Perspectives on learning in games

Tutorial — Part II

Gabriele Farina

MIT

✉ gfarina@mit.edu

Athens • 4 July 2024

Imperfect-information Extensive-Form Games

- Games played on a game tree (think chess, go, poker, monopoly, Avalon, Liar's dice, ...)
- Stochastic moves are allowed (random draws of cards, random roll of dice, random arrivals, ...)

We will be mostly interested in the general case of **imperfect-information** games

(i.e., certain moves or stochastic events are only observed by a subset of players)

Difficulties with Extensive-Form Games

Compared to normal-form games, imperfect-information extensive-form games bring many conceptual challenges

Difficulties with Extensive-Form Games

Compared to normal-form games, imperfect-information extensive-form games bring many conceptual challenges

1. The number of (deterministic) strategies grows **exponentially** in the game tree

Difficulties with Extensive-Form Games

Compared to normal-form games, imperfect-information extensive-form games bring many conceptual challenges

1. The number of (deterministic) strategies grows **exponentially** in the game tree

2. Imperfect information makes backward induction and local reasoning not viable

*General principle: you need to think about what the opponents don't know about you and leverage that to your advantage. Sometimes that means **bluffing**, to not reveal private information.*

Difficulties with Extensive-Form Games

Compared to normal-form games, imperfect-information extensive-form games bring many conceptual challenges

1. The number of (deterministic) strategies grows **exponentially** in the game tree

2. Imperfect information makes backward induction and local reasoning not viable

   *General principle: you need to think about what the opponents don't know about you and leverage that to your advantage. Sometimes that means **bluffing**, to not reveal private information.*

3. Other players have control over what part of the game tree is visited/explored

Difficulties with Extensive-Form Games

Compared to normal-form games, imperfect-information extensive-form games bring many conceptual challenges

1. The number of (deterministic) strategies grows **exponentially** in the game tree

2. Imperfect information makes backward induction and local reasoning not viable

   *General principle: you need to think about what the opponents don't know about you and leverage that to your advantage. Sometimes that means **bluffing**, to not reveal private information.*

3. Other players have control over what part of the game tree is visited/explored

- Nonetheless: many positive results 🎉

Imperfect-Information Extensive-Form Games

How it started:

H. W. Kuhn[1]

1950

A fascinating problem for the game theoretician is posed by the common card game, Poker. While generally regarded as partaking of psycho-logical aspects (such as bluffing) which supposedly render it inaccessible to mathematical treatment, it is evident that Poker falls within the general theory of games as elaborated by von Neumann and Morgenstern [1]. Relevant probability problems have been considered by Borel and Ville [2] and several variants are examined by von Neumann [1] and by Bellman and Blackwell [3].

As actually played, Poker is far too complex a game to permit a complete analysis at present; however, this complexity is computational and

Imperfect-Information Extensive-Form Games

How it started:

H. W. Kuhn[1]

1950

A fascinating problem for the game theoretician is posed by the common card game, Poker. While generally regarded as partaking of psycho-logical aspects (such as bluffing) which supposedly render it inaccessible to mathematical treatment, it is evident that Poker falls within the general theory of games as elaborated by von Neumann and Morgenstern [1]. Relevant probability problems have been considered by Borel and Ville [2] and several variants are examined by von Neumann [1] and by Bellman and Blackwell [3].

As actually played, Poker is far too complex a game to permit a complete analysis at present; however, this complexity is computational and

How it's going:

Science Current Issue First release papers Archive About Submit manus

HOME > SCIENCE > VOL. 359, NO. 6374 > SUPERHUMAN AI FOR HEADS-UP NO-LIMIT POKER: LIBRATUS BEATS TOP PROFESSIONALS

RESEARCH ARTICLE

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

NOAM BROWN AND TUOMAS SANDHOLM Authors Info & Affiliations

SCIENCE • 17 Dec 2017 • Vol 359, Issue 6374 • pp. 418-424 • DOI: 10.1126/science.aao1733

4,596 149

2017

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

Then,

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

Then,

- If Player 1 checks, Player 2 can check or bet another \$1 after matching the pot.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth $1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet $1.

Then,

- If Player 1 checks, Player 2 can check or bet another $1 after matching the pot.
    - If Player 2 checks, a showdown occurs; if Player 2 bets, Player 1 can fold or call.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

Then,

- If Player 1 checks, Player 2 can check or bet another \$1 after matching the pot.
    - If Player 2 checks, a showdown occurs; if Player 2 bets, Player 1 can fold or call.
        - If Player 1 folds, Player 2 takes the pot; if Player 1 calls, a showdown occurs.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth $1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet $1.

Then,

- If Player 1 checks, Player 2 can check or bet another $1 after matching the pot.
  - If Player 2 checks, a showdown occurs; if Player 2 bets, Player 1 can fold or call.
    - If Player 1 folds, Player 2 takes the pot; if Player 1 calls, a showdown occurs.
- If Player 1 bets, Player 2 can fold or call the bet by matching the pot.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

In Kuhn poker, each player puts an ante worth \$1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet \$1.

