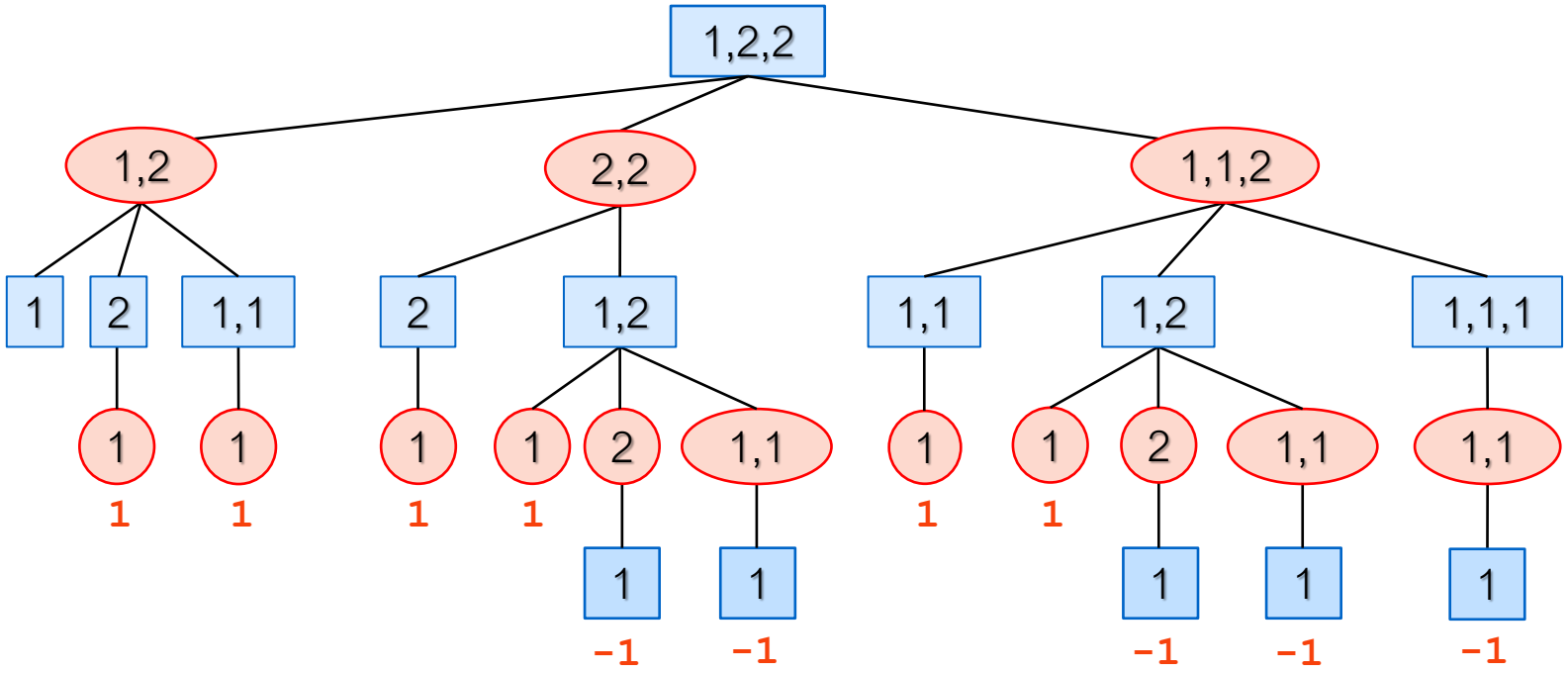
MAX (agent)

MIN (opponent)

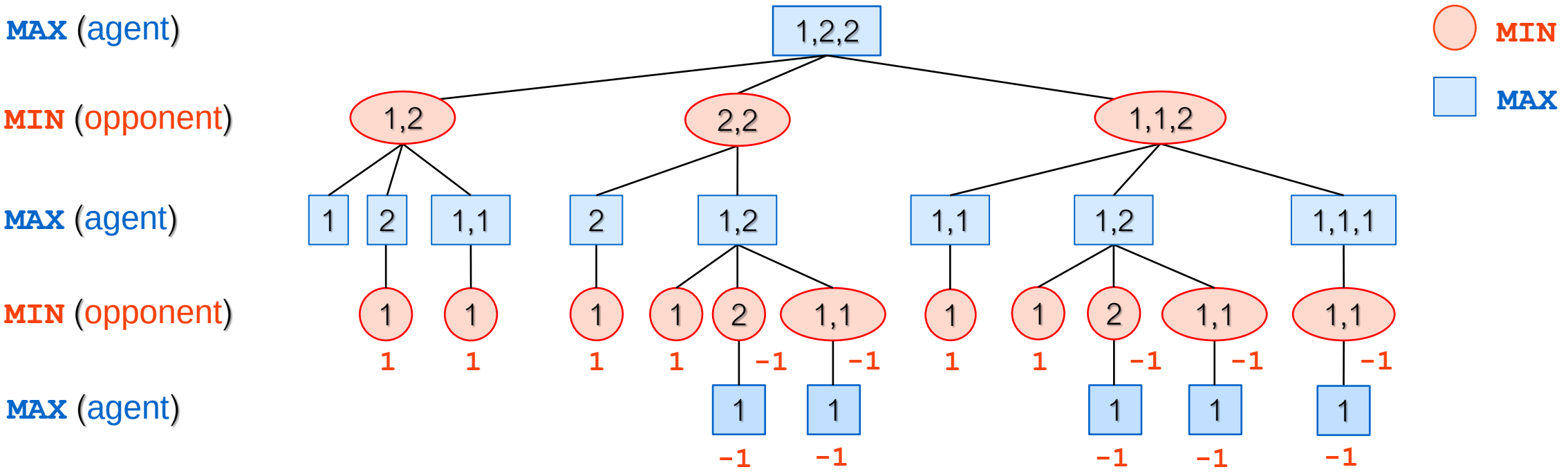
MAX (agent)

MIN (opponent)

MAX (agent)



Game of NIM: (1,2,2) Game Tree



Game of NIM: (1,2,2) Game Tree

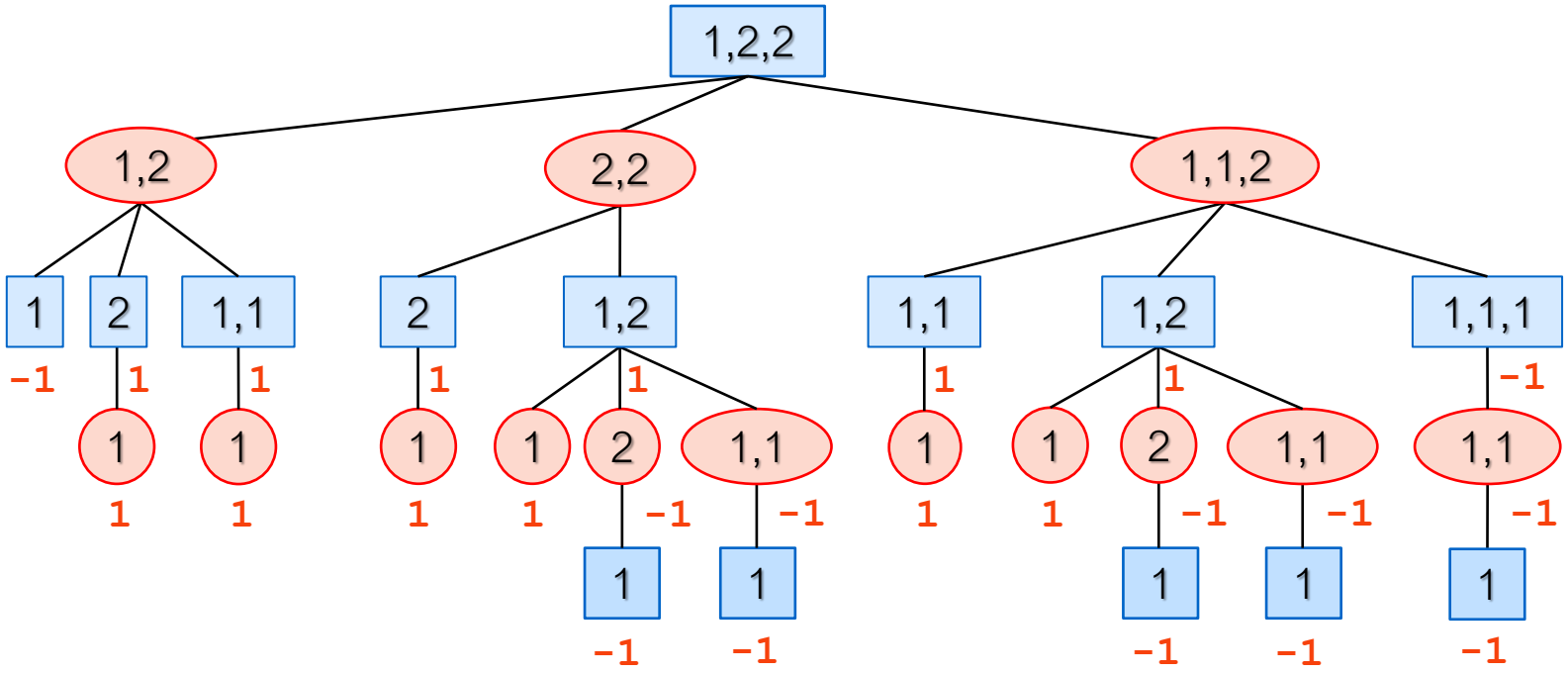
MAX (agent)

MIN (opponent)

MAX (agent)

MIN (opponent)

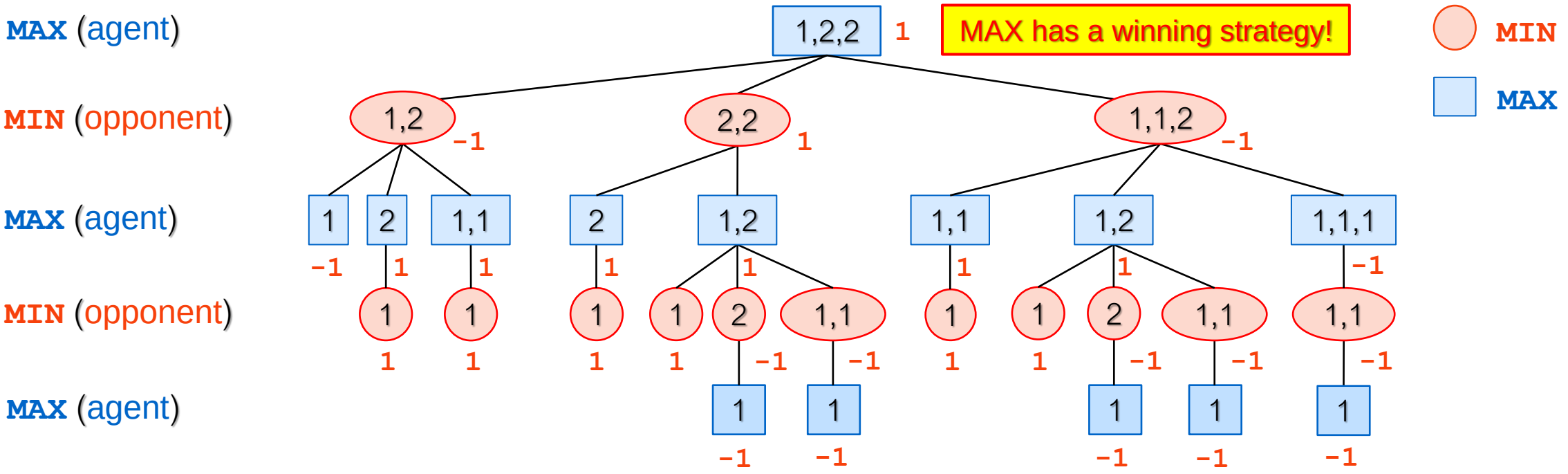
MAX (agent)



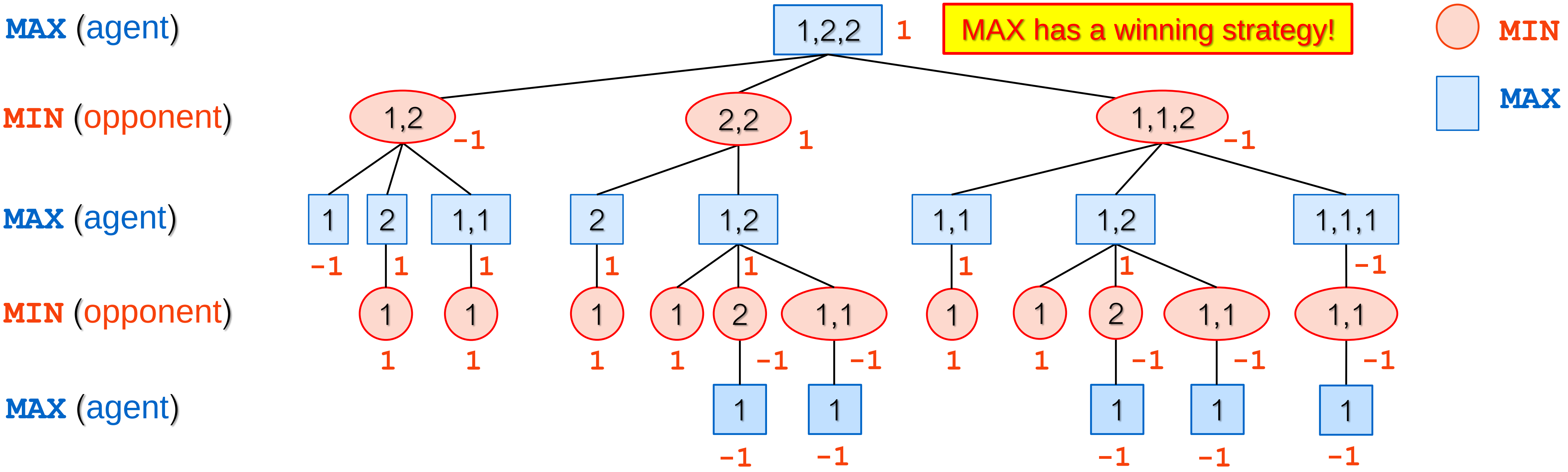
MIN

MAX

Game of NIM: (1,2,2) Game Tree



Game of NIM: (1,2,2) Game Tree



This is the Backwards Induction process to propagate utility values

DFS

Returns a utility-optimising strategy (assuming the opponent also plays optimally)

3 Strategies: Optimal Decisions via Minimax

Player Strategies

- A strategy `s` for player `i`:
  - What will player `i` do at every node of the game tree that they make a move in?
    - Need to specify behaviour in states that may never be reached!

- Winning strategy

A strategy s_1^* for Player 1 is denoted a winning strategy if
for any strategy s_2 by Player 2, the game ends with Player 1 as the winner.

- Non-losing strategy

A strategy t_1^* for Player 1 is denoted a non-losing strategy if
for any strategy s_2 by Player 2, the game ends in a draw or with Player 1 as the winner.

Optimal Strategy at each Node - Minimax

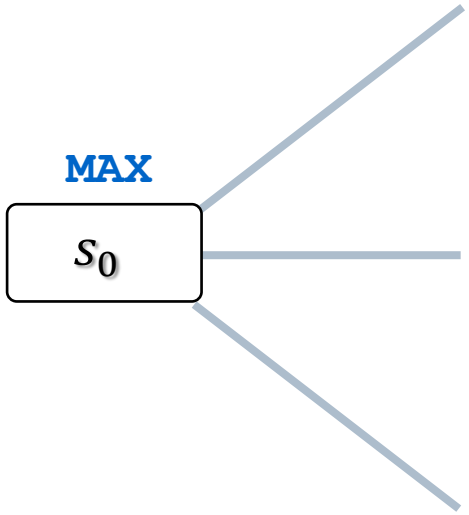
$$Minimax(s) = \begin{cases} \text{Utility}(s, \text{MAX}) \text{ if Is}-\text{Terminal}(s) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s,a)\big) \text{ if To}-\text{Move}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s,a)\big) \text{ if To}-\text{Move}(s) = \text{MIN} \end{cases}$$

Remember that **`Result(s,a)`** outputs the successor state (given **`a`** taken at **`s`**)
(i.e., **`MIN`** wants state with lowest utility, while **`MAX`** wants state with highest)

- Intuitively
  - **`MAX`** chooses move to maximise utility given the successor options provided by **`MIN`**
  - **`MIN`** chooses move to minimise utility given the successor options provided by **`MAX`**
  - Minimax determines a strategy – i.e., determines a game plan

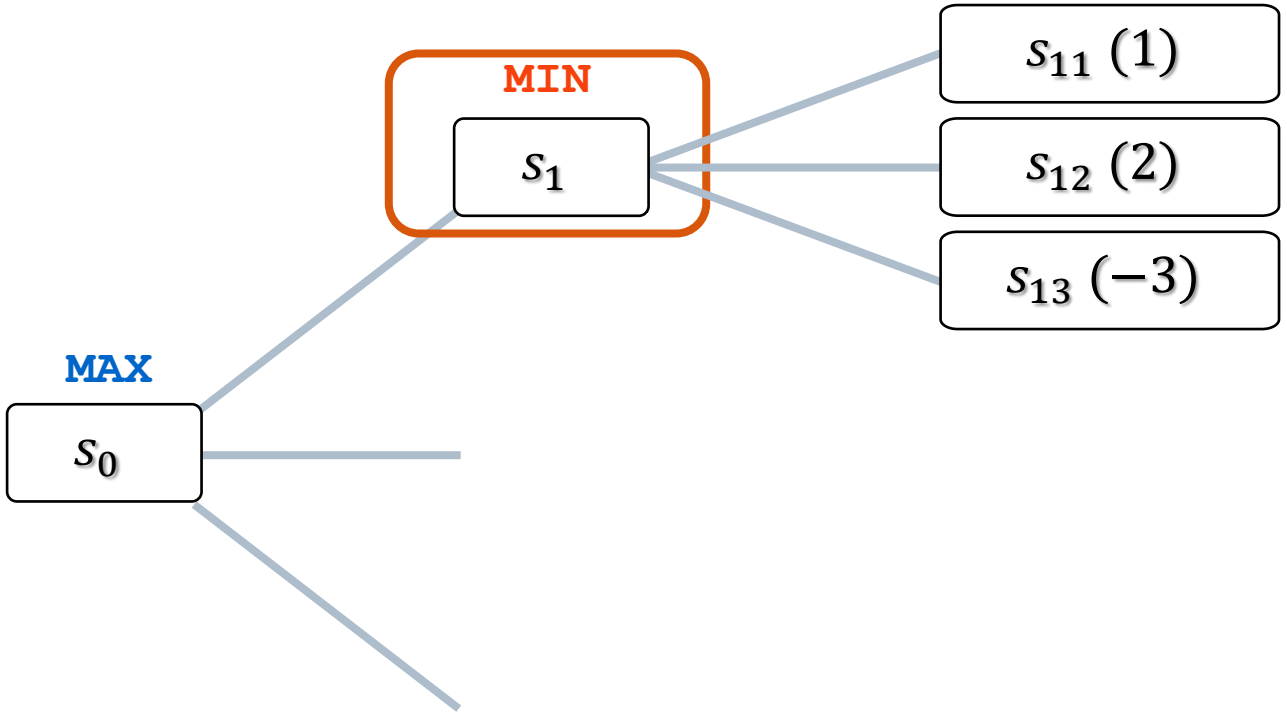
Minimax Play

Backwards Induction



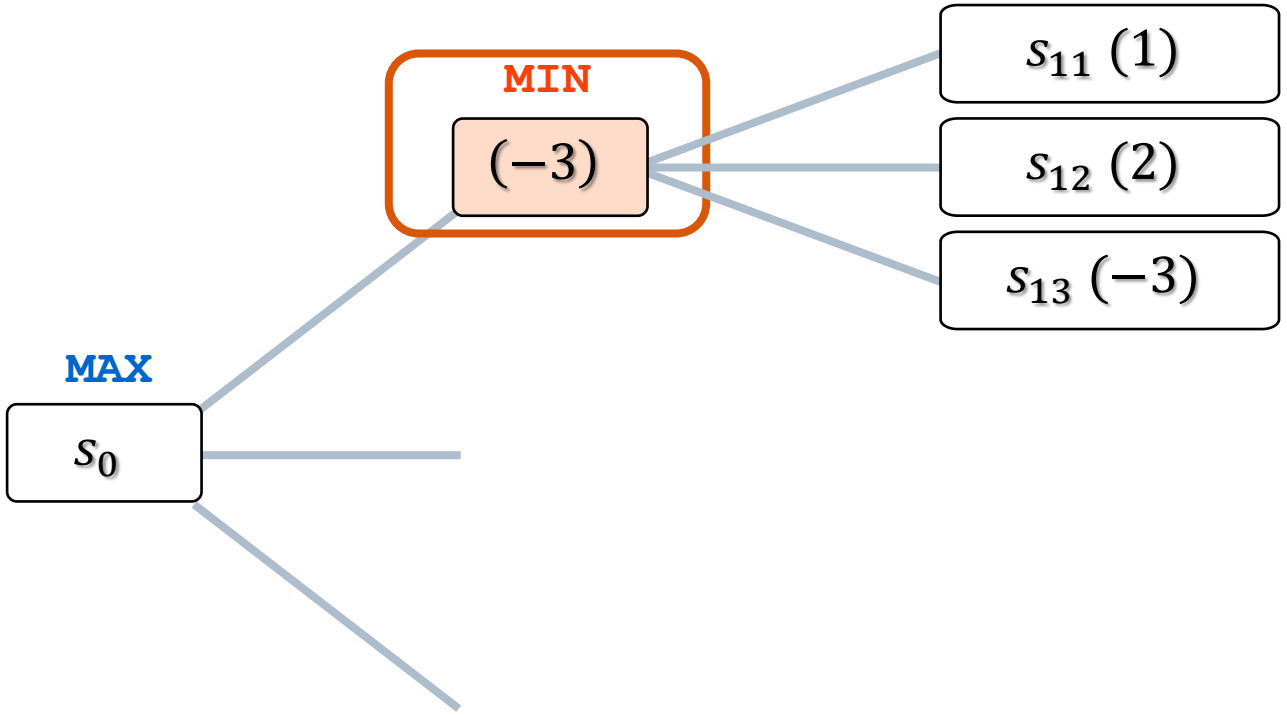
Minimax Play

Backwards Induction



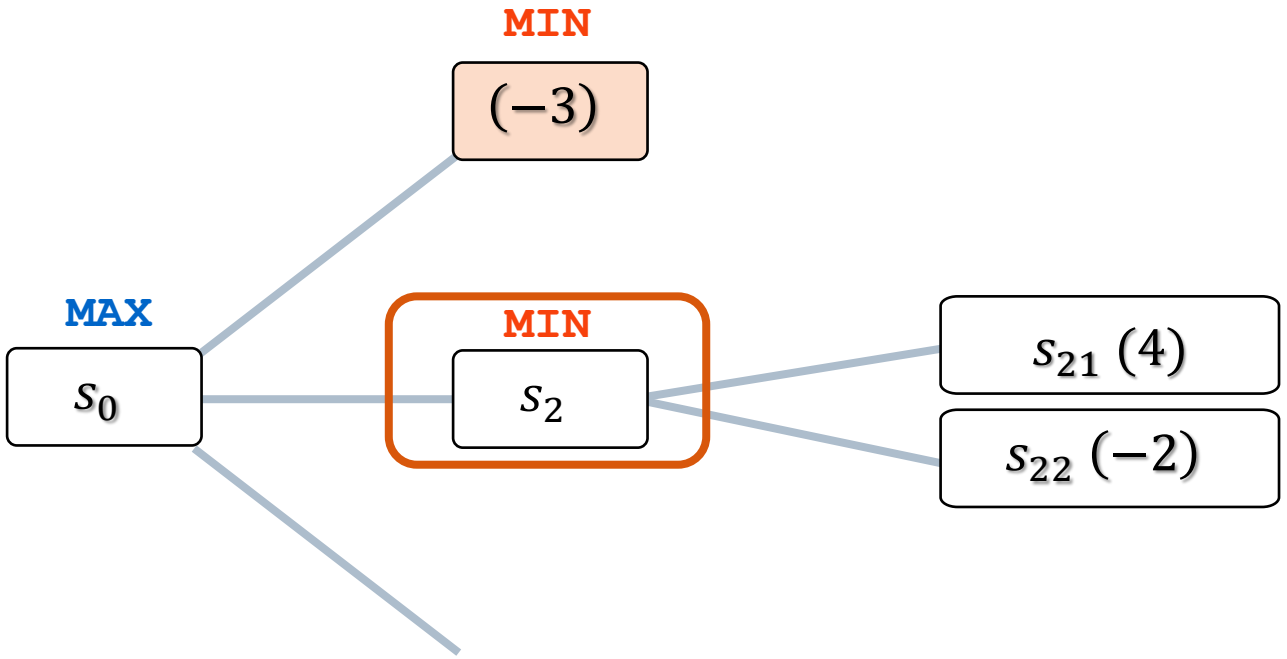
Minimax Play

Backwards Induction



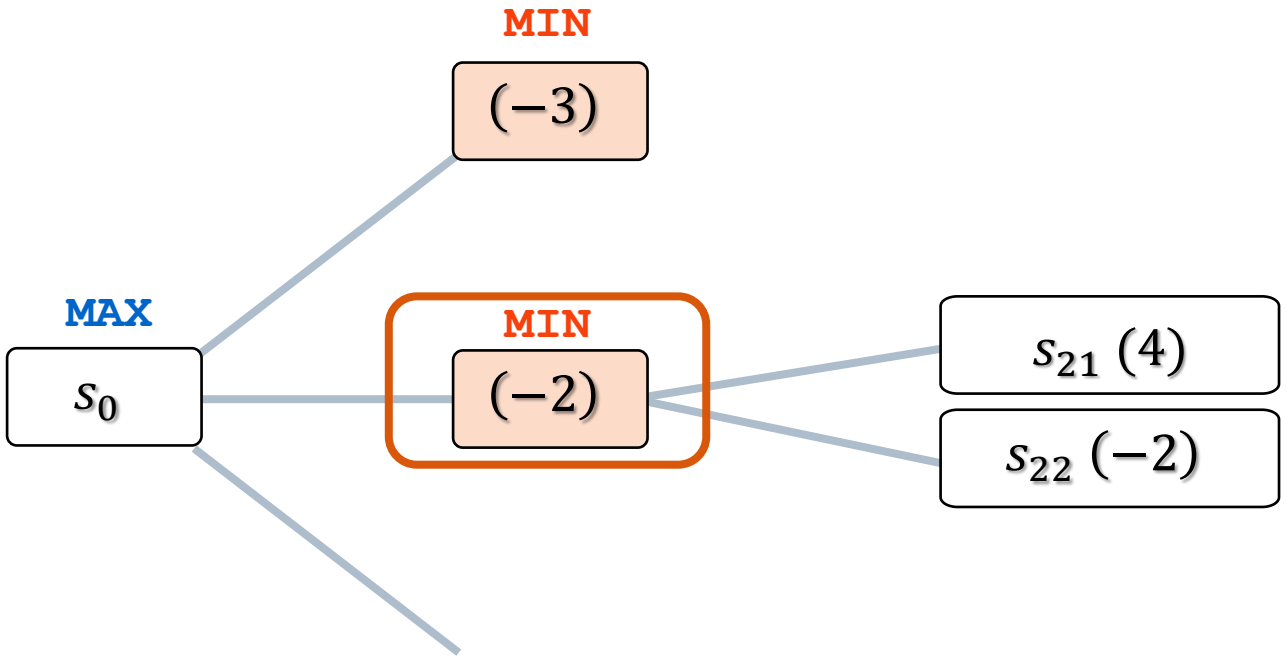
Minimax Play

Backwards Induction



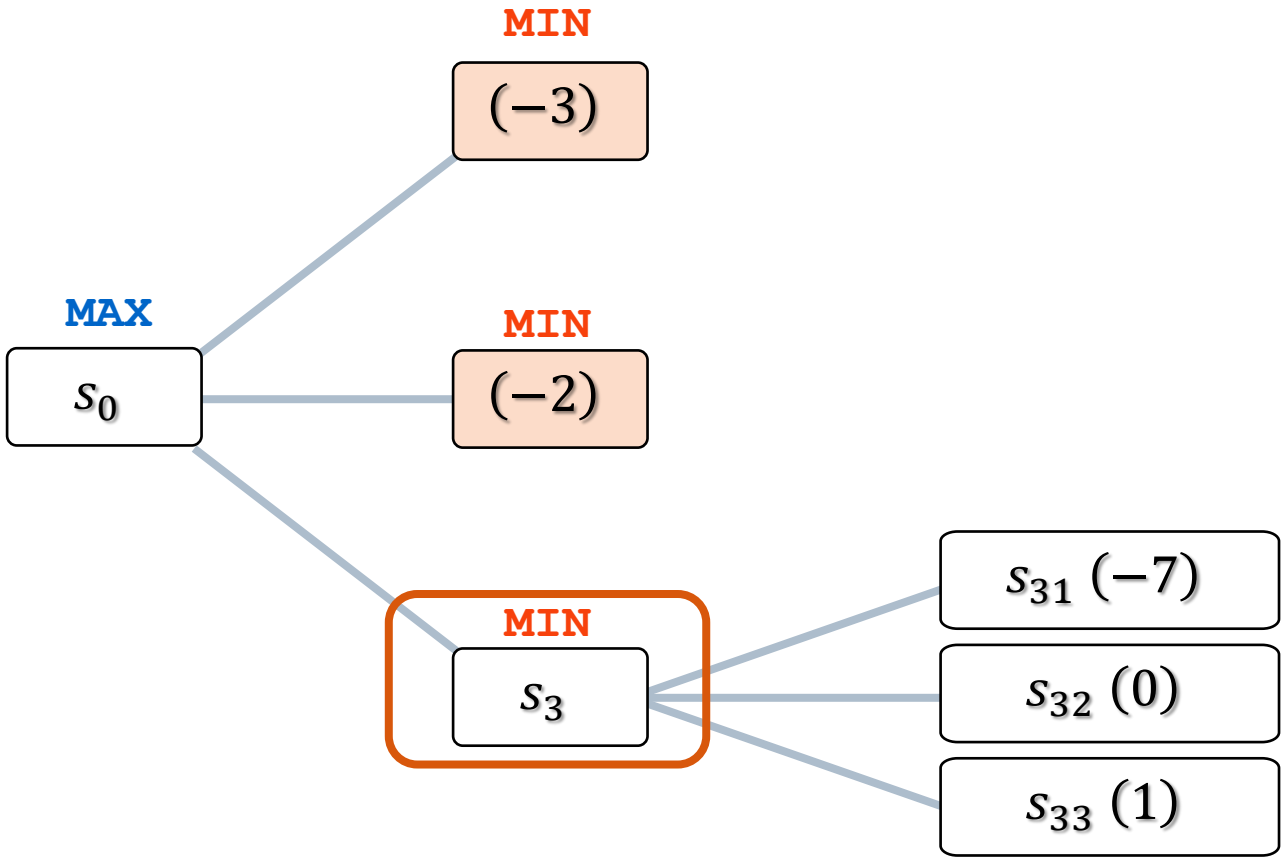
Minimax Play

Backwards Induction



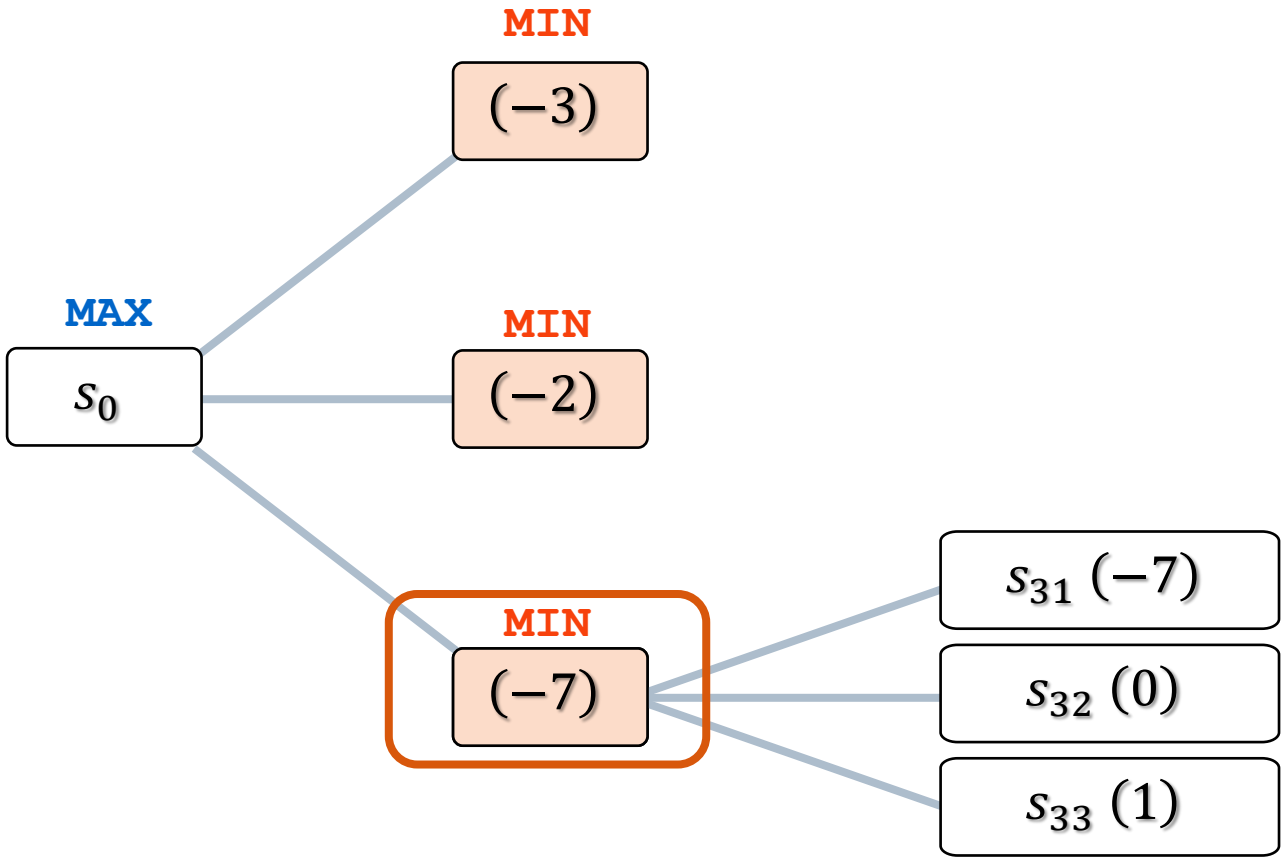
Minimax Play

Backwards Induction



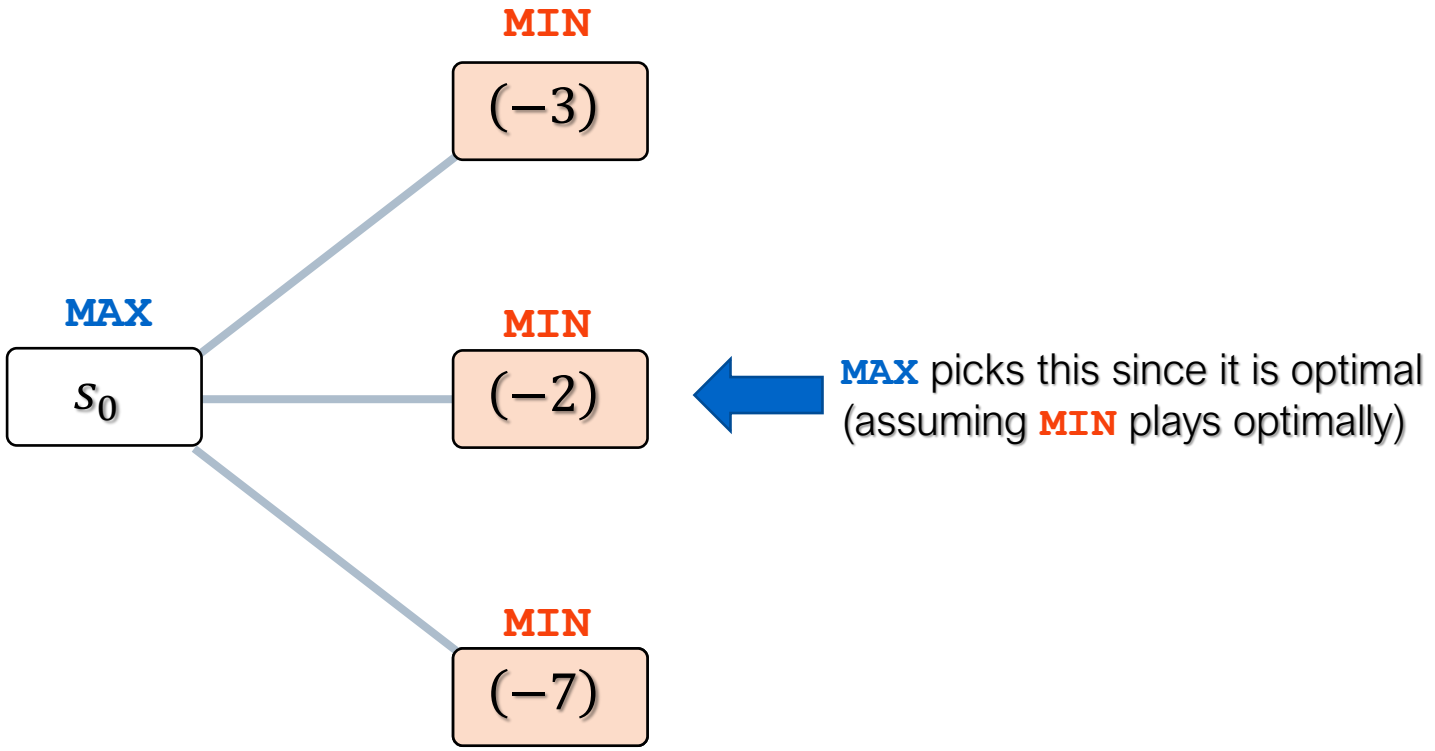
Minimax Play

Backwards Induction



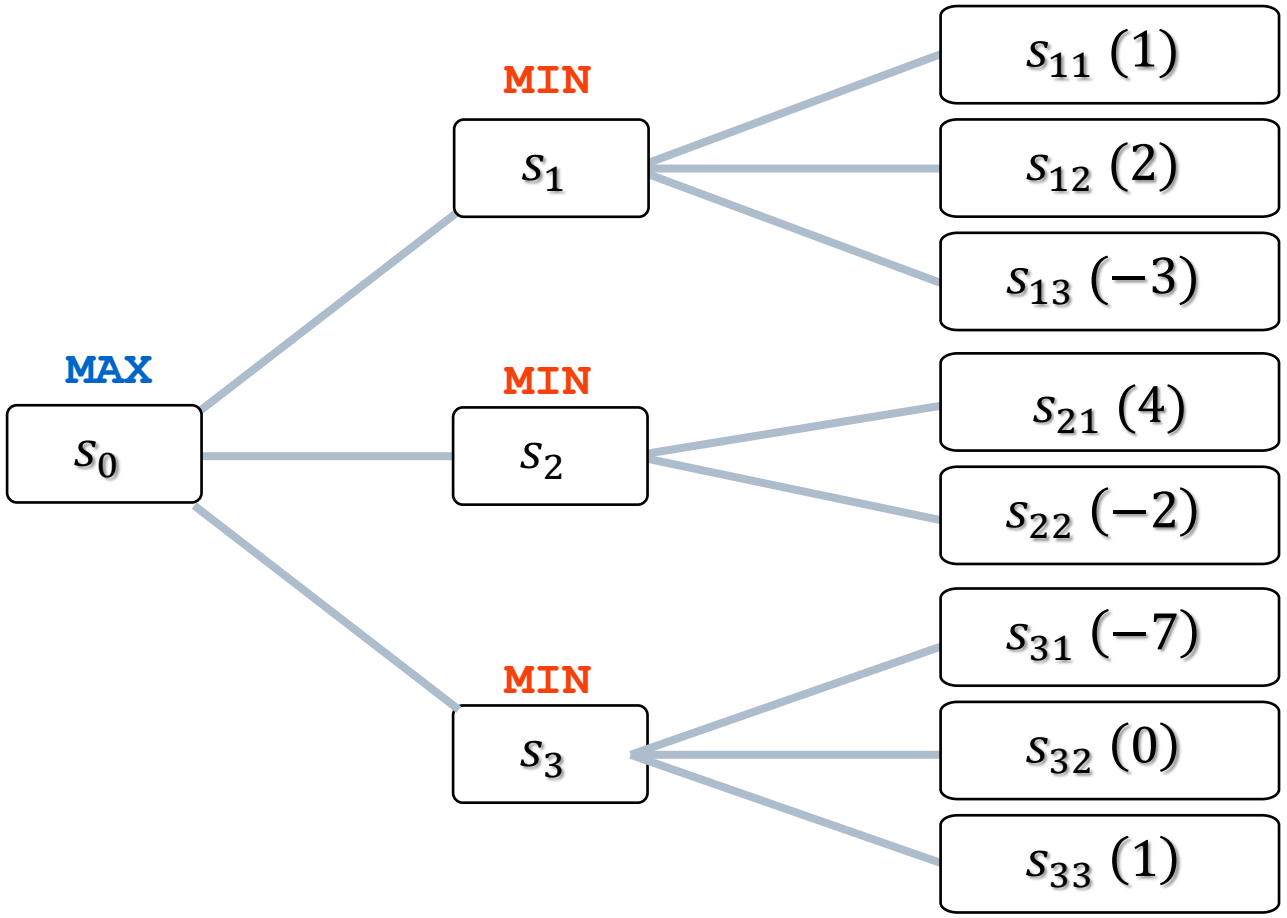
Minimax Play

Backwards Induction



Minimax Play

What are the optimal strategies in this game?

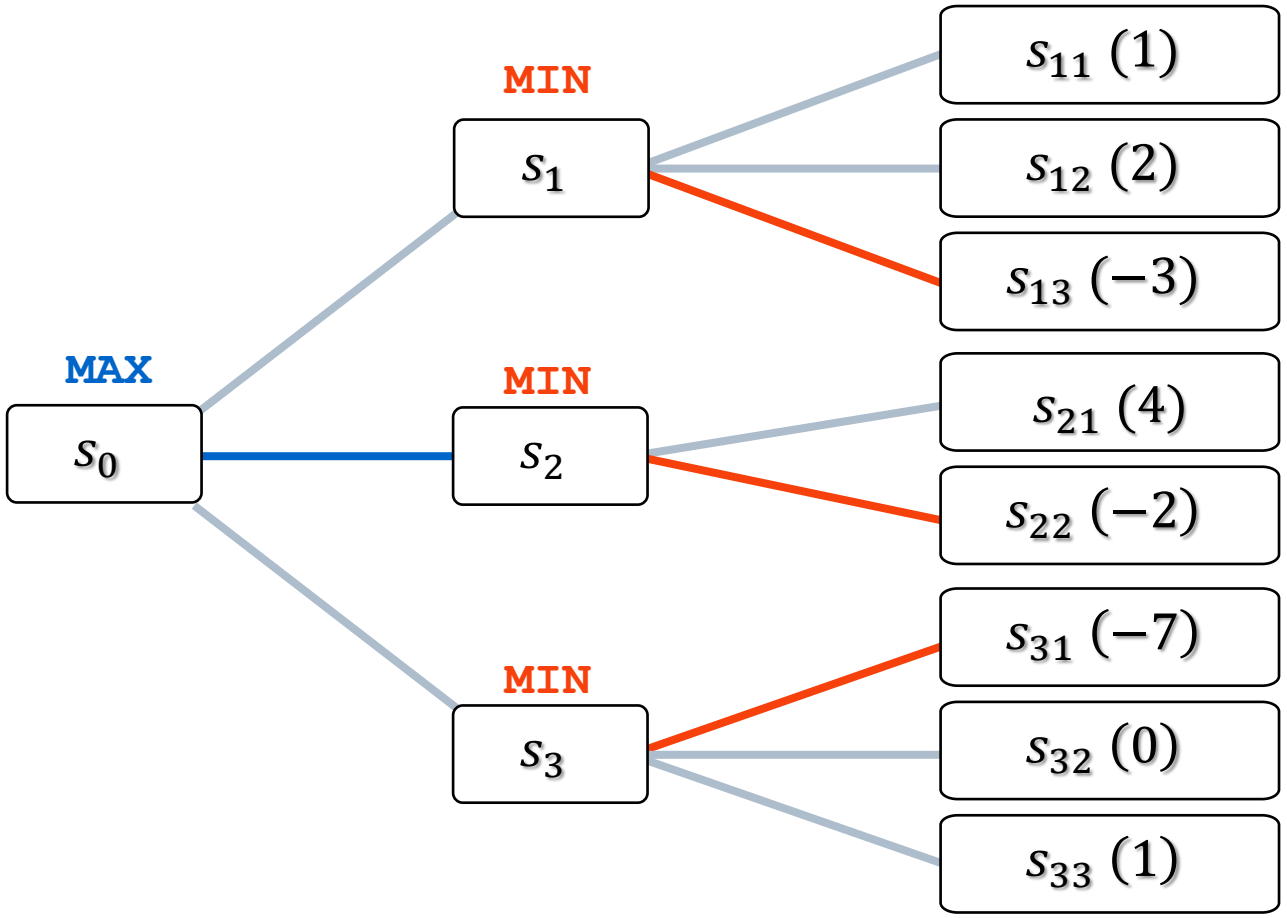


Minimax Play

What are the optimal strategies in this game?

MIN Strategy

MAX Strategy



Minimax Algorithm Properties

- Complete?
  - Yes (assuming finite game tree)
- Optimal?
  - Yes (optimal gameplay)
    - Assuming opponent plays optimally
- Time
  - $O(b^m)$
- Space
  - $O(bm)$

- Returns a Subgame Perfect Nash Equilibrium*
  - i.e., the best action at every node

  * Not examinable

- Minimax runs in time polynomial to tree size

Are we done?

Backwards Induction Issue

- Game trees are massive
  - Chess has a massive game tree
    - 10^{123} nodes
  - In comparison, planet Earth has about 10^{50} atoms ...

- Impossible to expand the entire tree

- Must find ways to shrink the search tree
  - We've seen this before
  - A common theme in search

Game tree complexities for other games: https://en.wikipedia.org/wiki/Game_complexity#Complexities_of_some_well-known_games

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/02185973212

4 | α-β Pruning

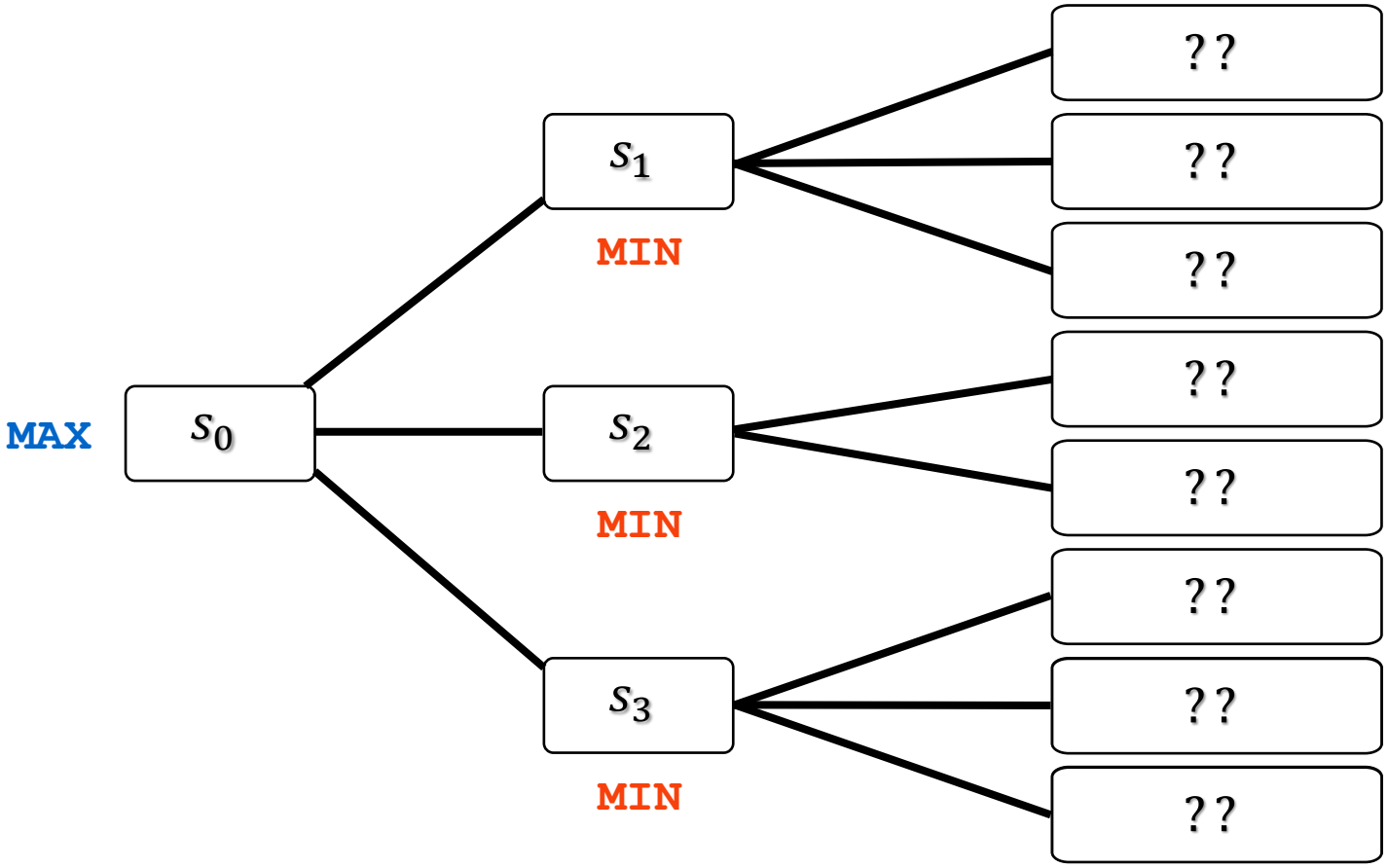
α-β Pruning

- General idea
  - Don't explore moves that would never be considered

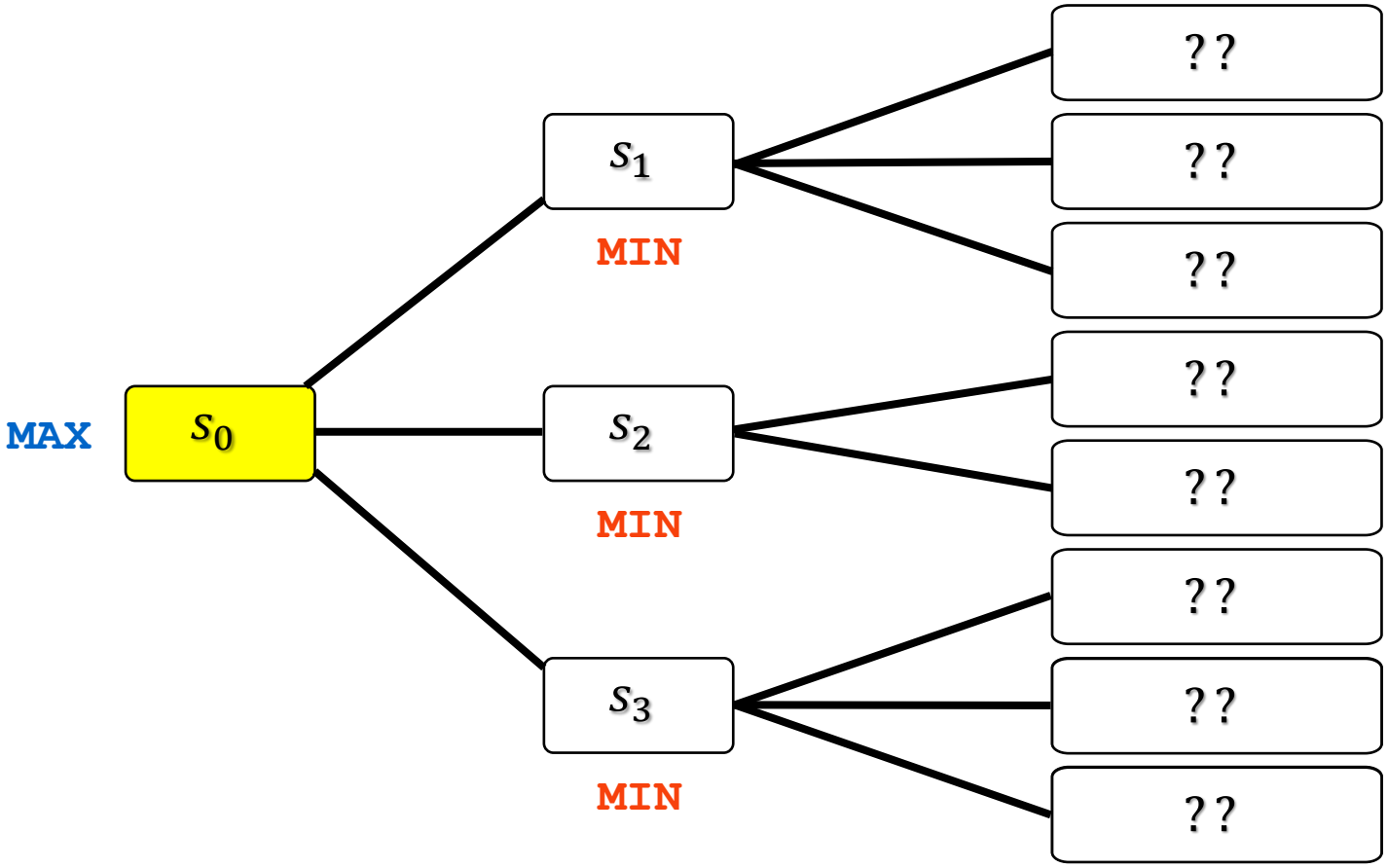
- Maintain bounds on values seen thus far while searching
  - α bounds `MAX`'s values
  - β bounds `MIN`'s values

- Prune subtrees that will never affect Minimax decision

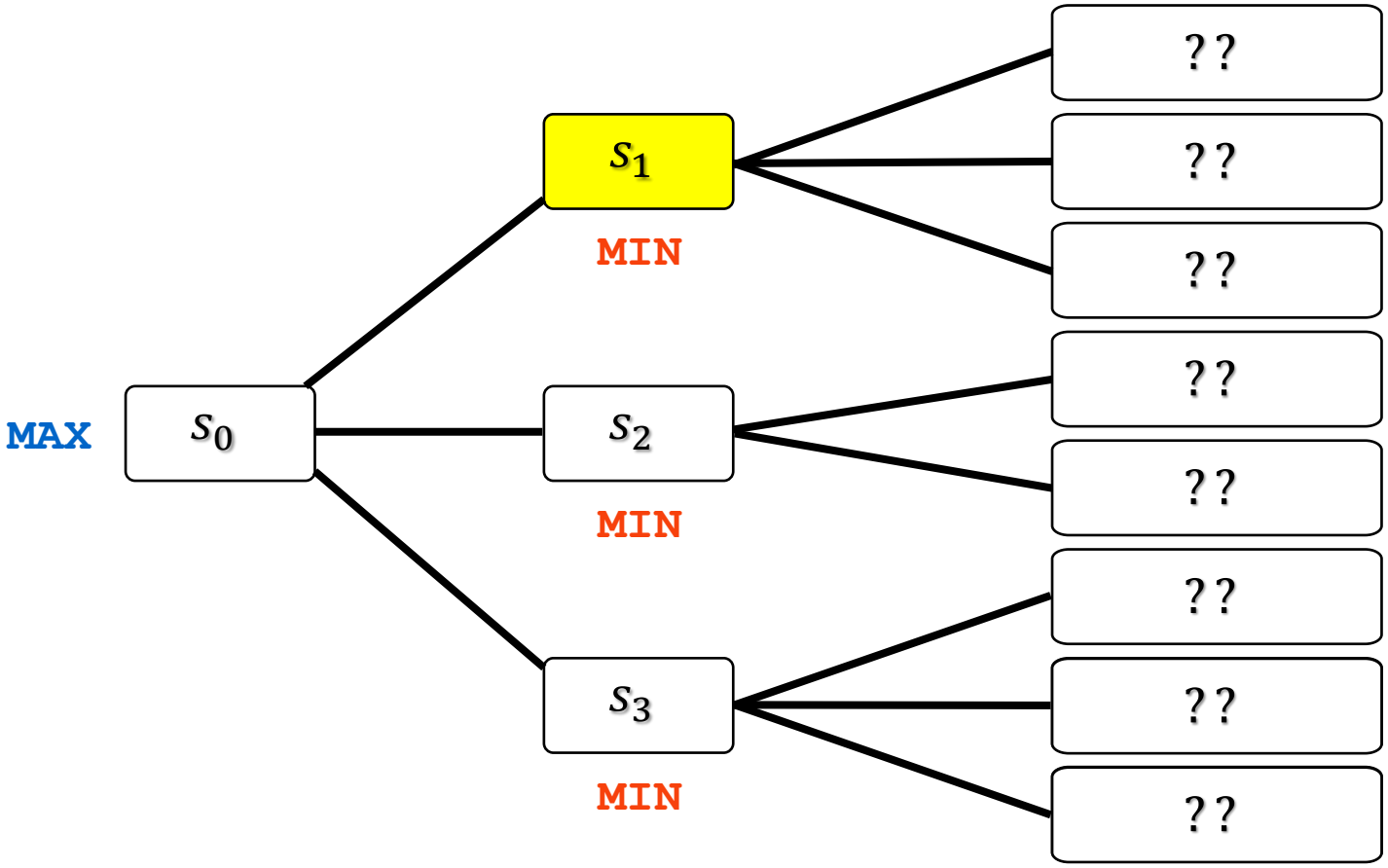
α-β Pruning – Example Trace



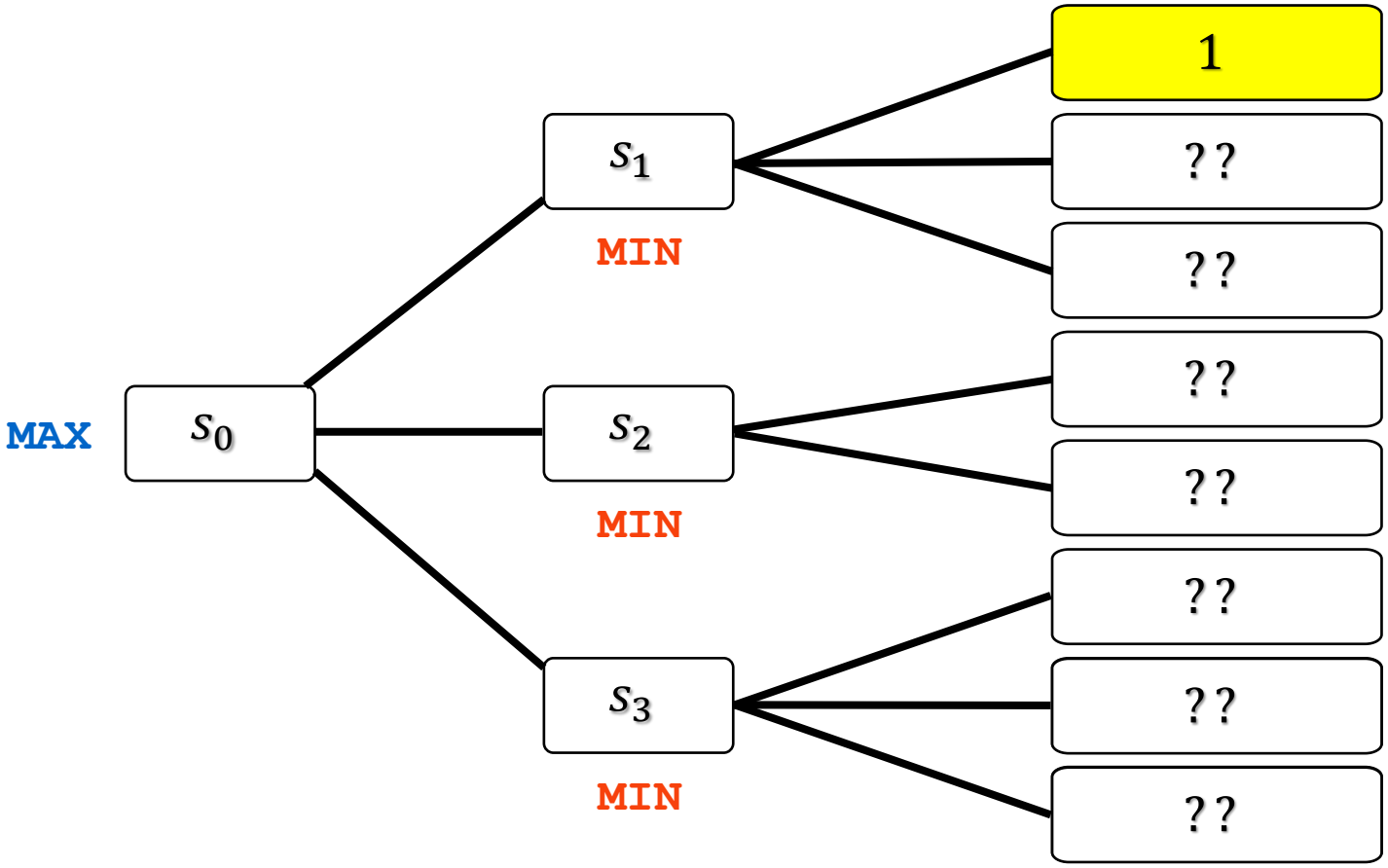
α-β Pruning – Example Trace



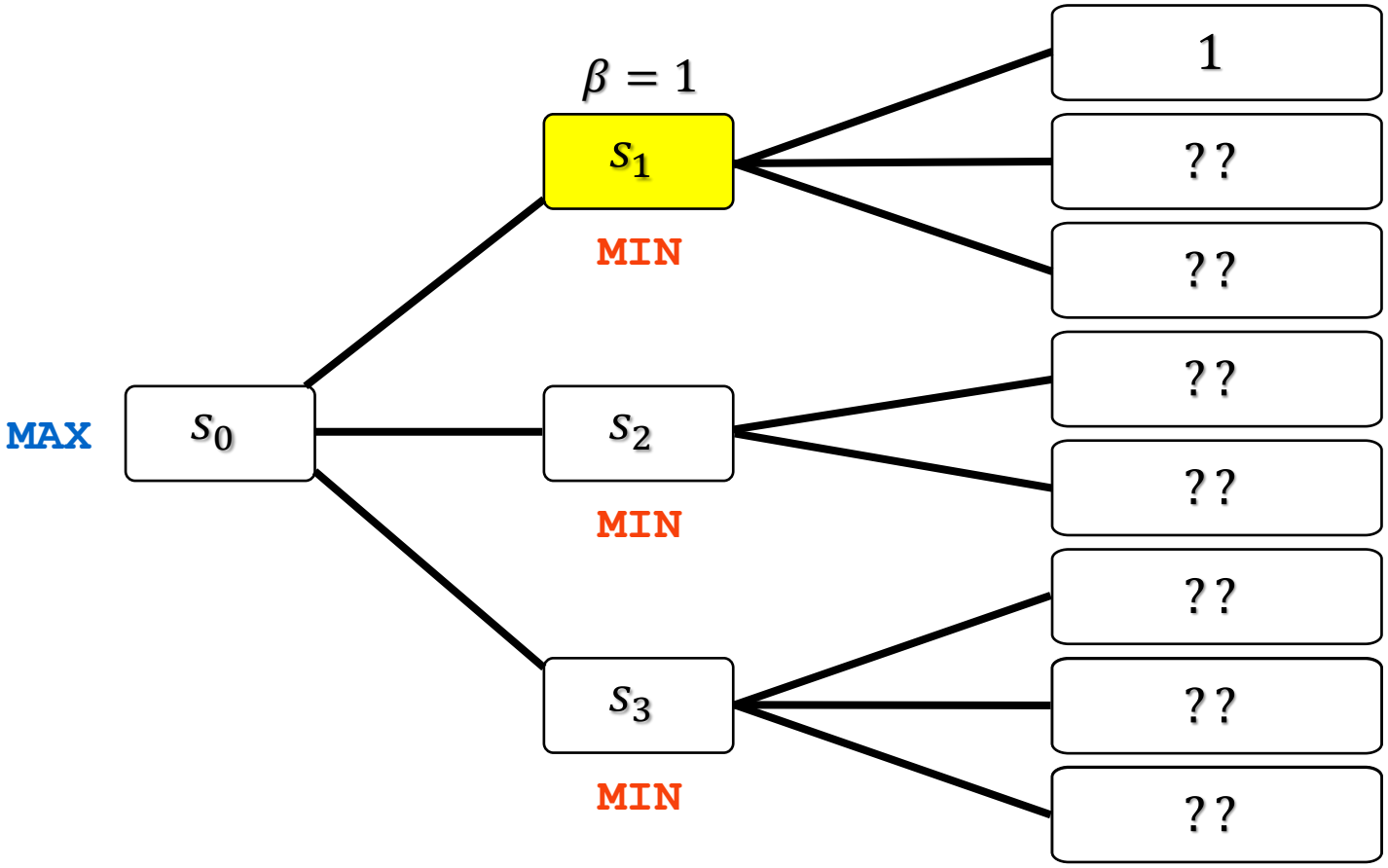
α-β Pruning – Example Trace



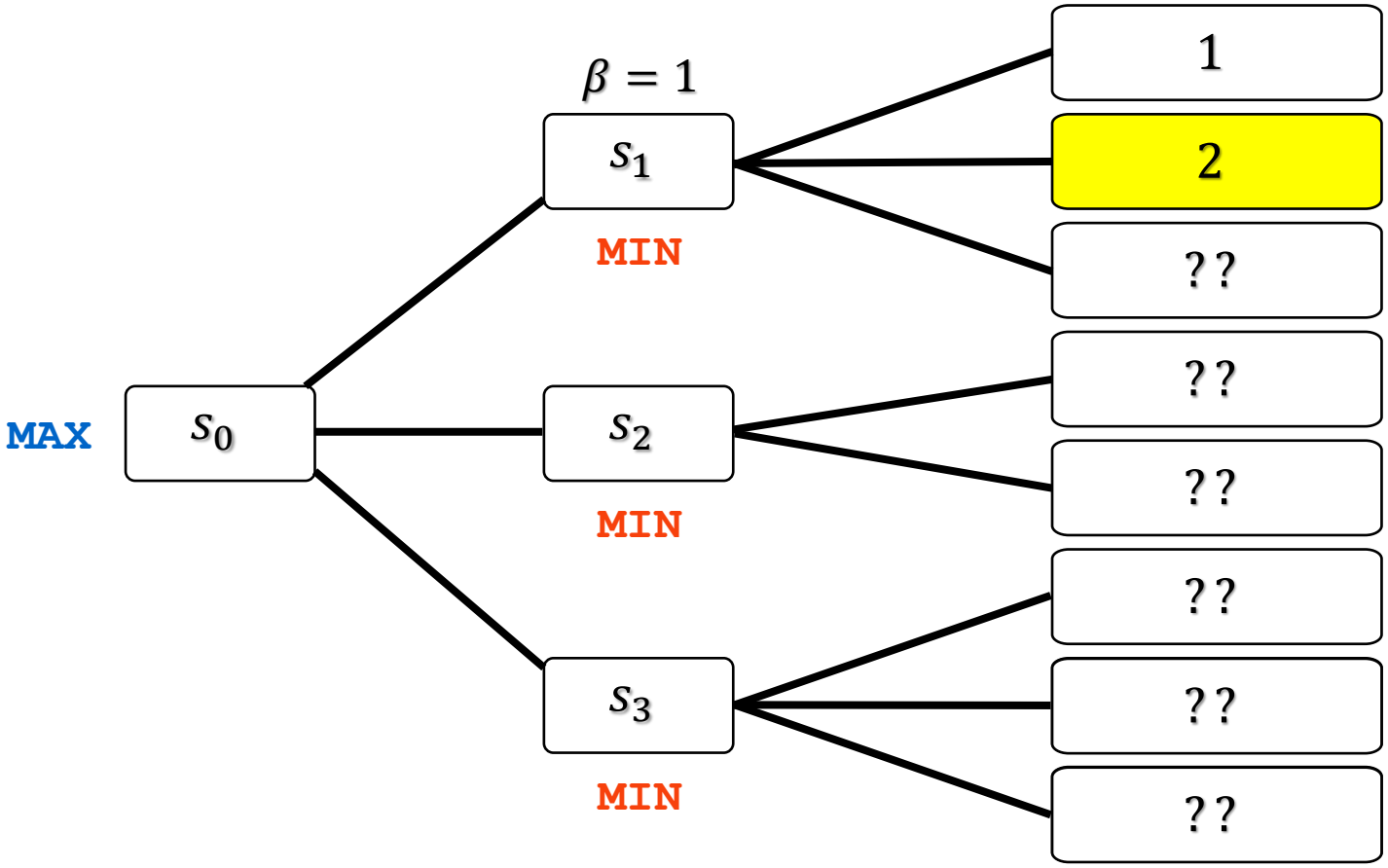
α-β Pruning – Example Trace



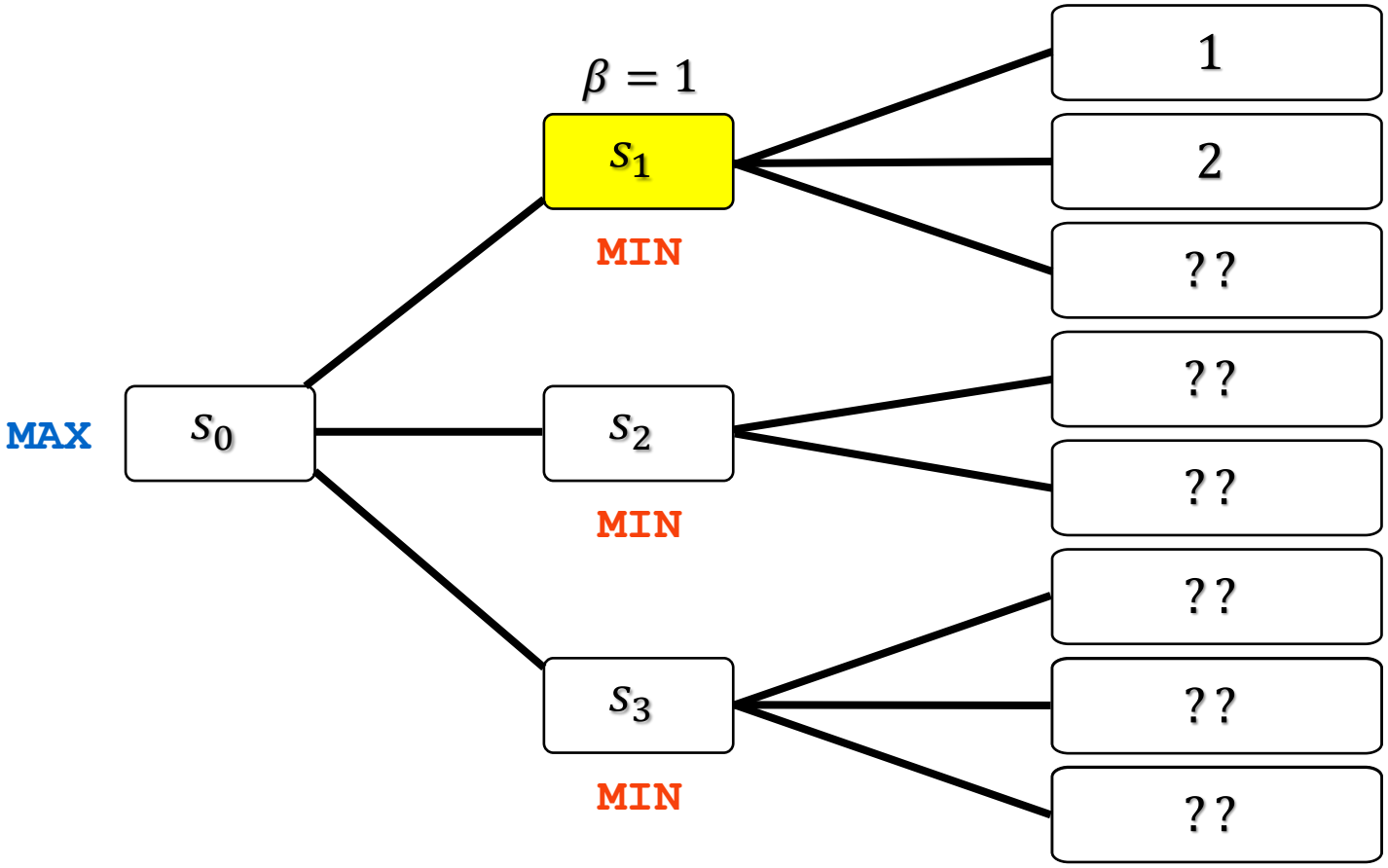
α-β Pruning – Example Trace



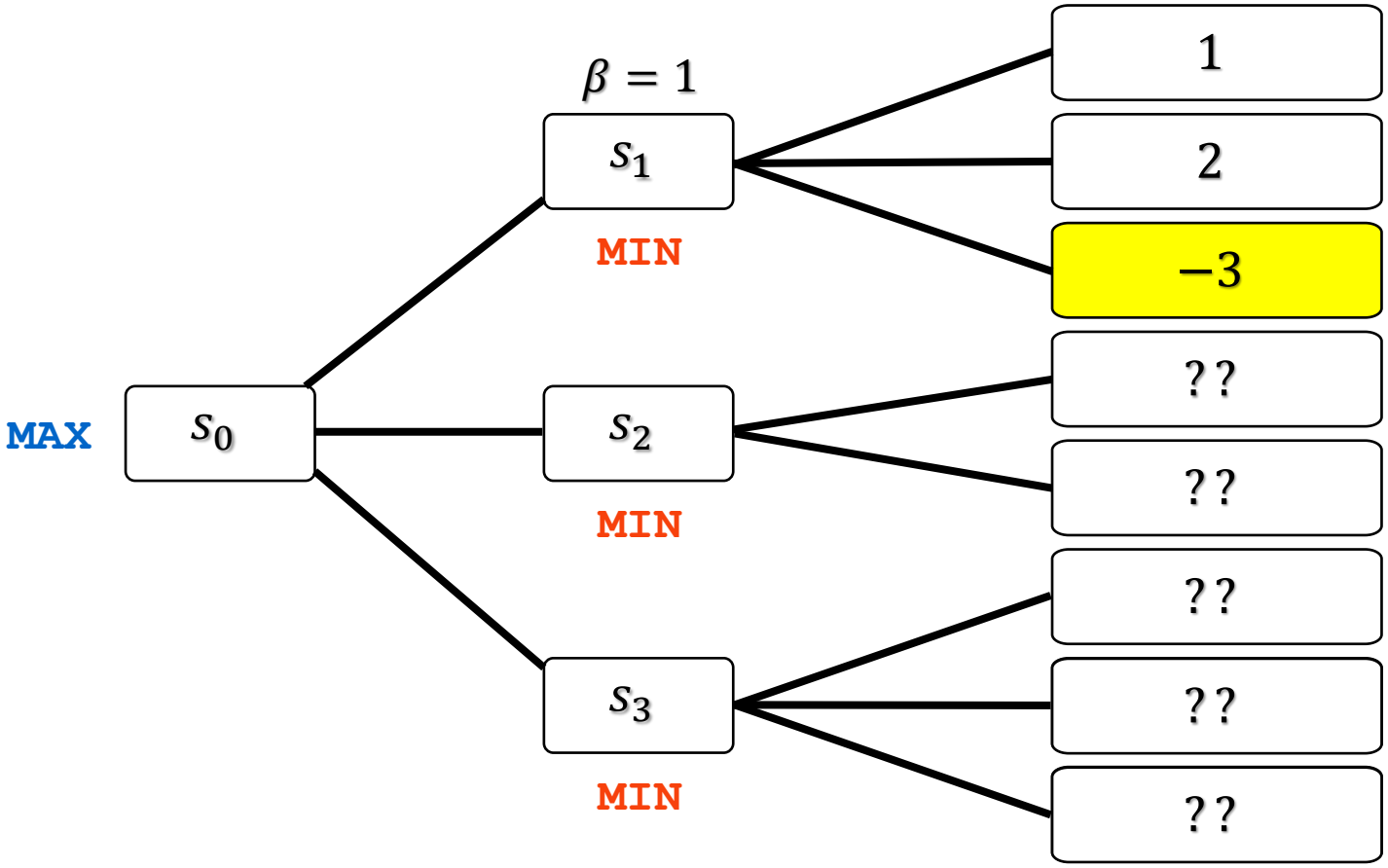
α-β Pruning – Example Trace



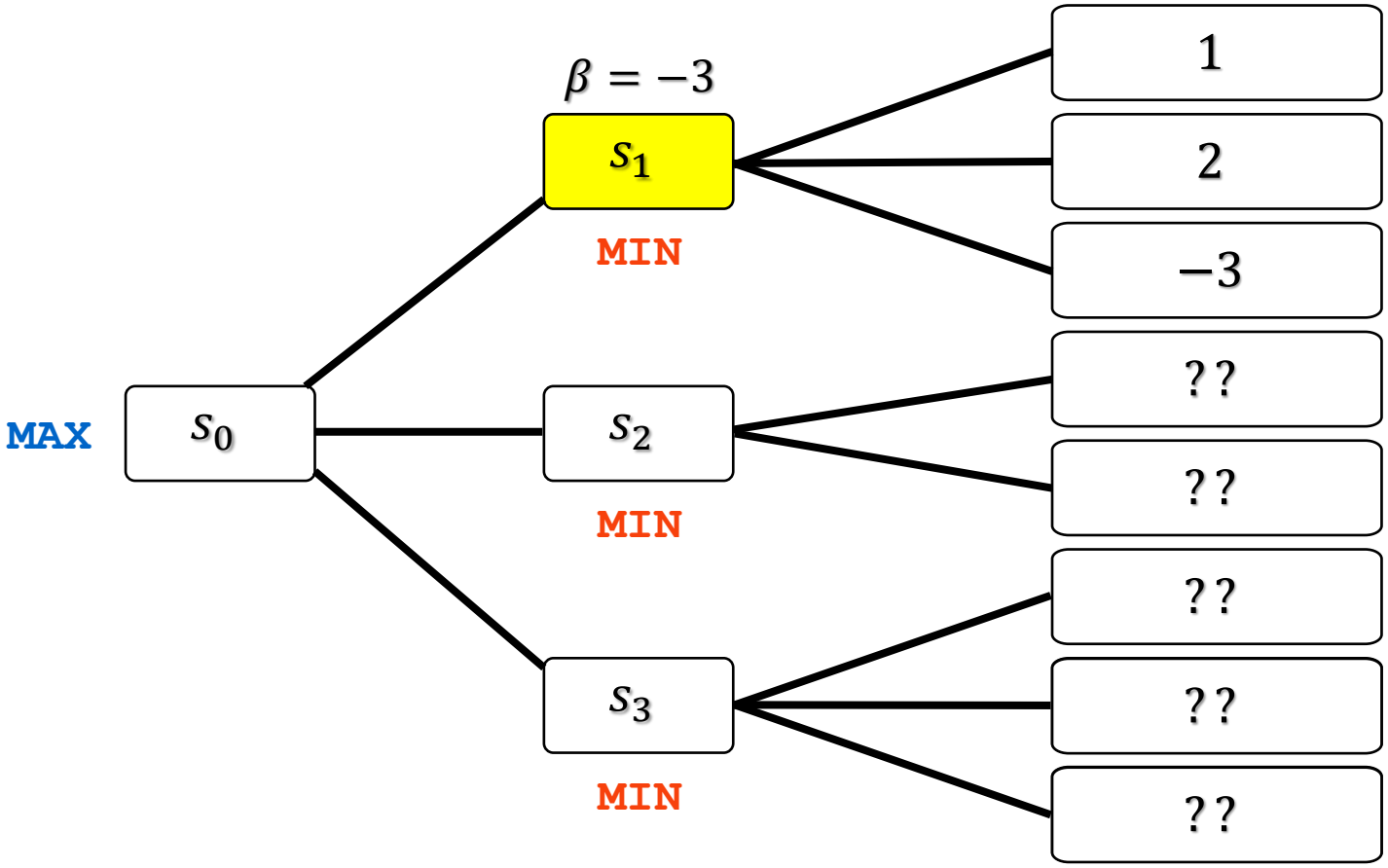
α-β Pruning – Example Trace



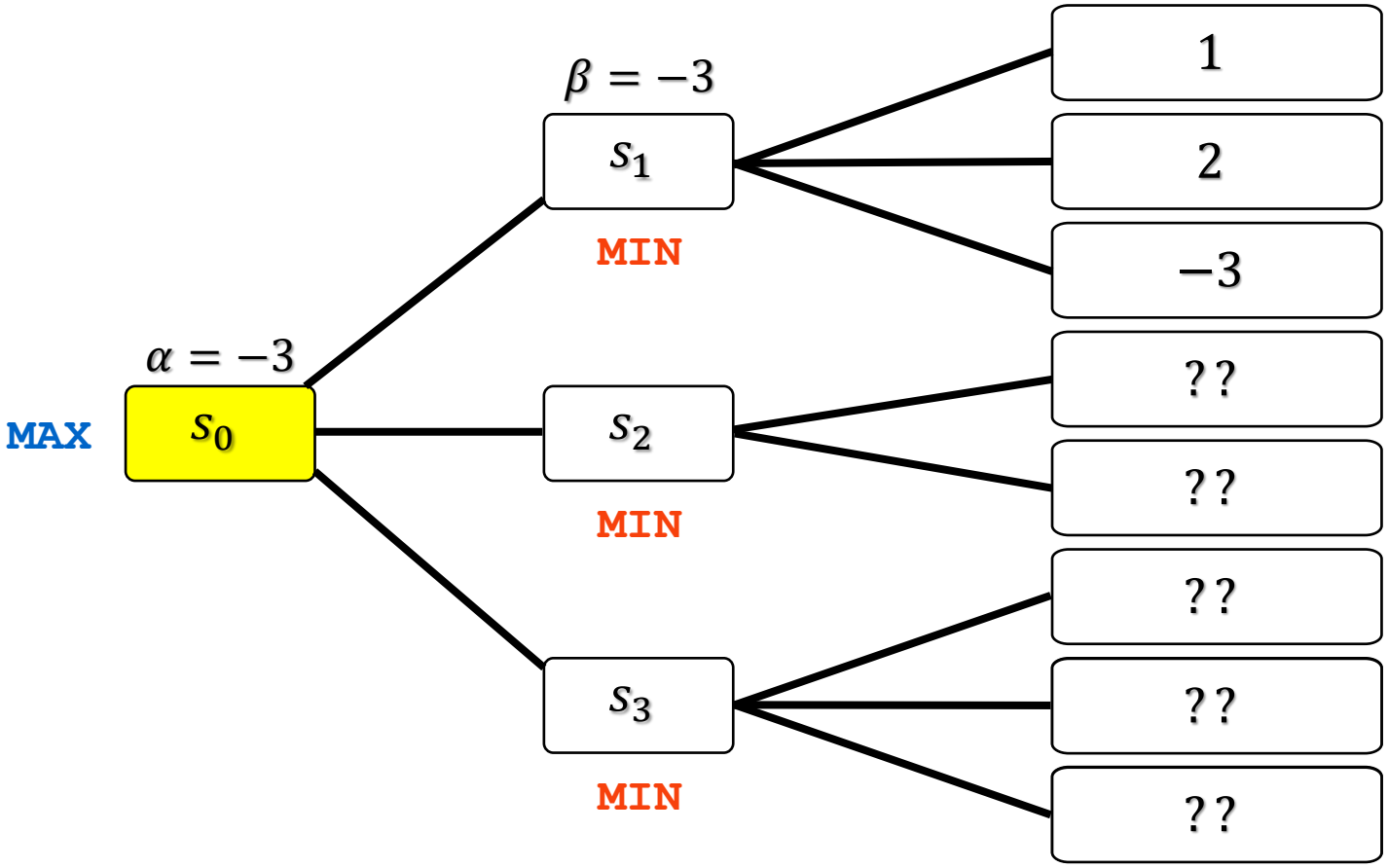
α-β Pruning – Example Trace



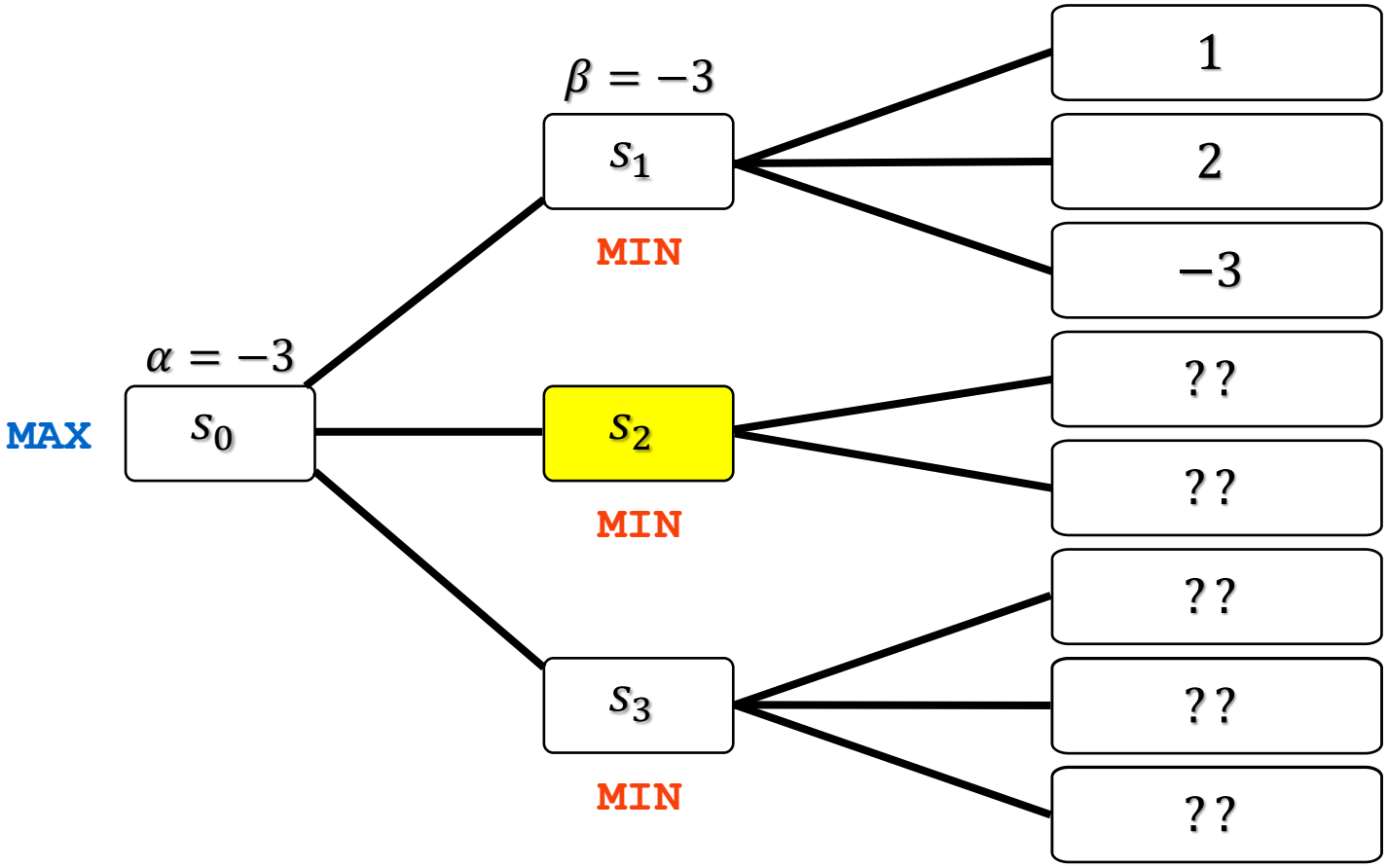
α-β Pruning – Example Trace



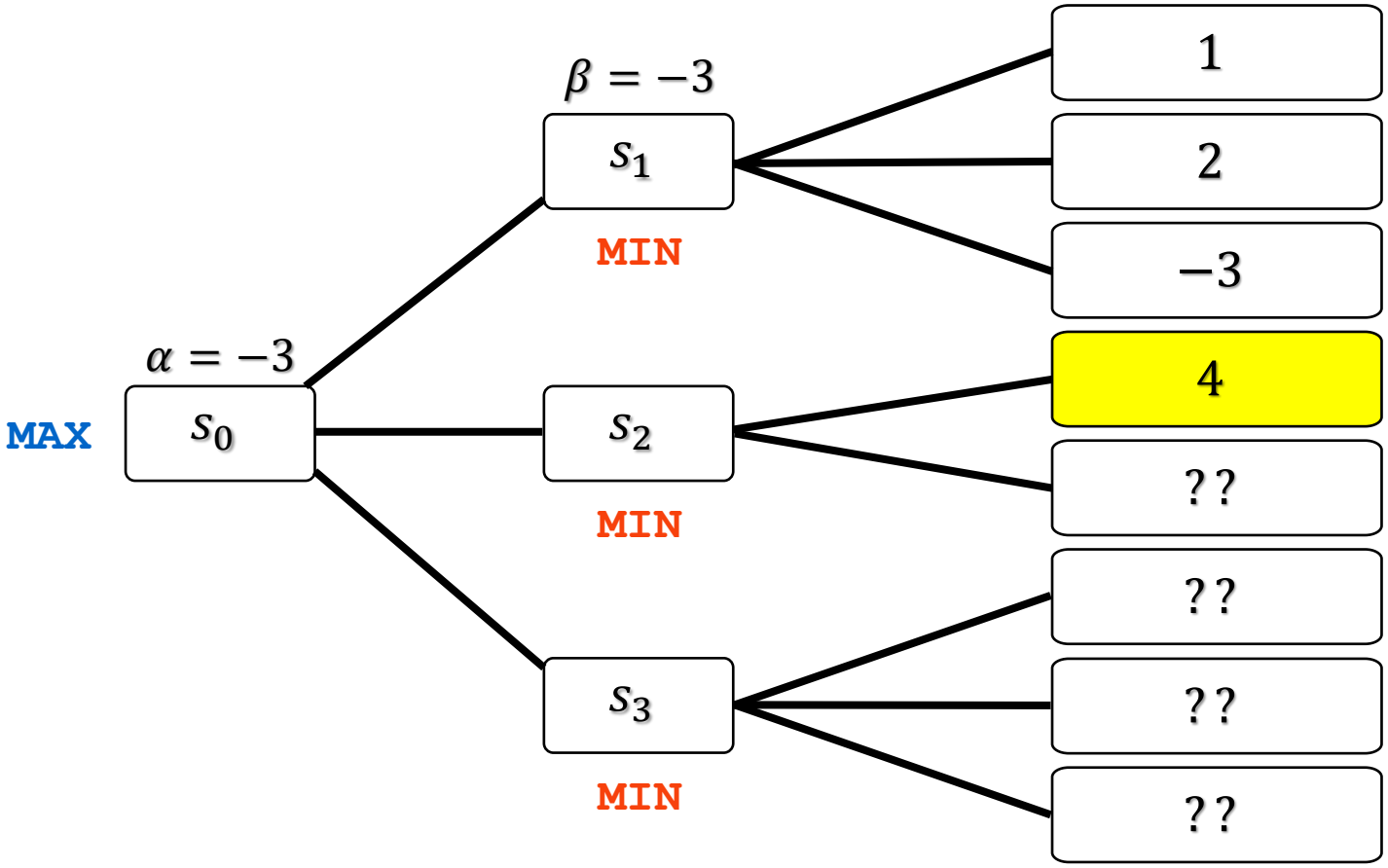
α-β Pruning – Example Trace



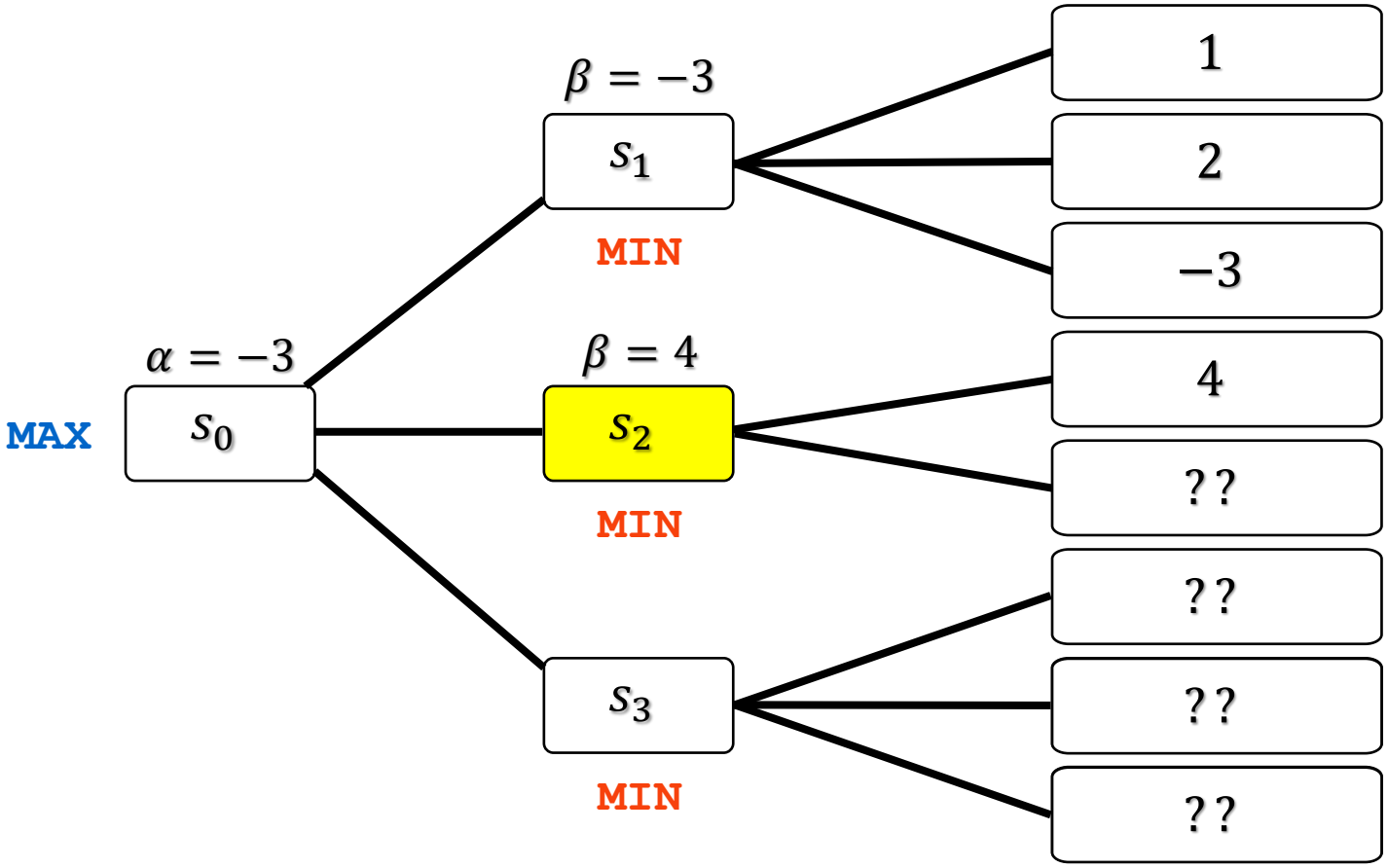
α-β Pruning – Example Trace



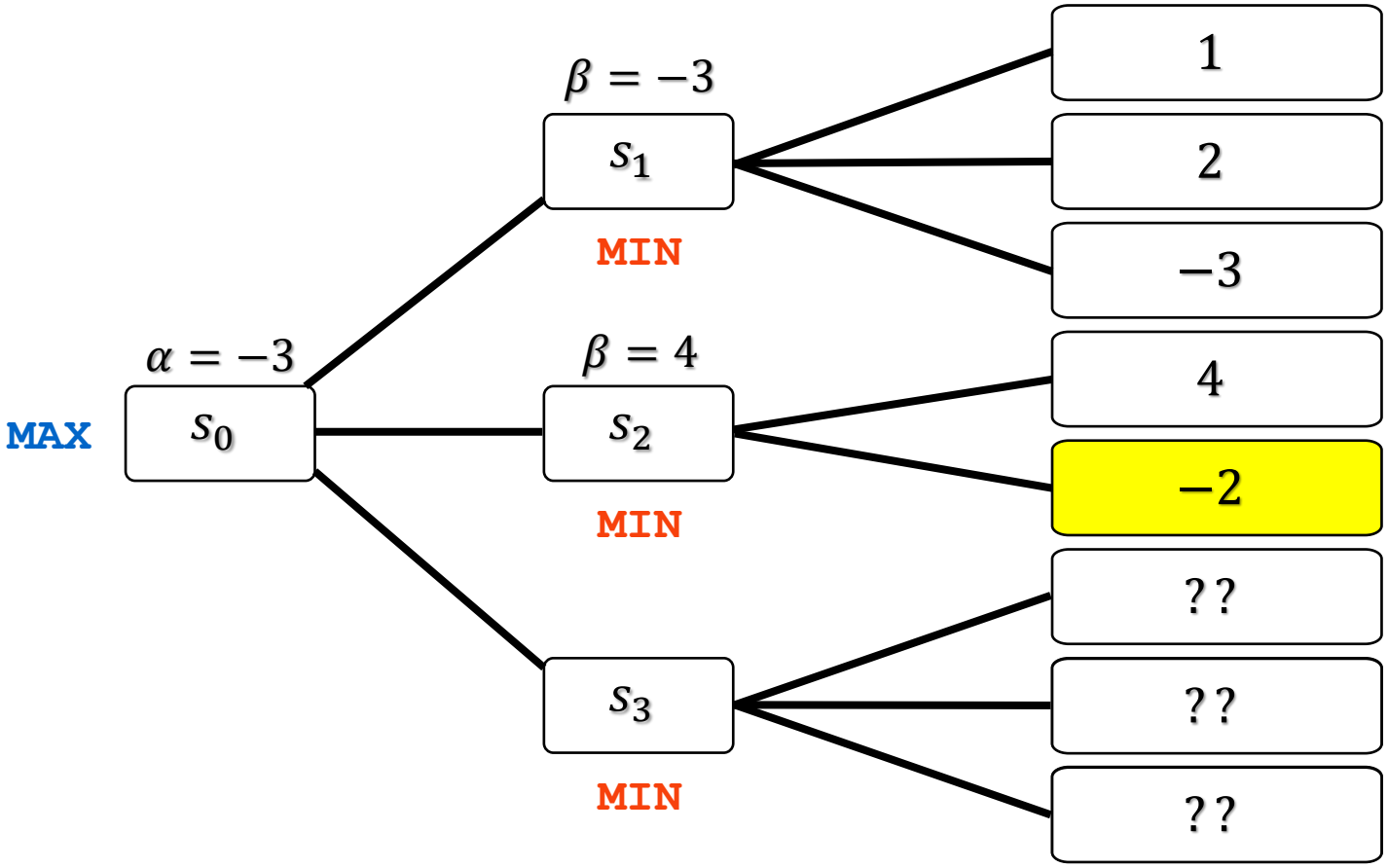
α-β Pruning – Example Trace



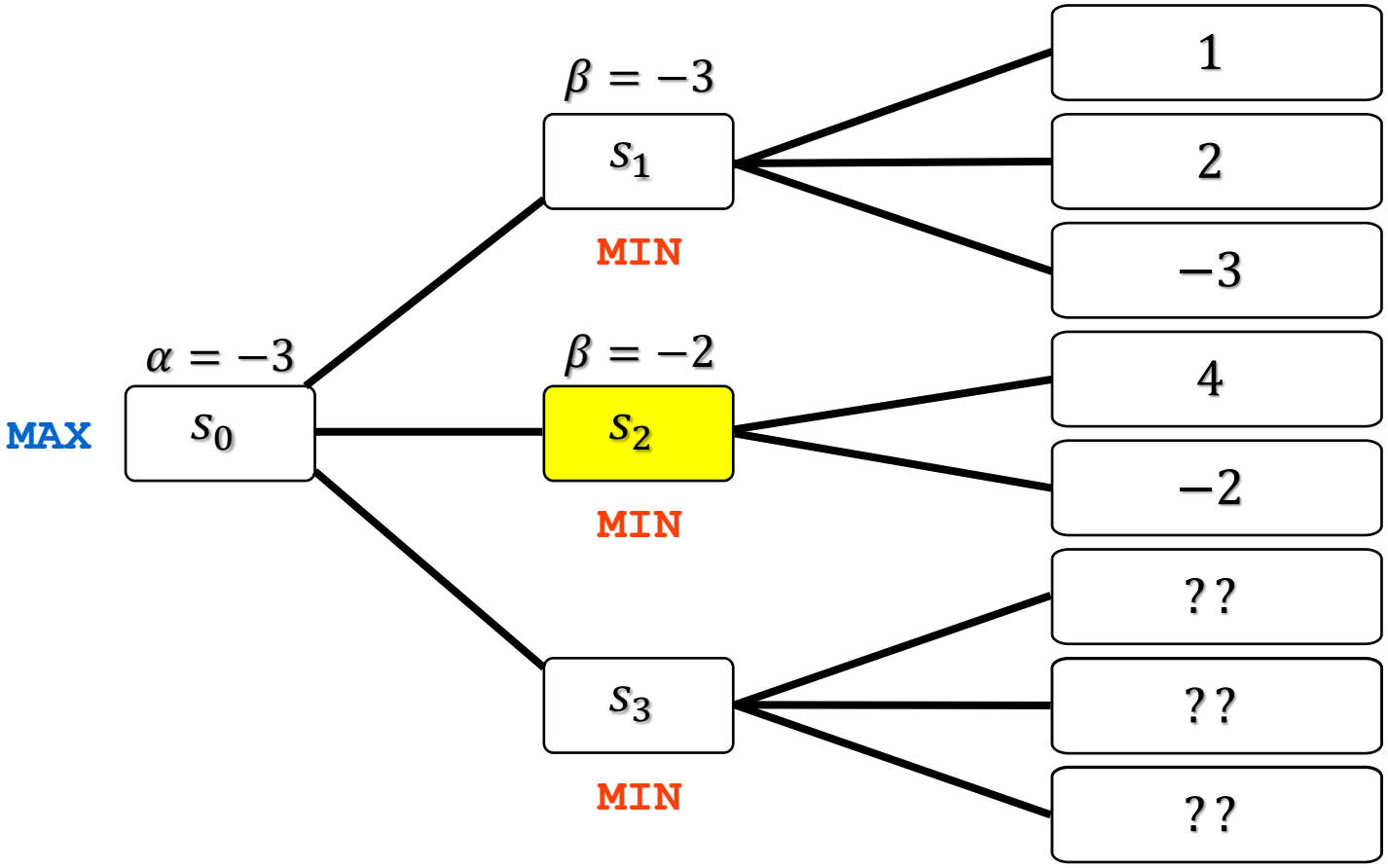
α-β Pruning – Example Trace



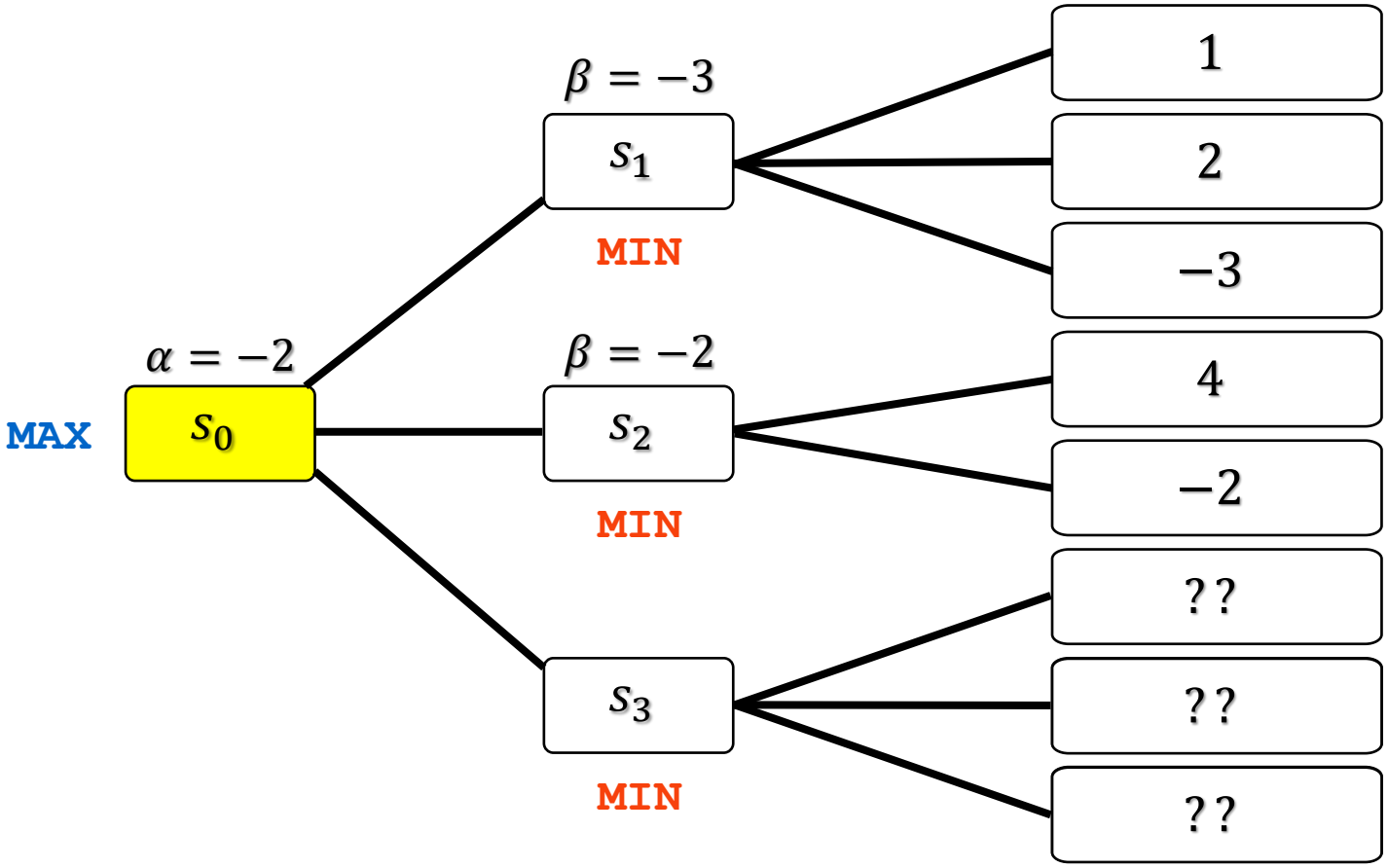
α-β Pruning – Example Trace



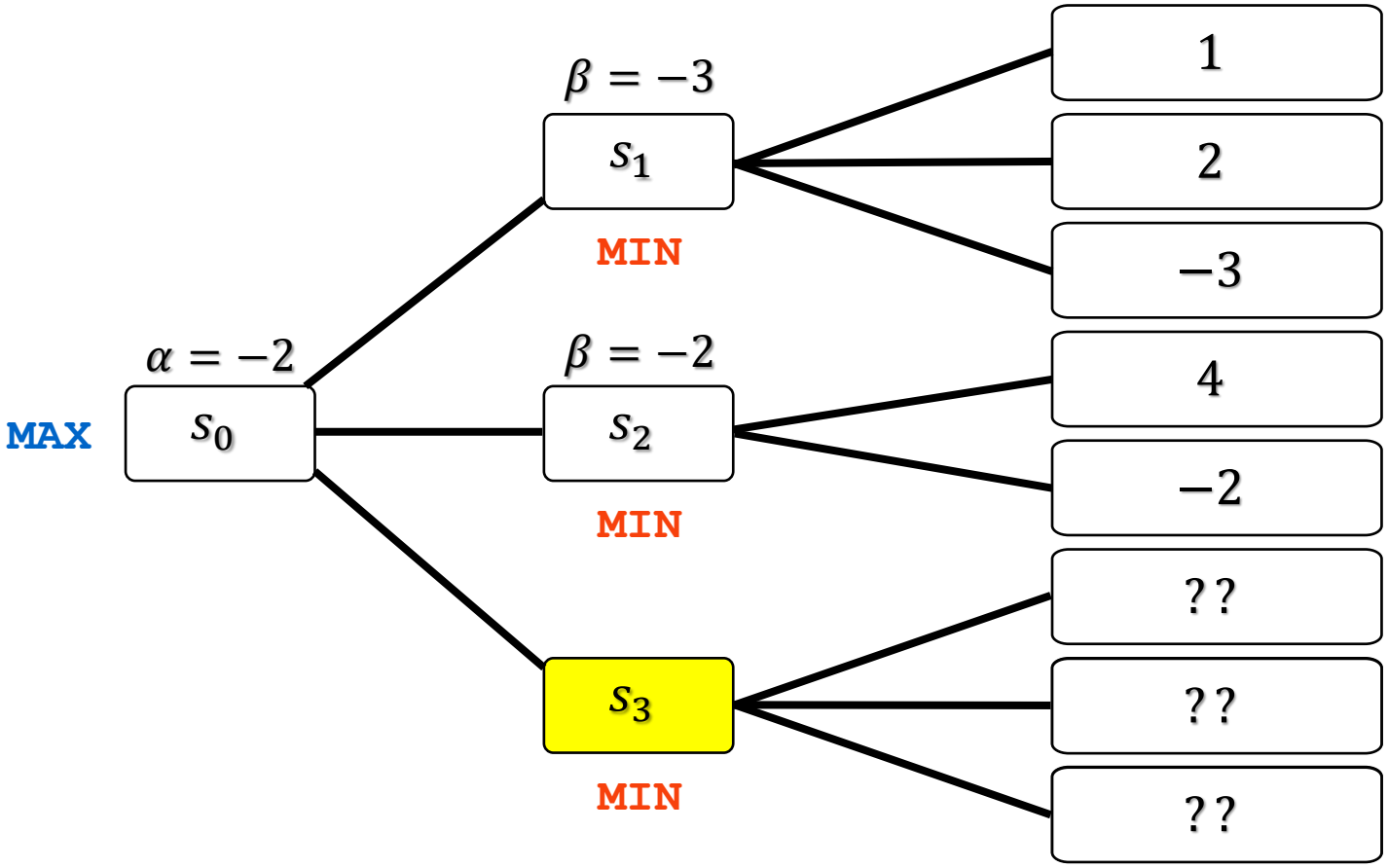
α-β Pruning – Example Trace



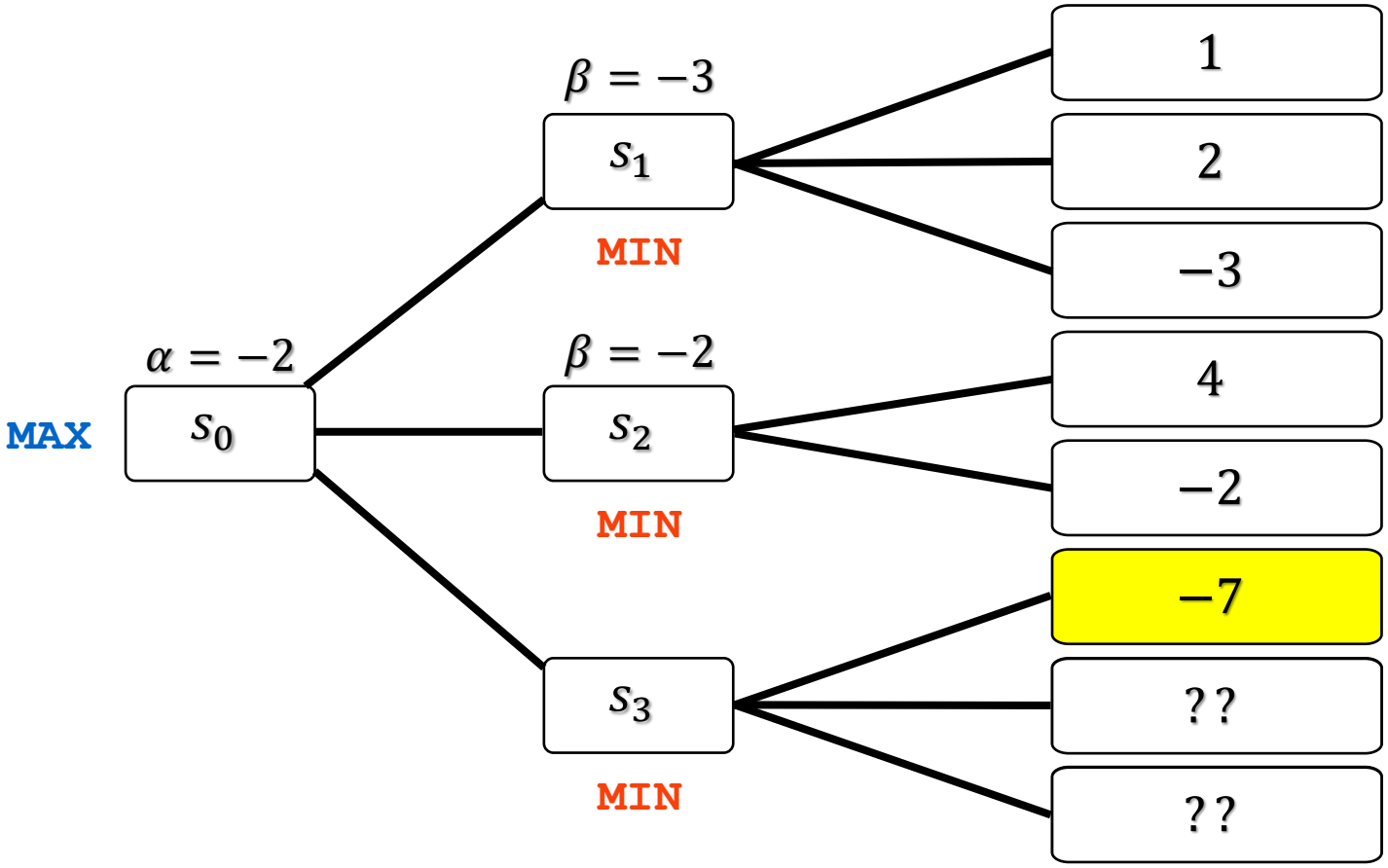
α-β Pruning – Example Trace



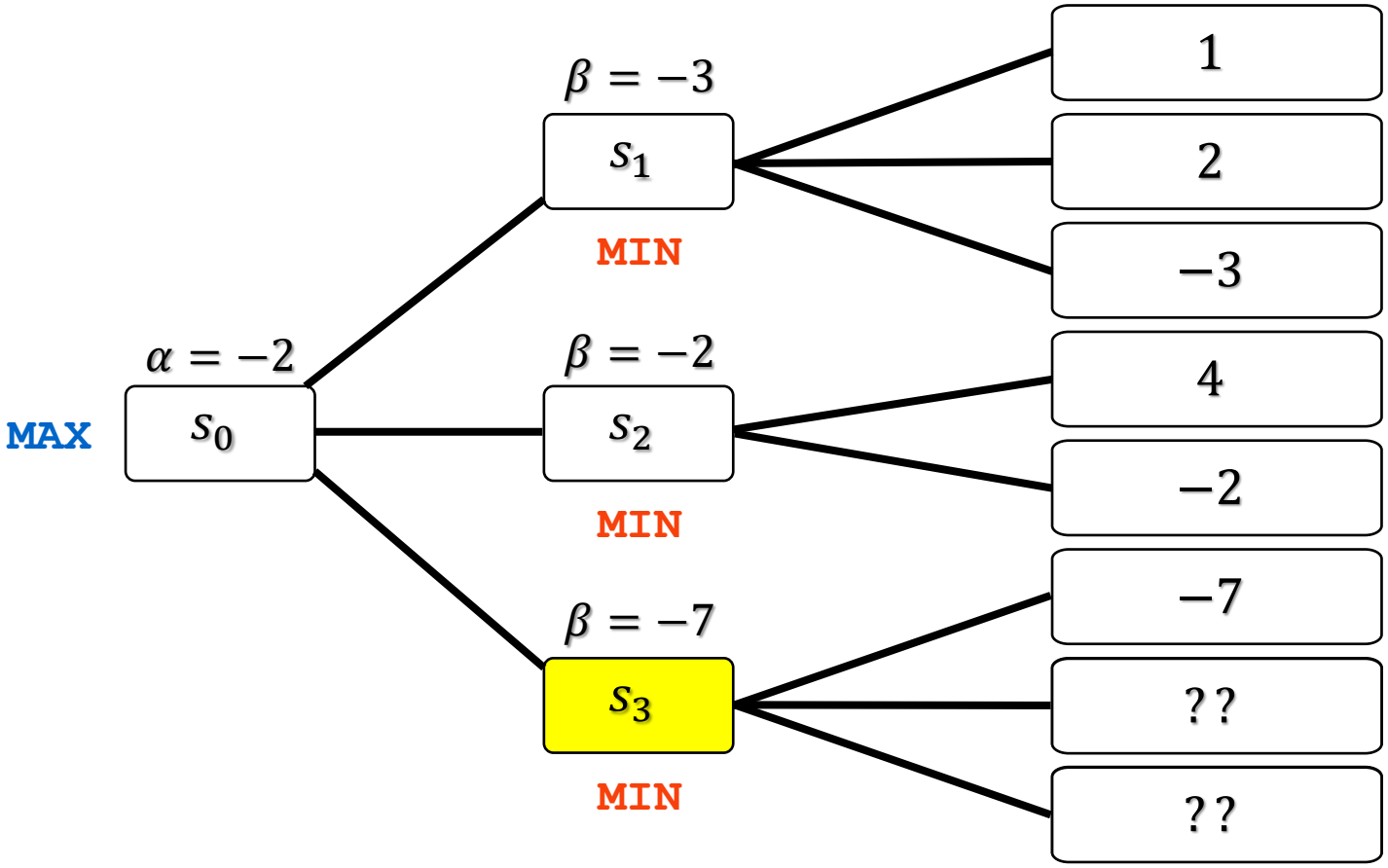
α-β Pruning – Example Trace



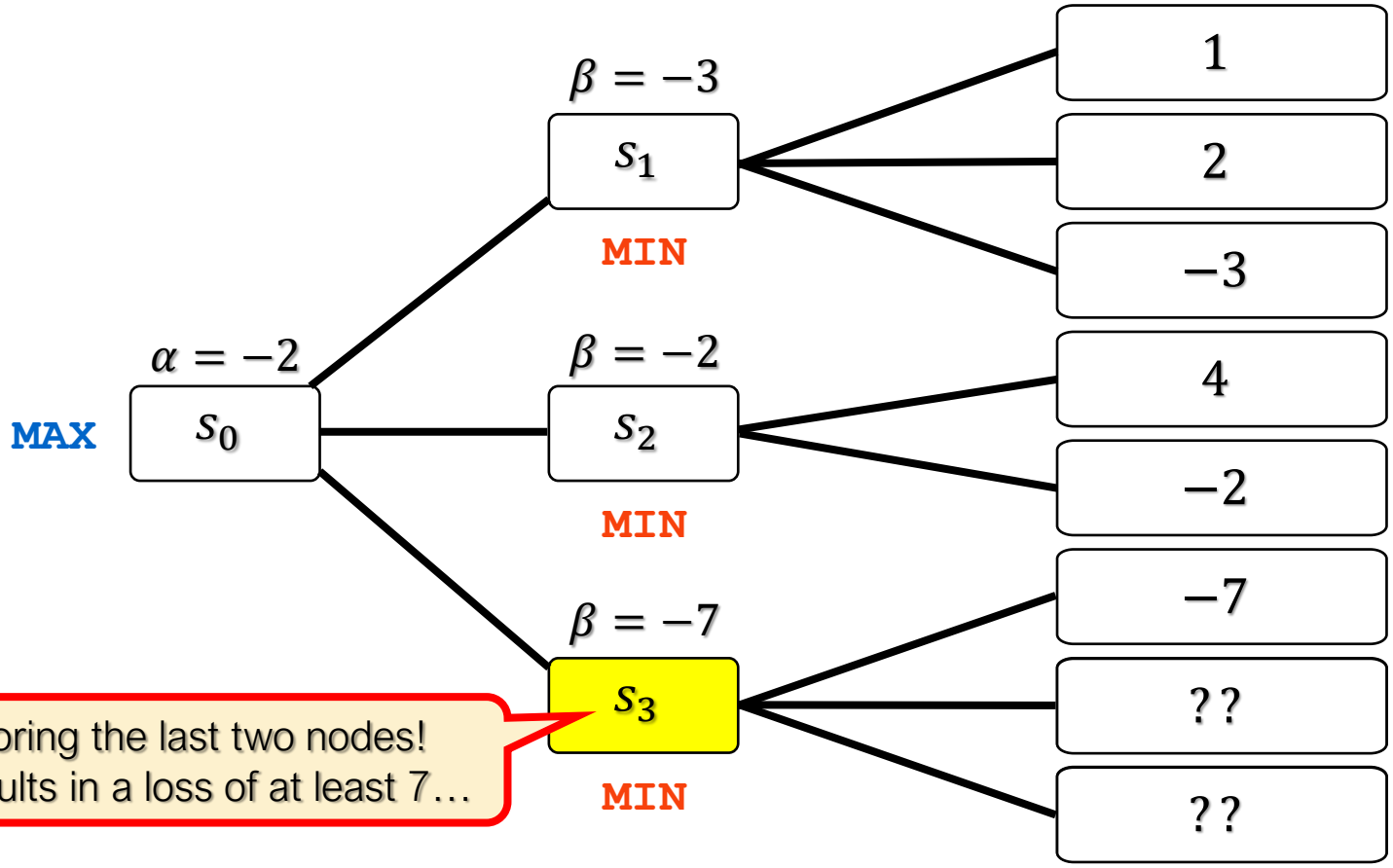
α-β Pruning – Example Trace



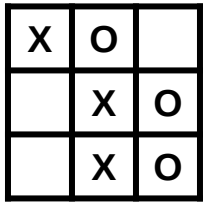
α-β Pruning – Example Trace



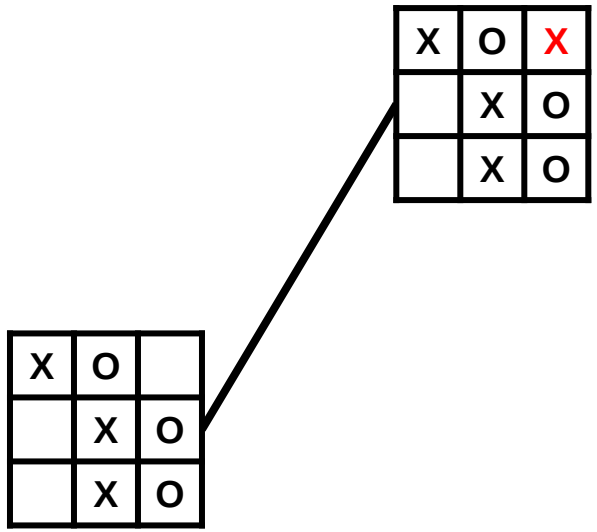
α-β Pruning – Example Trace



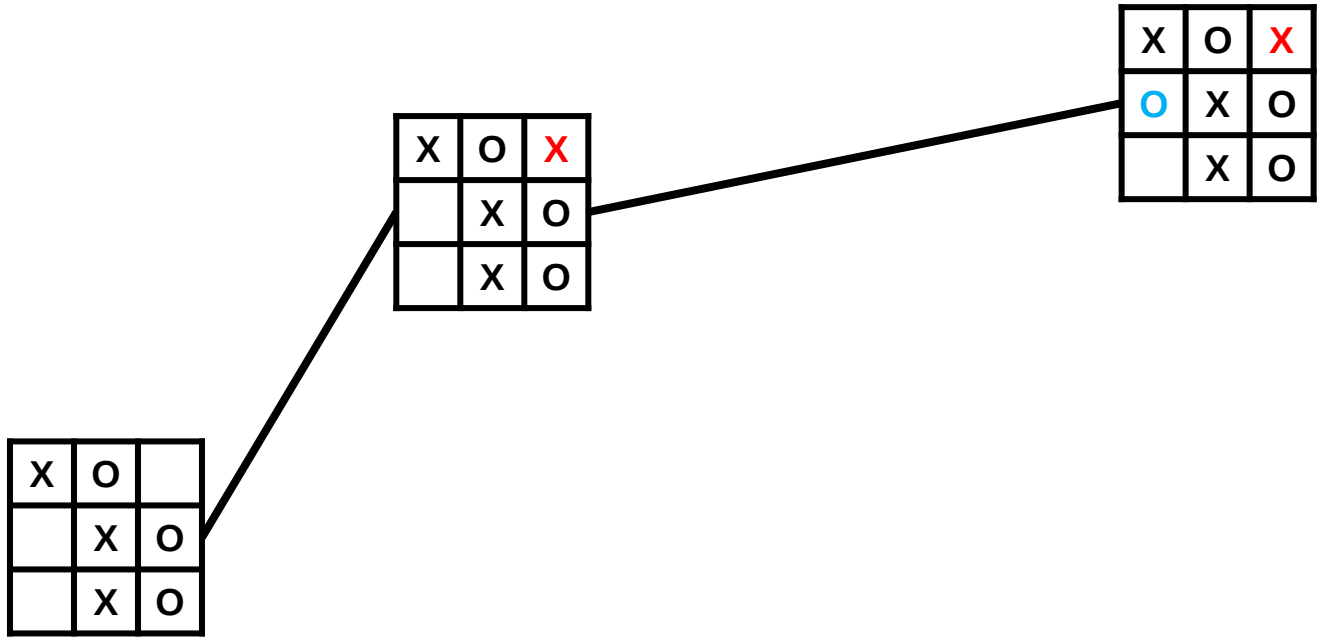
α-β Pruning – Tic-Tac-Toe Example Trace



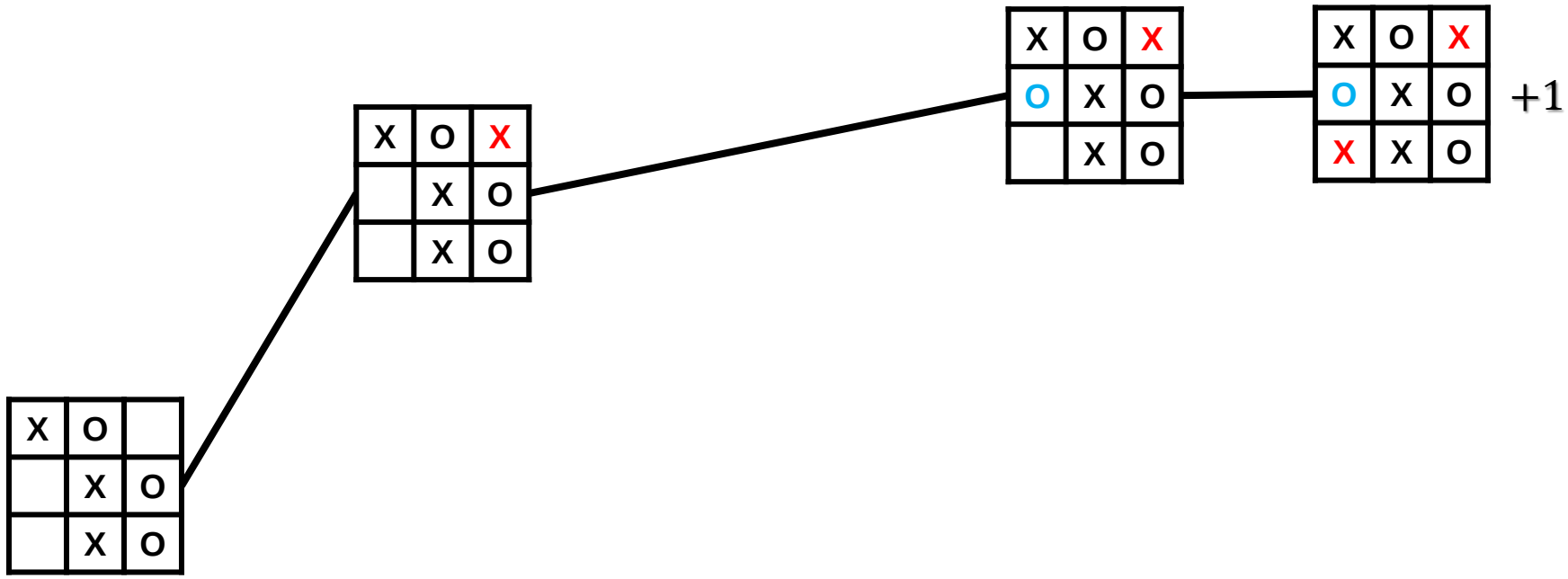
α-β Pruning – Tic-Tac-Toe Example Trace



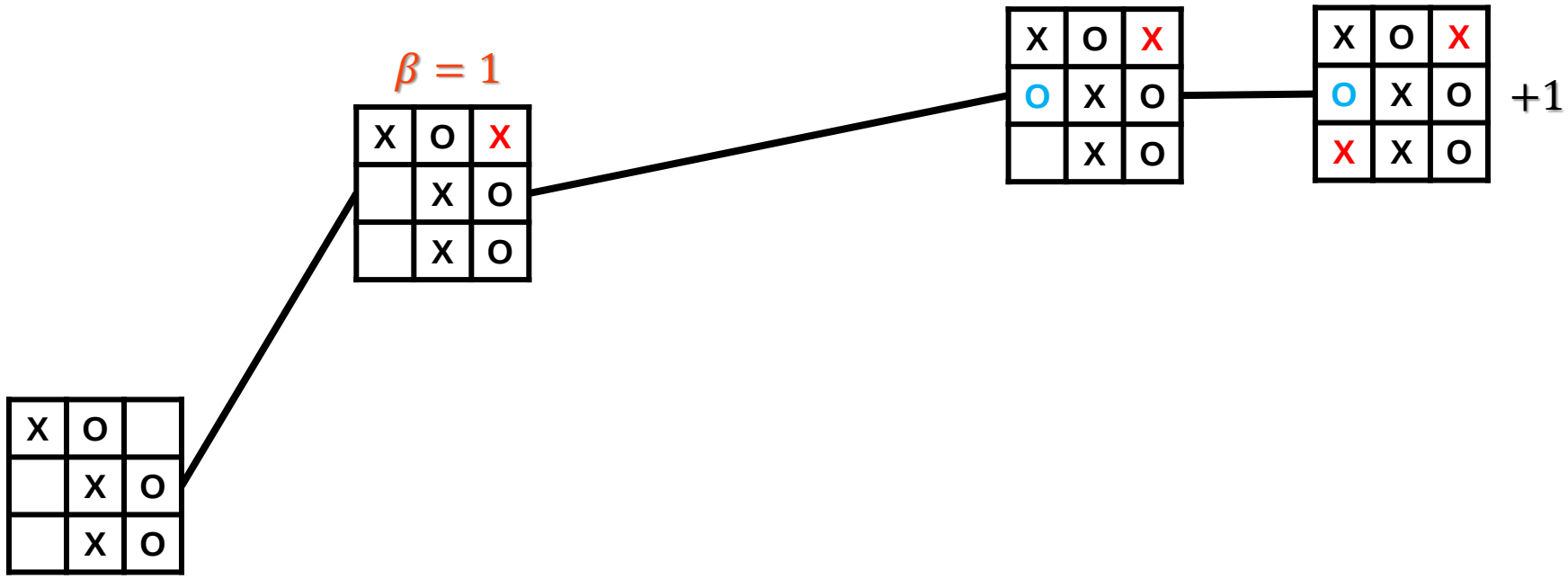
α-β Pruning – Tic-Tac-Toe Example Trace



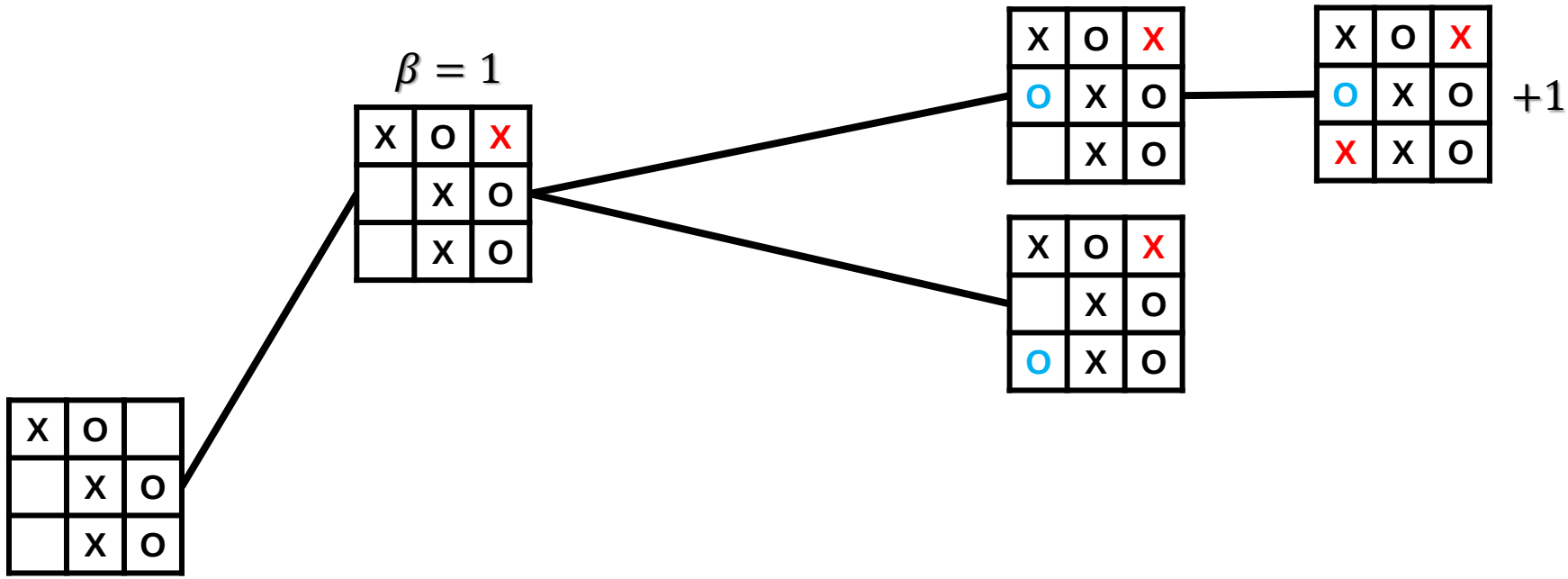
α-β Pruning – Tic-Tac-Toe Example Trace



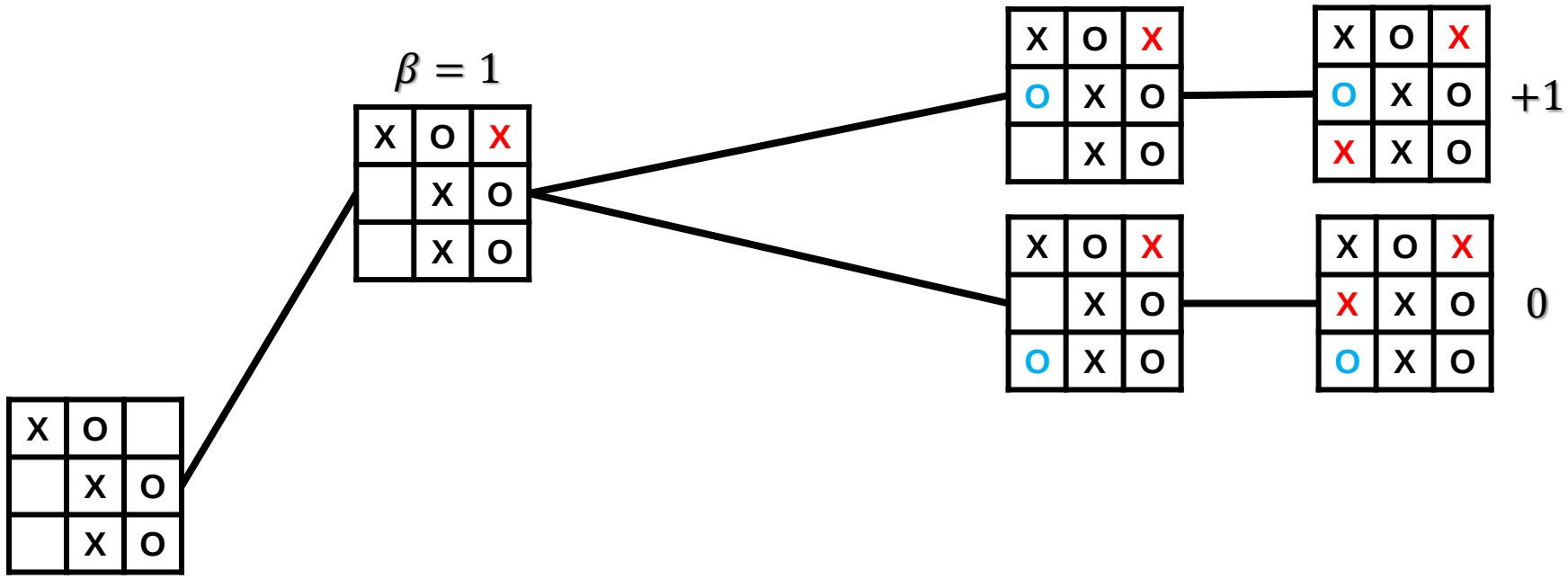
α-β Pruning – Tic-Tac-Toe Example Trace



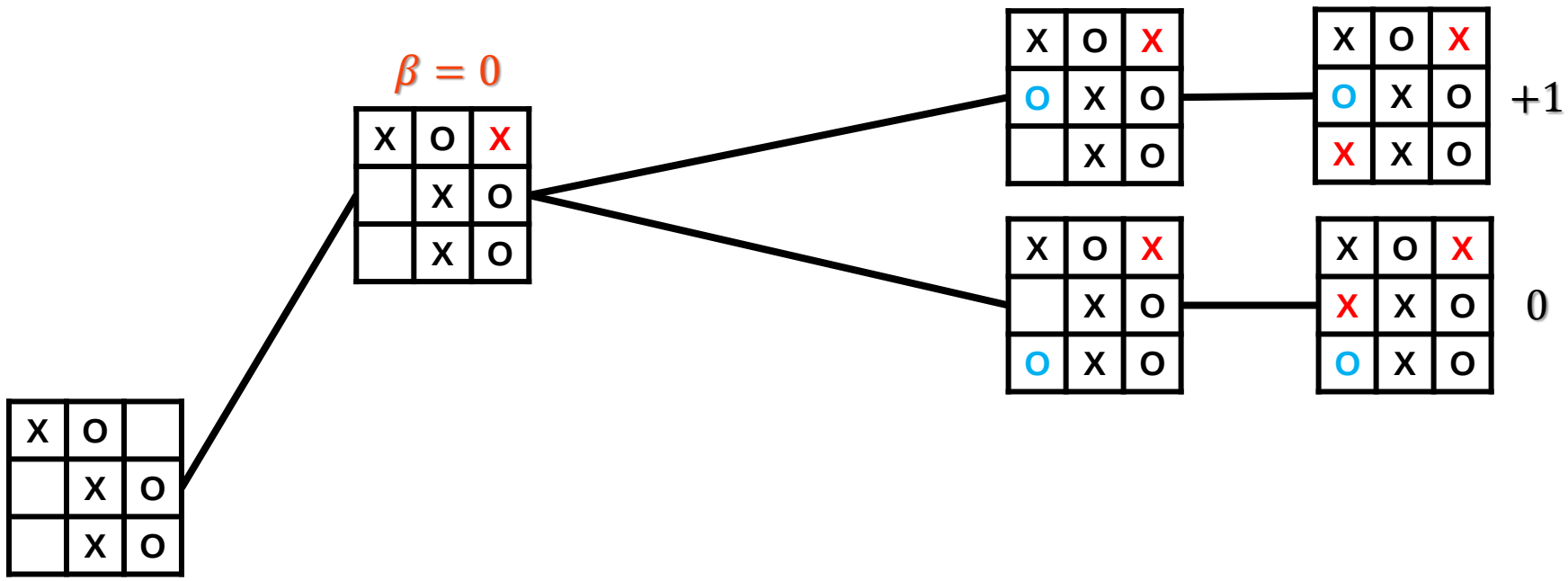
α-β Pruning – Tic-Tac-Toe Example Trace



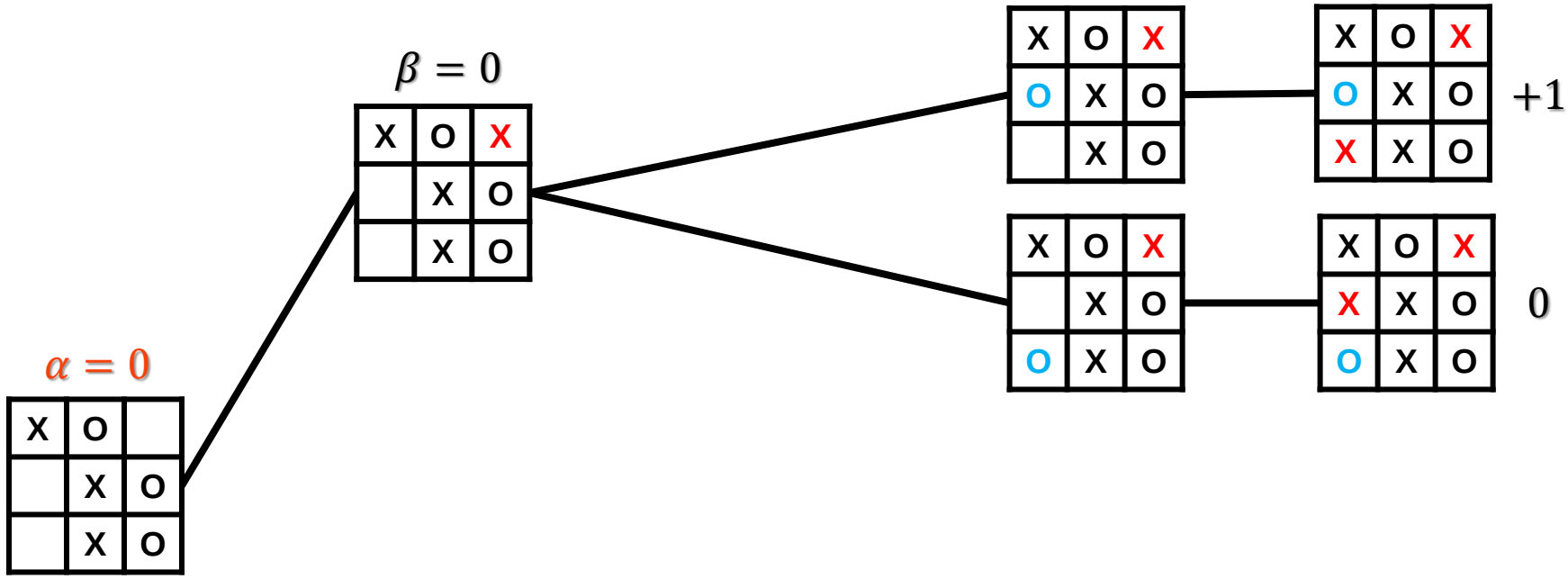
α-β Pruning – Tic-Tac-Toe Example Trace



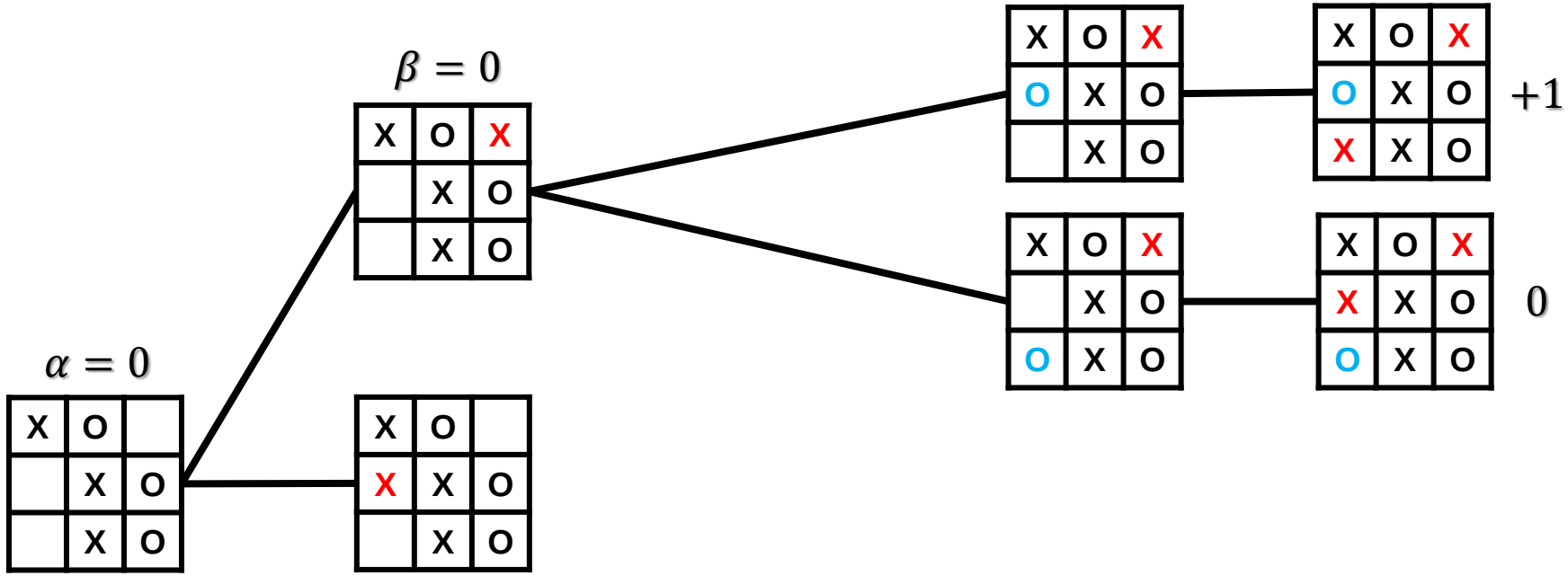
α-β Pruning – Tic-Tac-Toe Example Trace



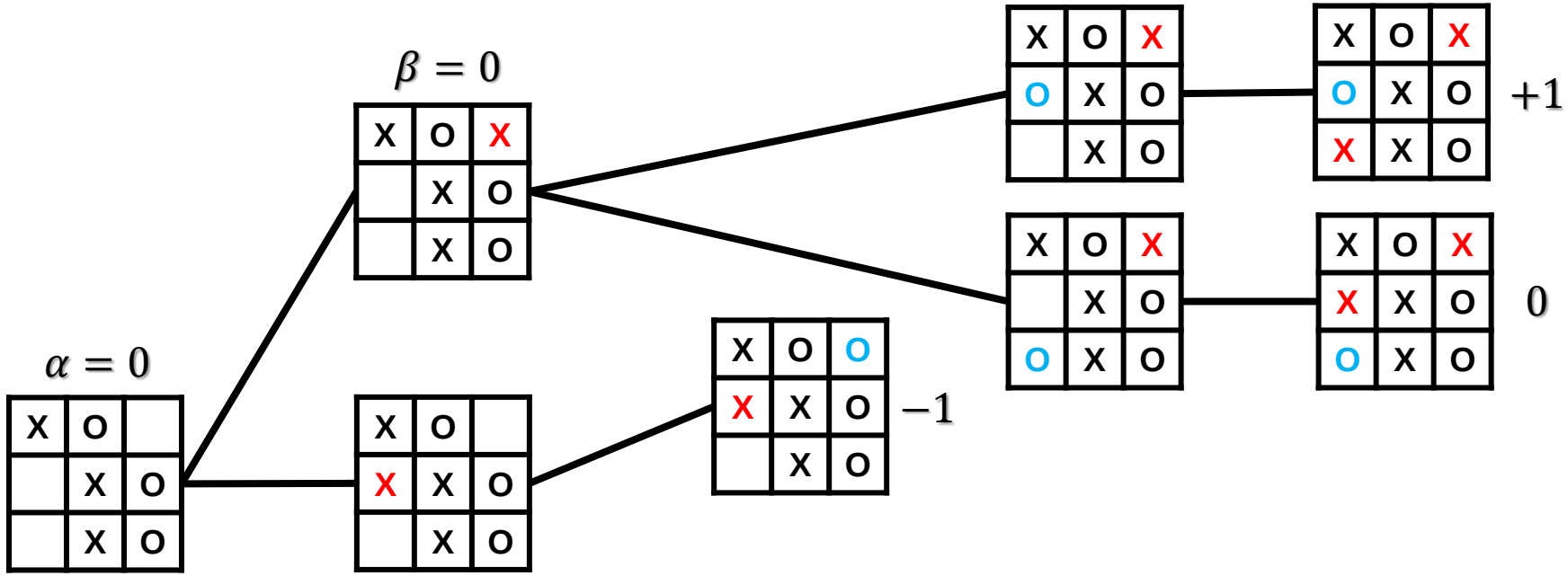
α-β Pruning – Tic-Tac-Toe Example Trace



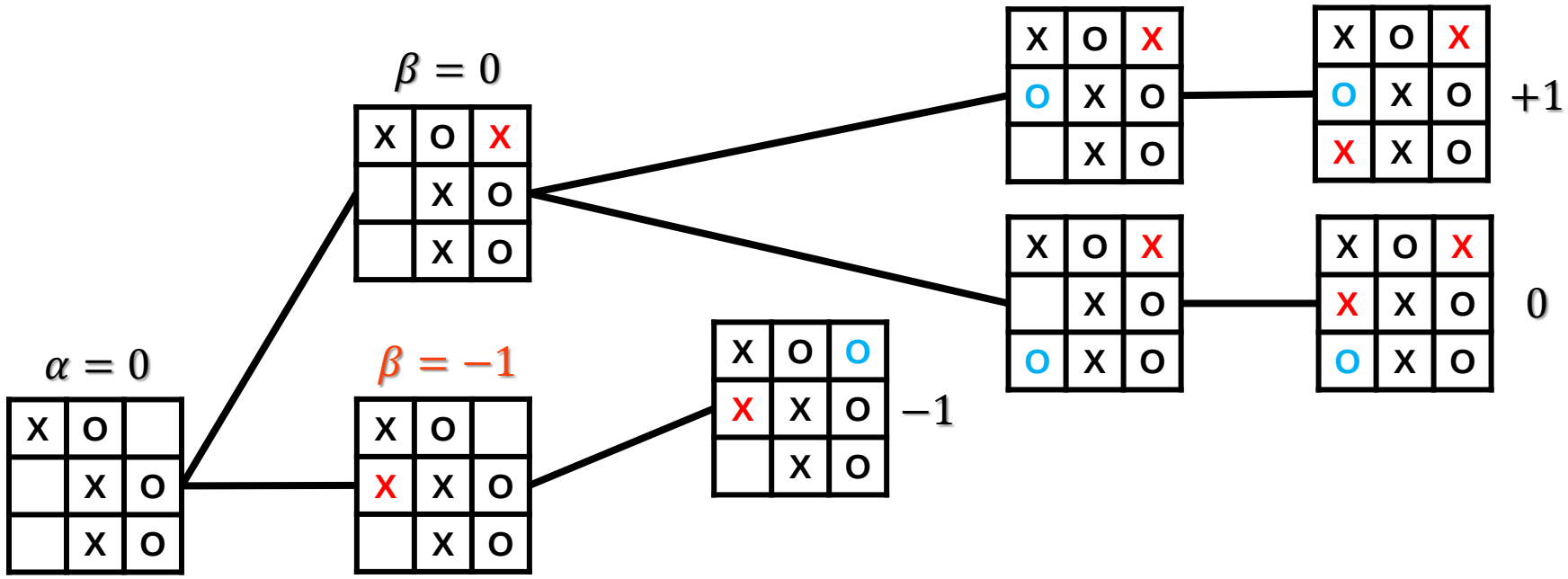
α-β Pruning – Tic-Tac-Toe Example Trace



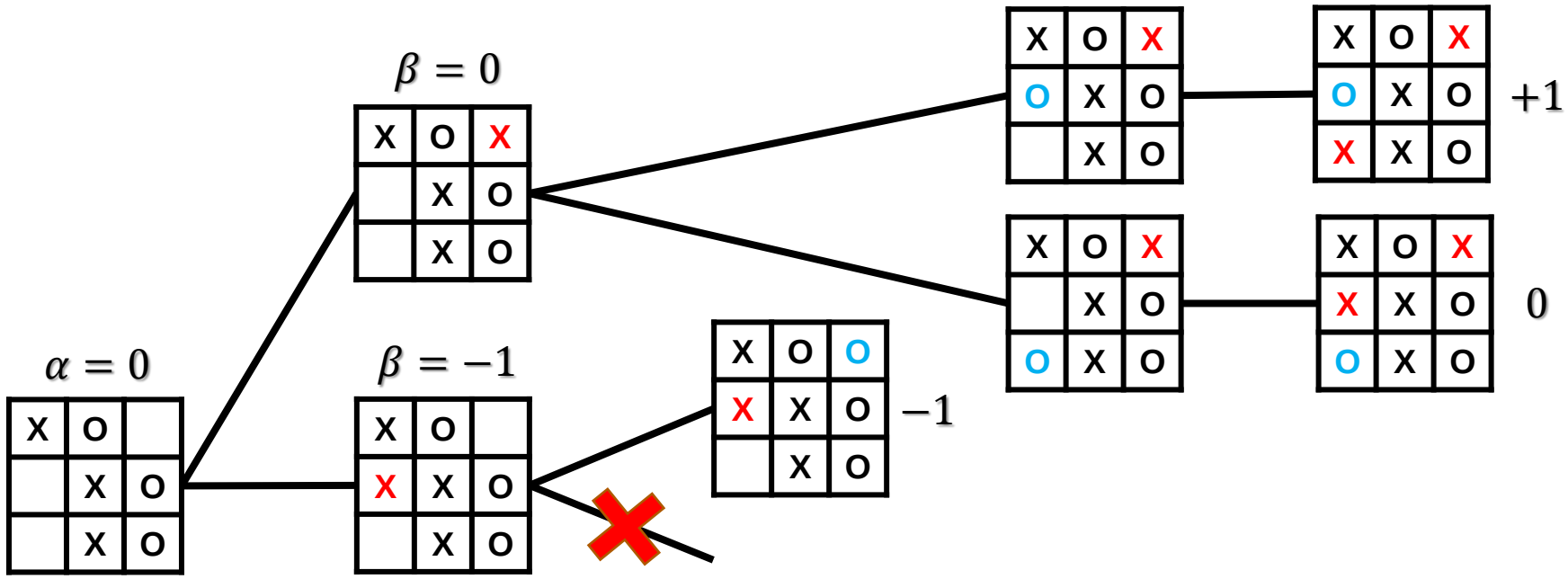
α-β Pruning – Tic-Tac-Toe Example Trace



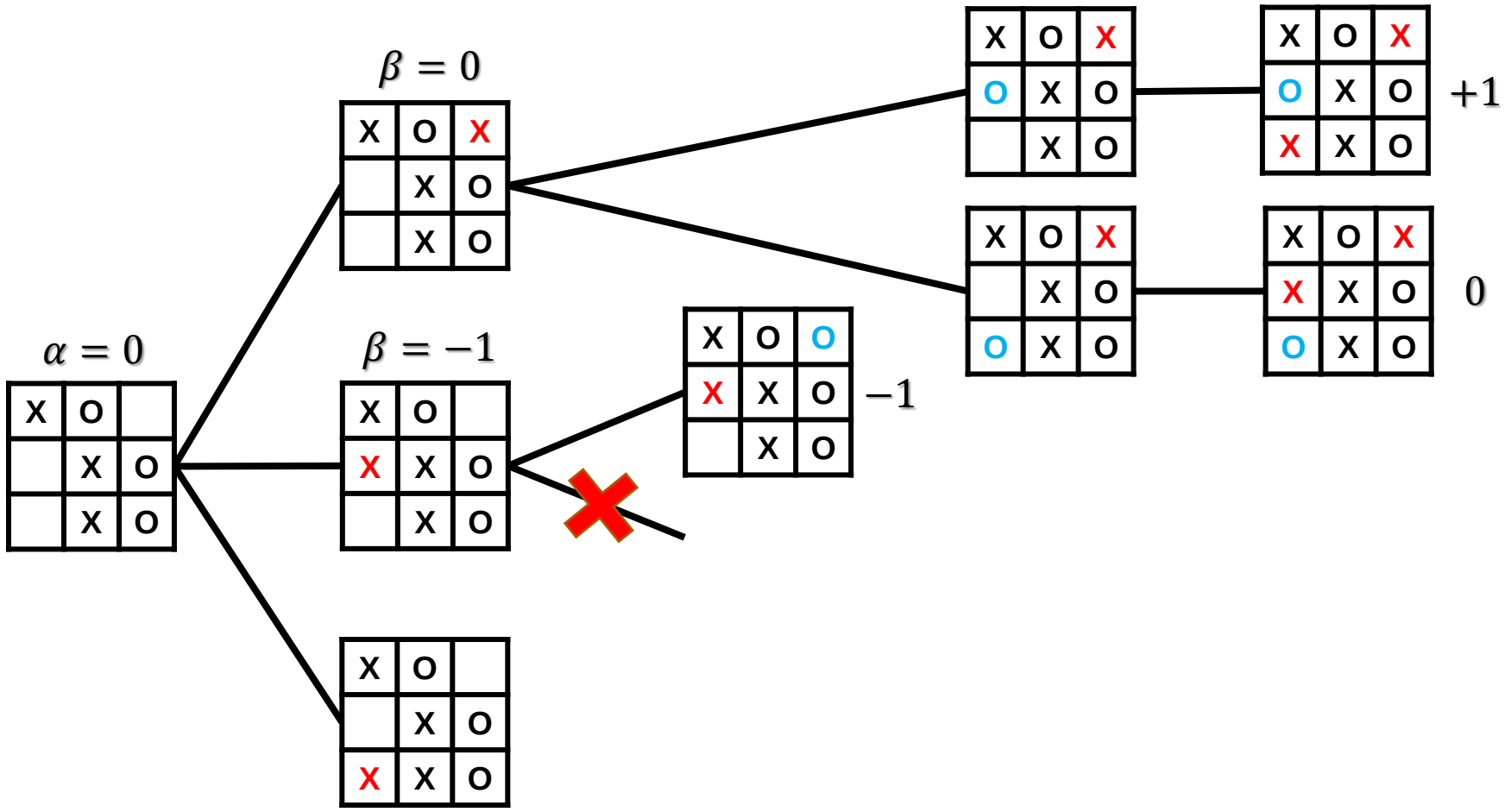
α-β Pruning – Tic-Tac-Toe Example Trace



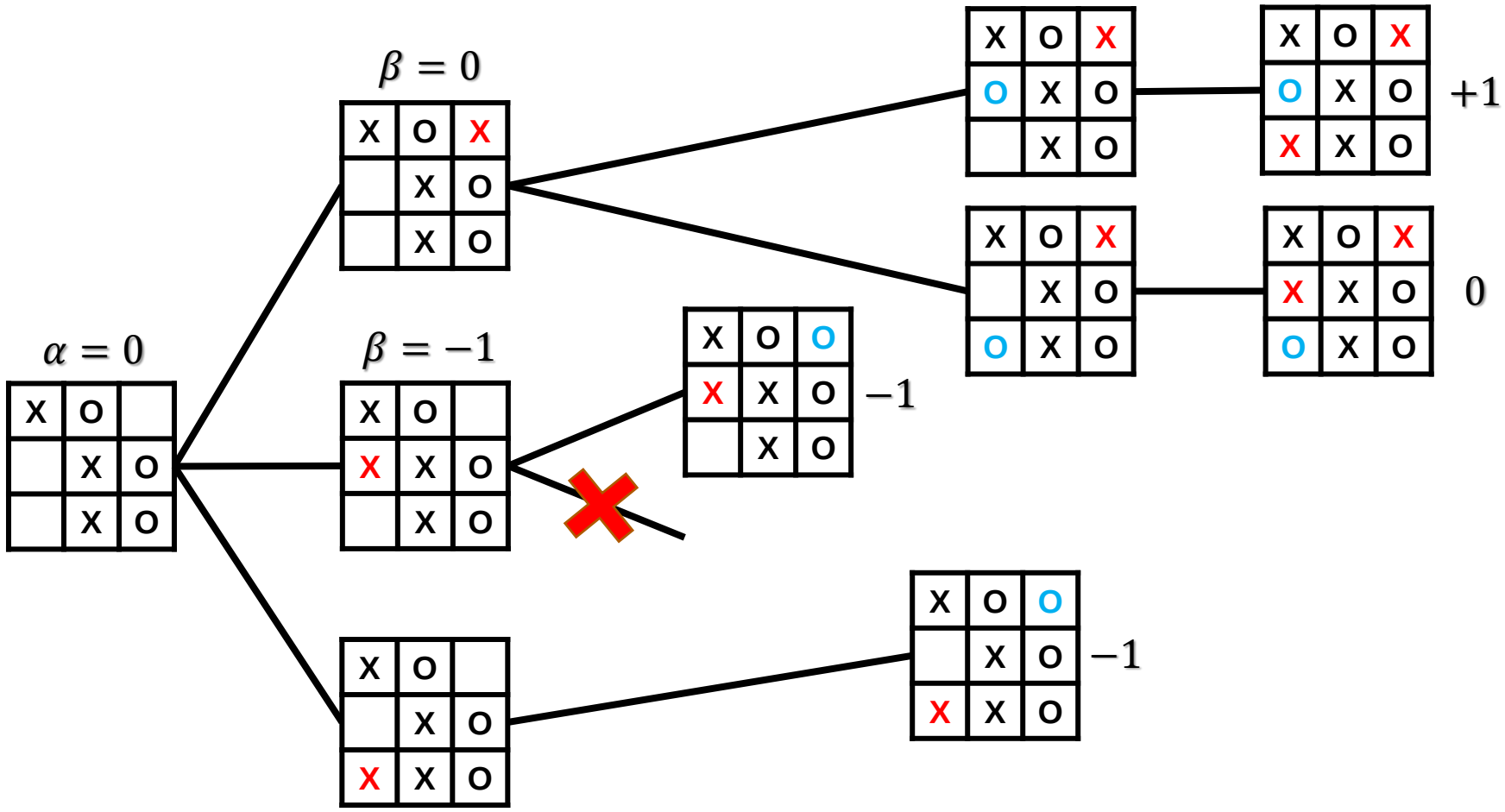
α-β Pruning – Tic-Tac-Toe Example Trace



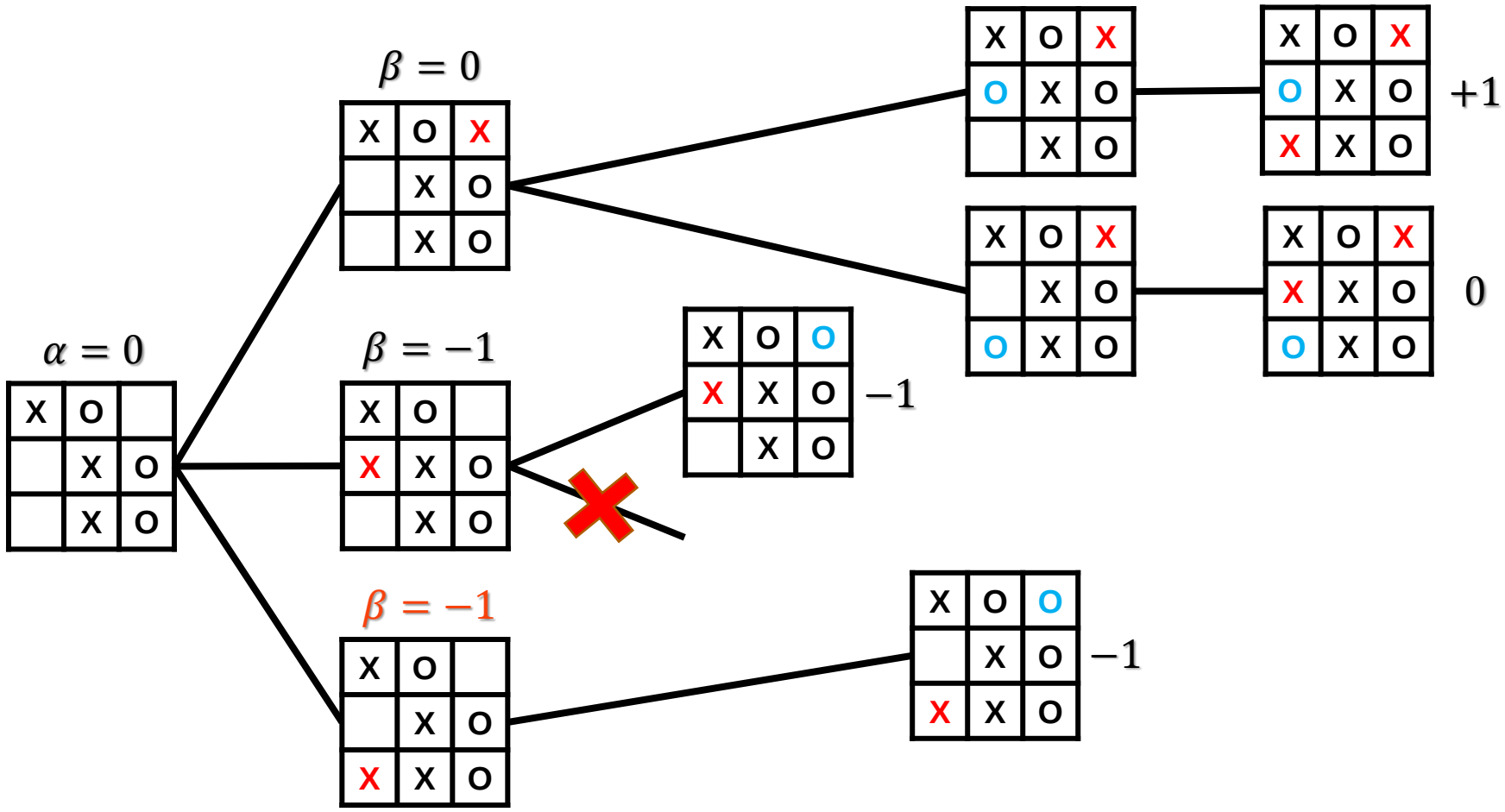
α-β Pruning – Tic-Tac-Toe Example Trace



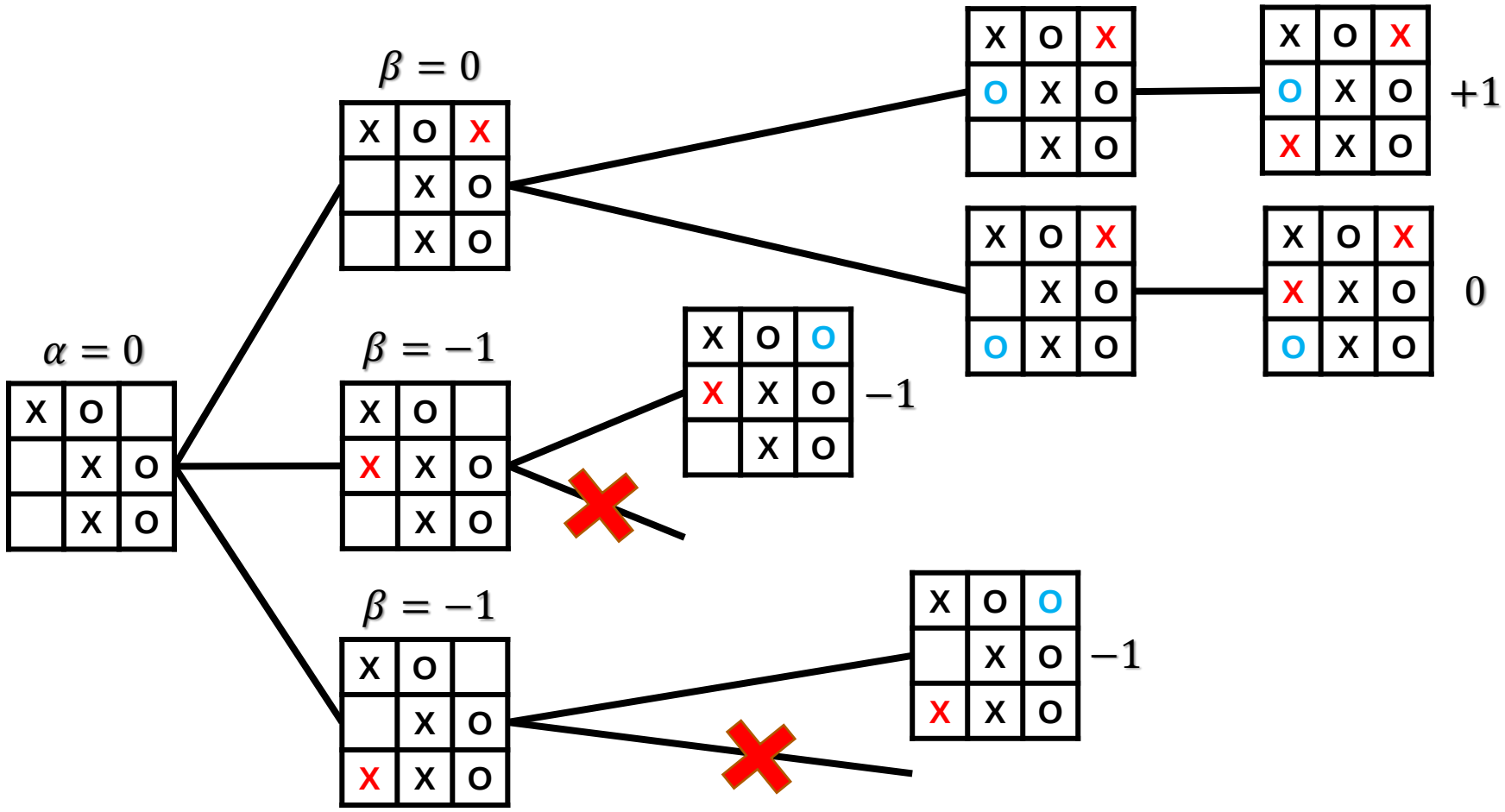
α-β Pruning – Tic-Tac-Toe Example Trace



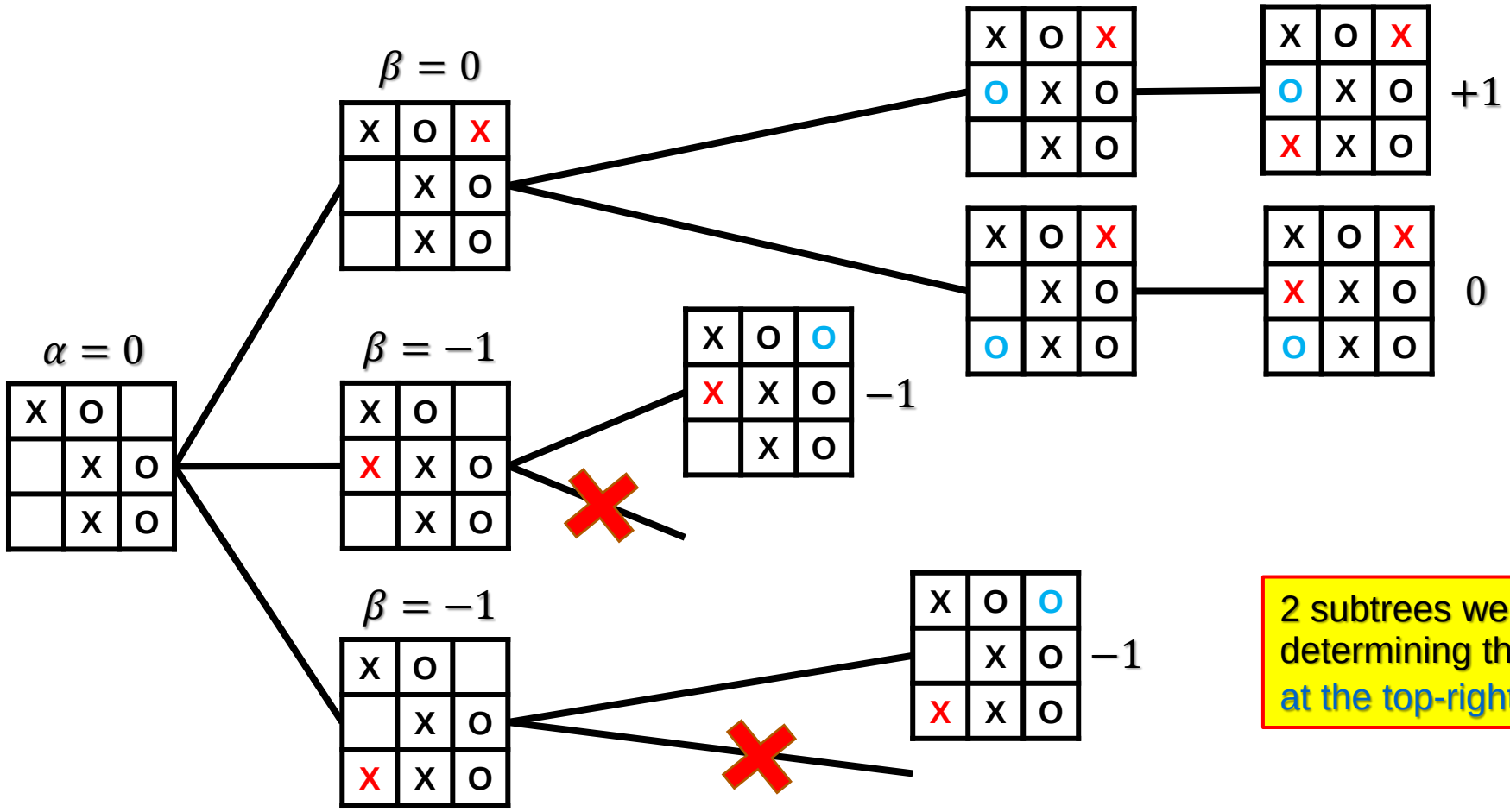
α-β Pruning – Tic-Tac-Toe Example Trace



α-β Pruning – Tic-Tac-Toe Example Trace



α-β Pruning – Tic-Tac-Toe Example Trace



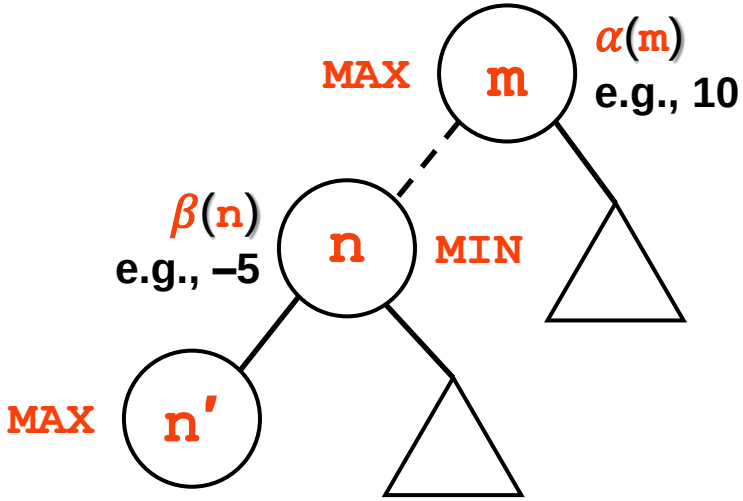
2 subtrees were pruned away in determining that MAX must play at the top-right to avoid losing

α-β Pruning

- **MAX** node **n**
  - α(**n**) = highest observed value found on path from **n**
  - Initially α(**n**) = $-\infty$
- **MIN** node **n**
  - β(**n**) = lowest observed value found on path from **n**
  - Initially β(**n**) = $+\infty$
- Pruning rules
  - Given a **MIN** node **n**, stop searching below **n** if
    - There exists some **MAX** ancestor **m** (of **n**) where α(**m**) ≥ β(**n**)
  - Given a **MAX** node **m**, stop searching below **m** if
    - There exists some **MIN** ancestor **n** (of **m**) with β(**n**) ≤ α(**m**)

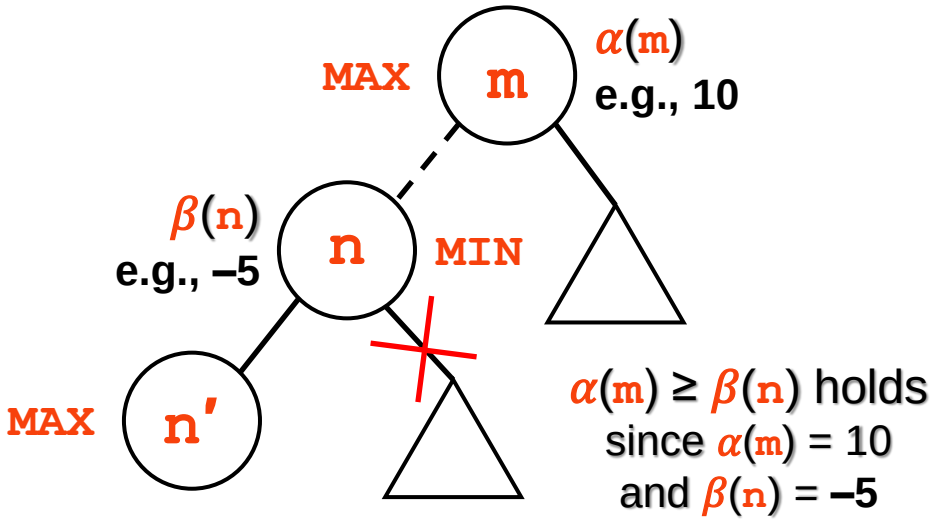
α-β Pruning

- **MAX** node **n**
  - α(**n**) = highest observed value found on path from **n**
  - Initially α(**n**) = $-\infty$
- **MIN** node **n**
  - β(**n**) = lowest observed value found on path from **n**
  - Initially β(**n**) = $+\infty$
- Pruning rules
  - Given a **MIN** node **n**, stop searching below **n** if
    - There exists some **MAX** ancestor **m** (of **n**) where α(**m**) ≥ β(**n**)
  - Given a **MAX** node **m**, stop searching below **m** if
    - There exists some **MIN** ancestor **n** (of **m**) with β(**n**) ≤ α(**m**)



α-β Pruning

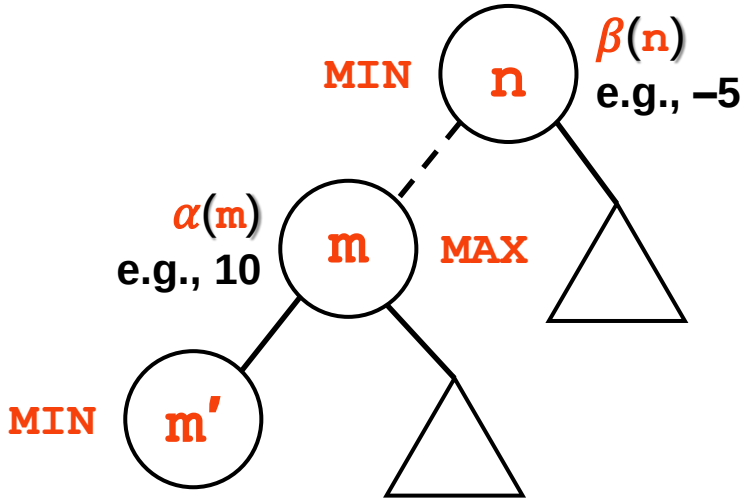
- **MAX** node **n**
  - α(**n**) = highest observed value found on path from **n**
  - Initially α(**n**) = $-\infty$
- **MIN** node **n**
  - β(**n**) = lowest observed value found on path from **n**
  - Initially β(**n**) = $+\infty$
- Pruning rules
  - Given a **MIN** node **n**, stop searching below **n** if
    - There exists some **MAX** ancestor **m** (of **n**) where α(**m**) ≥ β(**n**)
  - Given a **MAX** node **m**, stop searching below **m** if
    - There exists some **MIN** ancestor **n** (of **m**) with β(**n**) ≤ α(**m**)



MIN will choose β(**n**) or lower at **n**, but ancestor **MAX** will NEVER choose the subtree at **n** since at **m** there is a better option with higher value α(**m**)

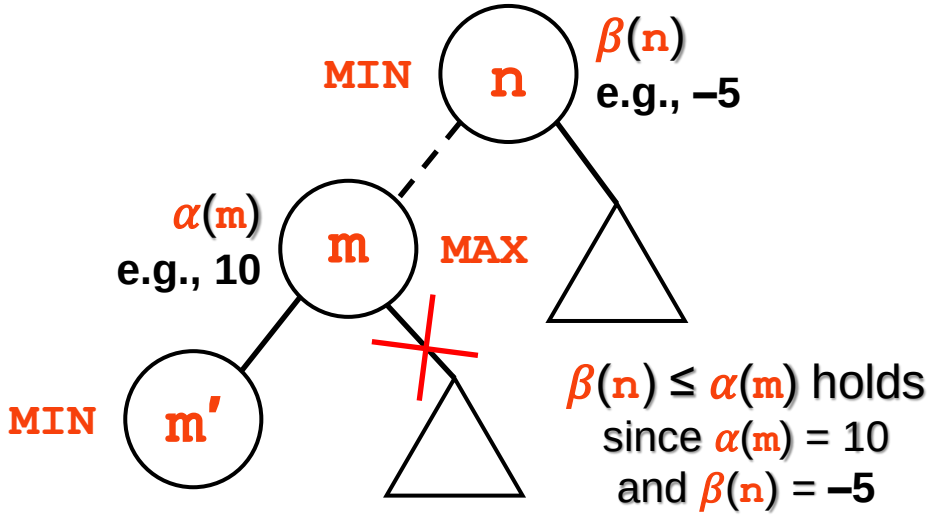
α-β Pruning

- **MAX** node **n**
  - α(**n**) = highest observed value found on path from **n**
  - Initially α(**n**) = $-\infty$
- **MIN** node **n**
  - β(**n**) = lowest observed value found on path from **n**
  - Initially β(**n**) = $+\infty$
- Pruning rules
  - Given a **MIN** node **n**, stop searching below **n** if
    - There exists some **MAX** ancestor **m** (of **n**) where α(**m**) ≥ β(**n**)
  - Given a **MAX** node **m**, stop searching below **m** if
    - There exists some **MIN** ancestor **n** (of **m**) with β(**n**) ≤ α(**m**)



α-β Pruning

- **MAX** node **n**
  - α(**n**) = highest observed value found on path from **n**
  - Initially α(**n**) = $-\infty$
- **MIN** node **n**
  - β(**n**) = lowest observed value found on path from **n**
  - Initially β(**n**) = $+\infty$
- Pruning rules
  - Given a **MIN** node **n**, stop searching below **n** if
    - There exists some **MAX** ancestor **m** (of **n**) where α(**m**) ≥ β(**n**)
  - Given a **MAX** node **m**, stop searching below **m** if
    - There exists some **MIN** ancestor **n** (of **m**) with β(**n**) ≤ α(**m**)



MAX will choose α(**m**) or higher at **m**, but ancestor **MIN** will NEVER choose the subtree at **m** since at **n** there is a better option with lower value β(**n**)

α-β Pruning Analysis

- Pruning a branch never affects the final outcome
- Expansion-order heuristics will improve the search
- Good move ordering improves effectiveness of pruning
  - "Perfect" ordering
    - Time complexity*: $O(b^{m/2})$
    - Good pruning strategies allow us to search twice as deep!
  - Example: Chess
    - Simple ordering gets you close to best-case result
      - Checks
      - Taken pieces
      - Forward moves
      - Backwards moves
- Random ordering gives complexity* $O(b^{3m/4})$ for $b < 1000$

* Specifics on these complexities will not be examinable

Issue with α-β Pruning

- Original problem
  - Most games have very large game trees
  - Solution: use α-β pruning since it can prune large parts of search trees
- Unresolved issue
  - Maximum depth of tree
    - Backwards induction works backwards from terminal states
    - Must still traverse to terminal states
  - Standard solution – Heuristic Minimax
    - Cutoff test – e.g., depth limit (DLS)
    - Uses an evaluation function – estimates expected utility of state

5 Heuristic Minimax

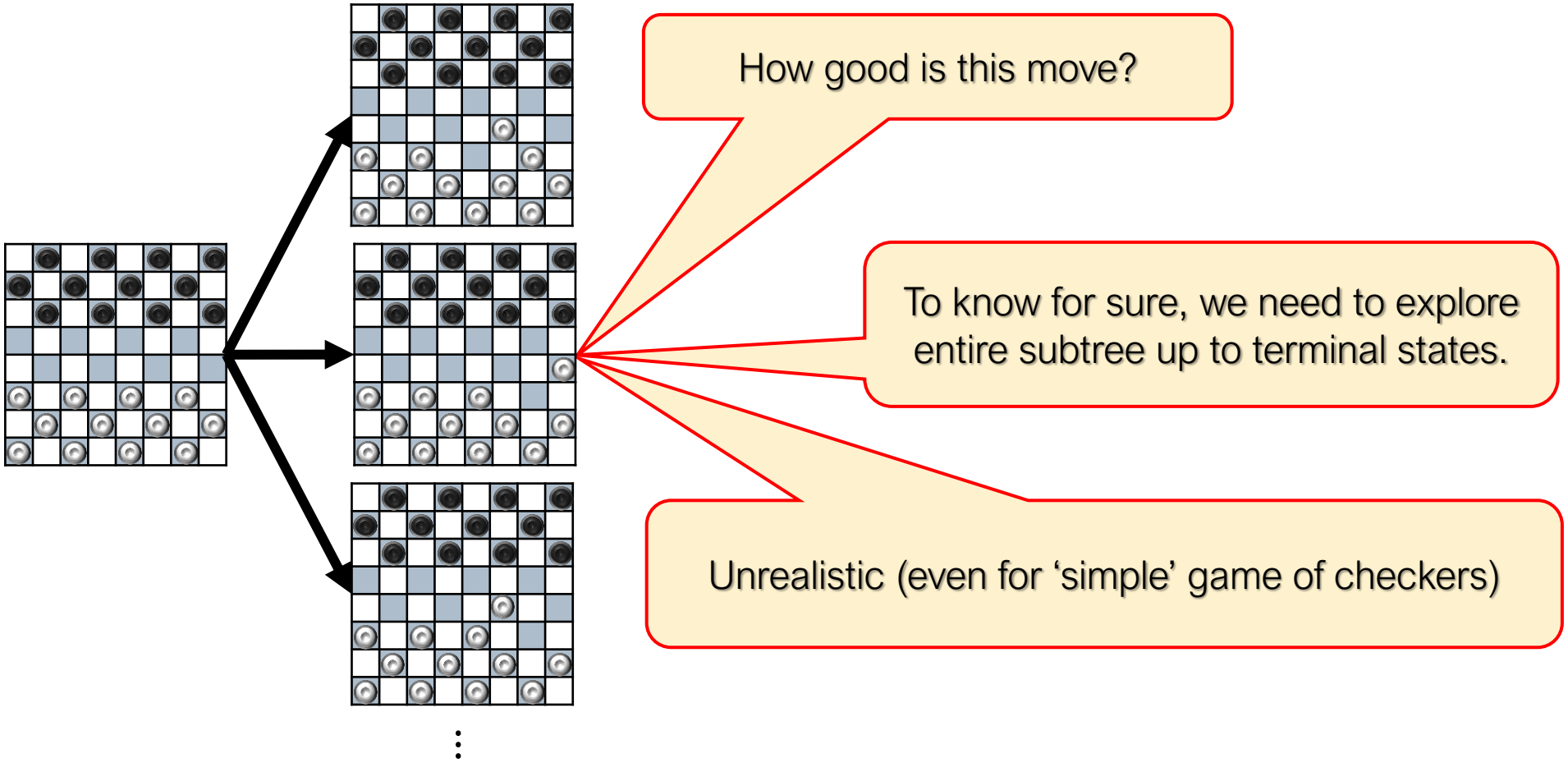
Heuristic Minimax Value

$$Minimax(s) = \begin{cases} \text{Utility}(s, \text{MAX}) & \text{if Is}-\text{Terminal}(s) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if To}-\text{Move}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if To}-\text{Move}(s) = \text{MIN} \end{cases}$$

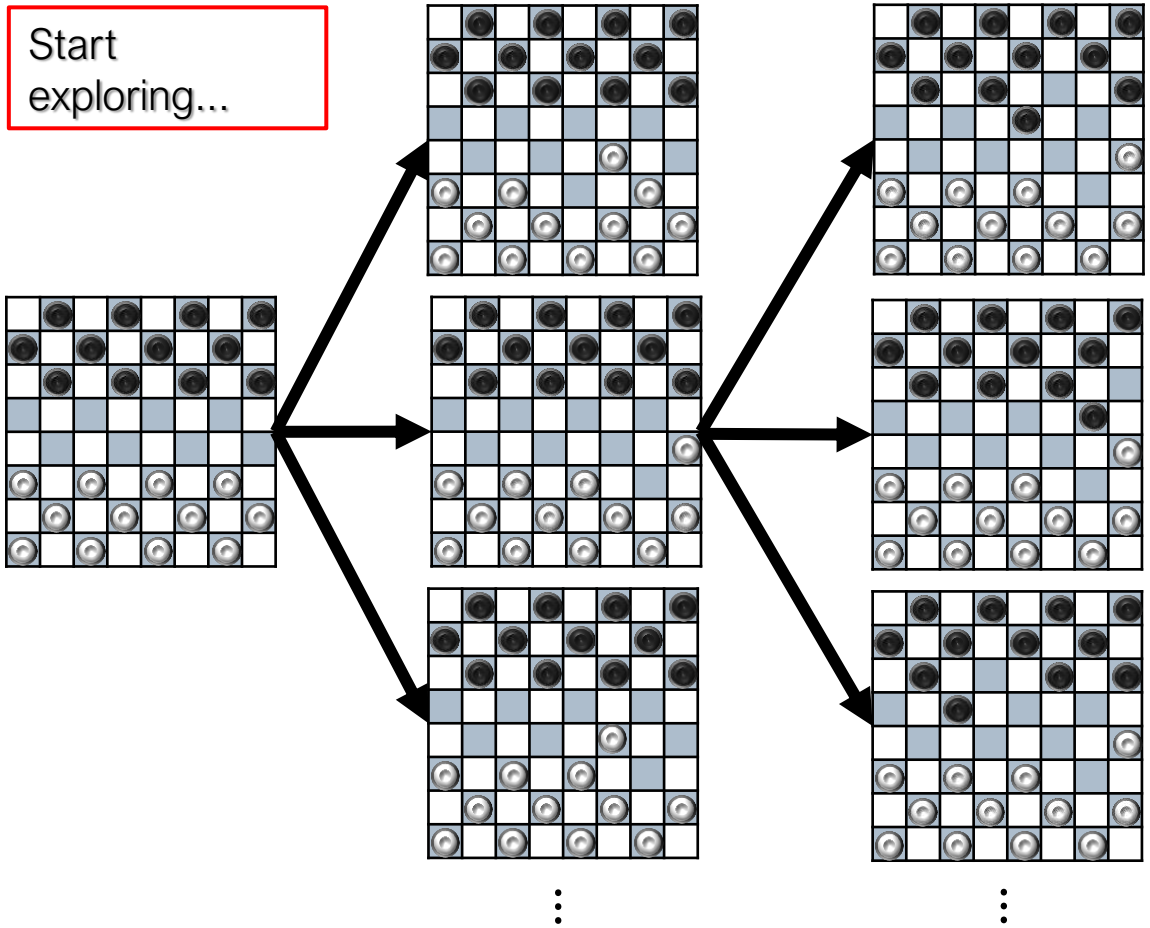
$$H\text{-}Minimax(s,d) = \begin{cases} Eval(s, \text{MAX}) & \text{if Cutoff}-\text{Test}(s,d) \\ \max_{a \in \text{Actions}(s)} \text{H}-\text{Minimax}(\text{Result}(s,a), d+1) & \text{if To}-\text{Move}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H}-\text{Minimax}(\text{Result}(s,a), d+1) & \text{if To}-\text{Move}(s) = \text{MIN} \end{cases}$$

Run `Minimax` until depth `d`; then start using the evaluation function to choose nodes

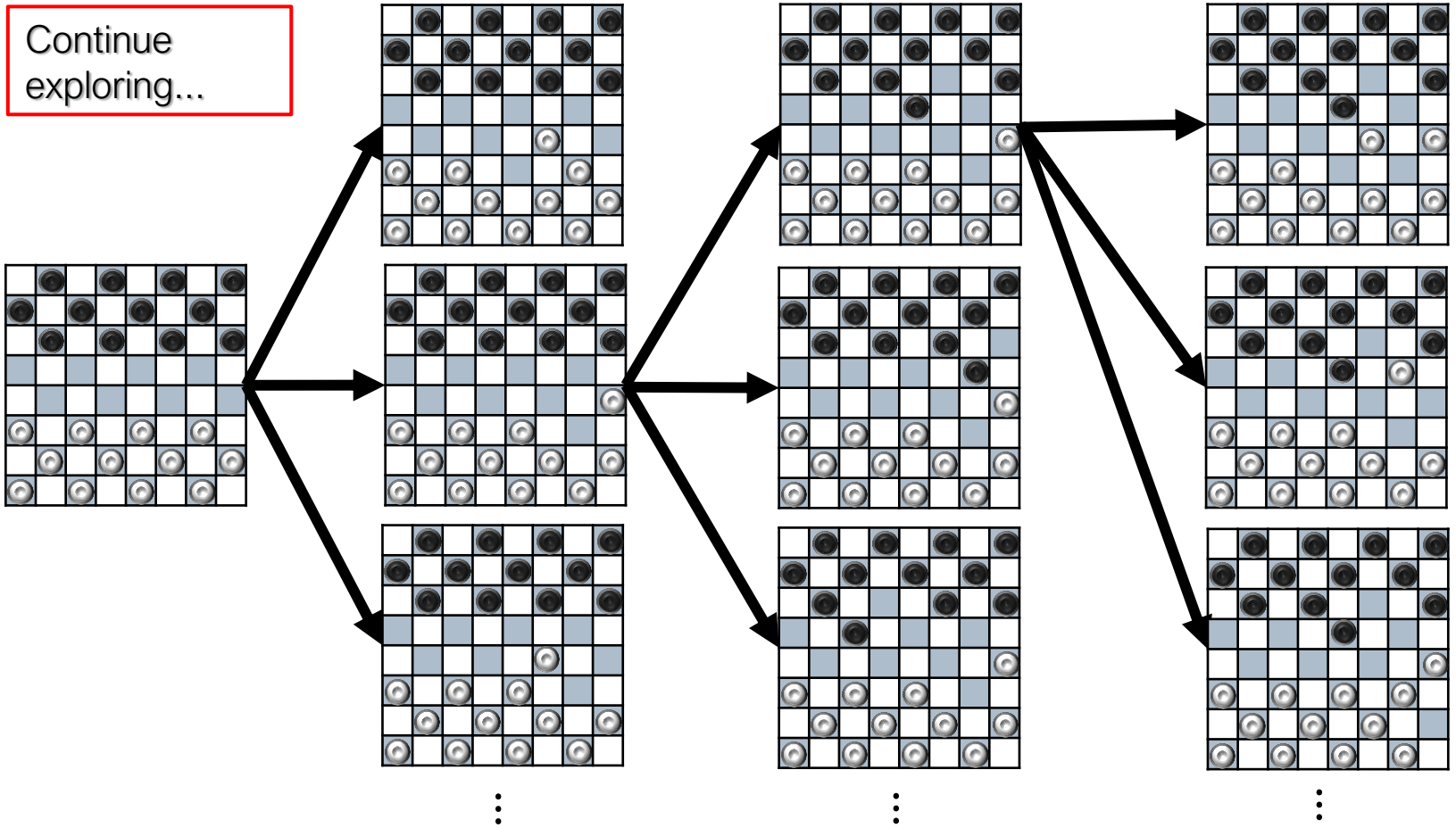
Evaluation Functions – Checkers Example



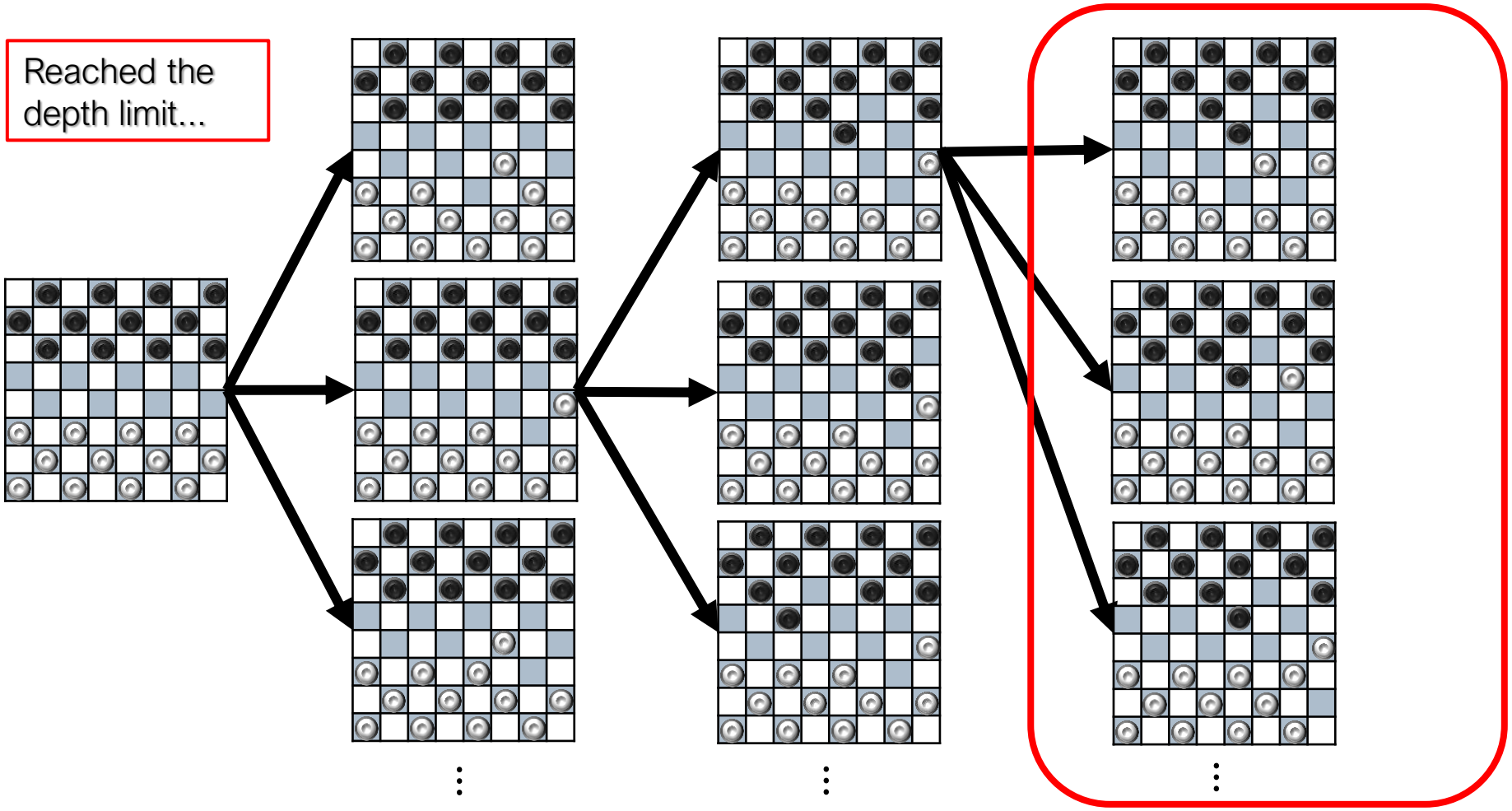
Evaluation Functions – Checkers Example



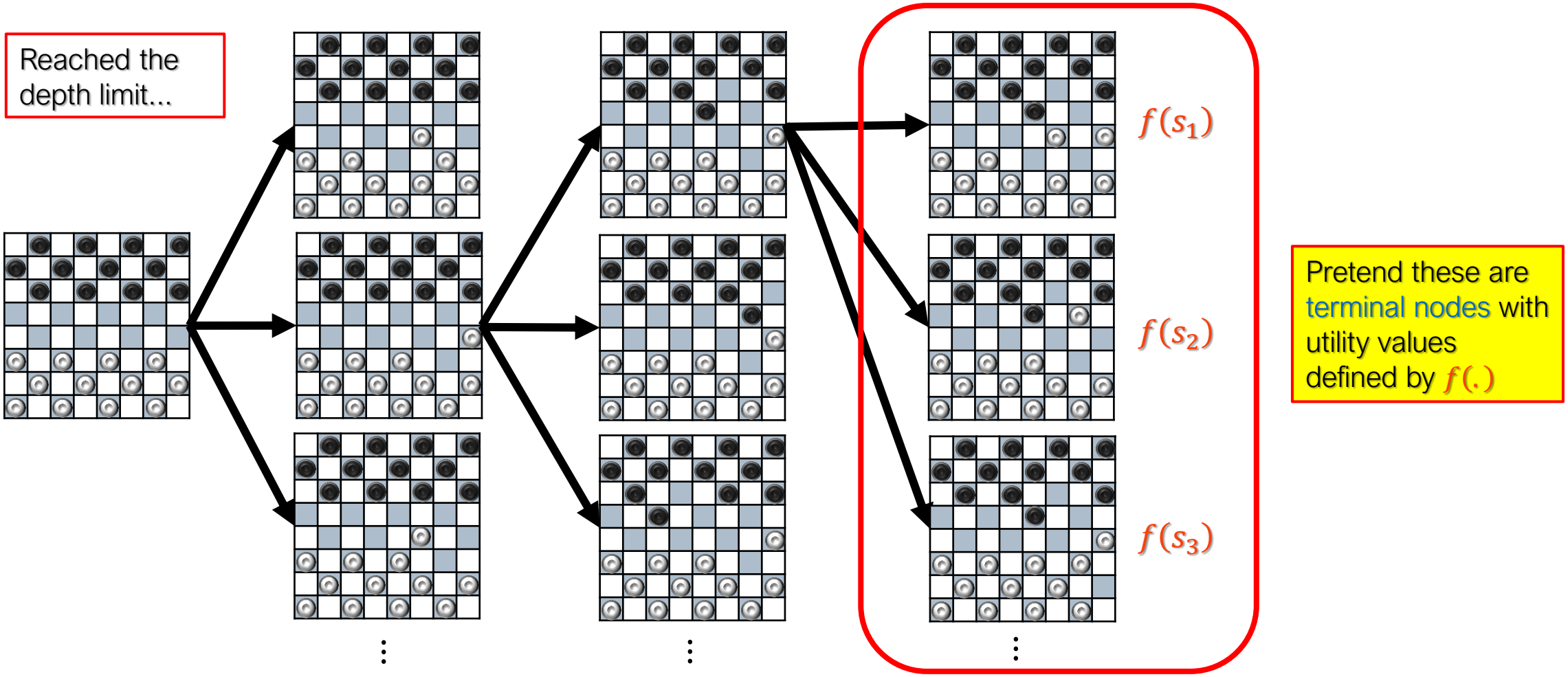
Evaluation Functions – Checkers Example



Evaluation Functions – Checkers Example



Evaluation Functions – Checkers Example



Notes on H-Minimax

- Modifies Minimax or α-β Pruning algorithms
  - Replaces `Is-Terminal(s)` with `Cutoff-Test(s,d)`
    - How deep to search depends on:
      - branching factor,
      - computational resources,
      - speed requirements,
      - how accurately you can model the utility,
      - etc.
  - Replaces `Utility(s,p)` with `Eval(s,p)`
    - Model utility via an evaluation function

Notes on H-Minimax

- An evaluation function is a mapping from game states to real values
  - $f: s \rightarrow \mathbb{R}$

- Default evaluation function:
  - $f(s) = \begin{cases} \text{Utility}(s, \texttt{MAX}) \; if \; \text{Is}-\text{Terminal}(s) \\ 0 \; otherwise \end{cases}$

No information on quality of non-terminal nodes

  - Determine a function to estimate value that is strongly correlated to actual chances of winning (i.e., the score)
    - Modelling problem (similar to heuristic design problem from informed/local search)

Evaluation Functions

- Determine important features/variables
- Chess example
  - # of pieces (*NPcs*)
  - # of queens (*NQns*)
  - # of controlled squares (*CtlSqs*)
  - # of threatened opponent pieces (*ThrPcs*)
  - etc.

  - $f(s) = w_1 \times NPcs + w_2 \times NQns + w_3 \times CtlSqs + w_4 \times ThrPcs + \ldots$

    Determine values for w_1, ..., w_4, ...

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/02185973212

6 Appendix – Stochastic Games

Stochastic Games

- Many games have randomisation
  - Settlers of Catan
  - Poker
- How do we deal with uncertainty?
  - Can still use Minimax
  - Search tree is larger

Calculate expected value of a state
(MUCH harder than deterministic games)