Then,

- If Player 1 checks, Player 2 can check or bet another \$1 after matching the pot.
    - If Player 2 checks, a showdown occurs; if Player 2 bets, Player 1 can fold or call.
        - If Player 1 folds, Player 2 takes the pot; if Player 1 calls, a showdown occurs.
- If Player 1 bets, Player 2 can fold or call the bet by matching the pot.
    - If Player 2 folds, Player 1 takes the pot; if Player 2 calls, a showdown occurs.

How Extensive-Form Games Are Drawn

Example (Kuhn poker).

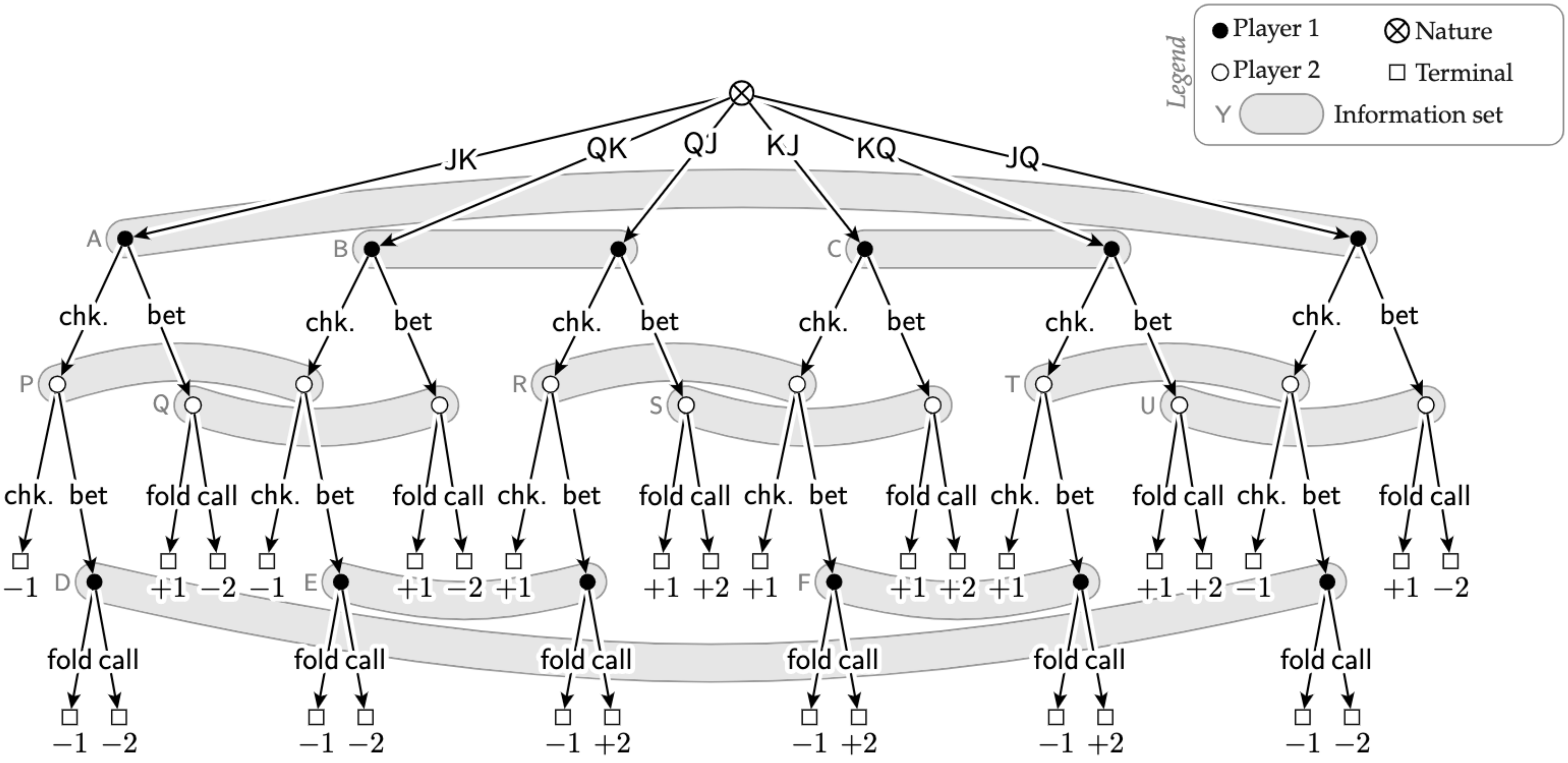
In Kuhn poker, each player puts an ante worth $1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet $1.

Then,

- If Player 1 checks, Player 2 can check or bet another $1 after matching the pot.
    - If Player 2 checks, a showdown occurs; if Player 2 bets, Player 1 can fold or call.
        - If Player 1 folds, Player 2 takes the pot; if Player 1 calls, a showdown occurs.
- If Player 1 bets, Player 2 can fold or call the bet by matching the pot.
    - If Player 2 folds, Player 1 takes the pot; if Player 2 calls, a showdown occurs.

When a showdown occurs, the player with the higher card wins the pot and the game immediately ends

How Extensive-Form Games Are Drawn



As noted by Kuhn himself, even the previous small game already captures central aspects of deceptive behavior

> The presence of <u>bluffing</u> and <u>underbidding</u> in these solutions is noteworthy (<u>bluffing</u> means betting with a J ; <u>underbidding</u> means passing on a K). All but the extreme strategies for player I, in terms of the behavior parameters, involve both bluffing and underbidding while player II's single optimal strategy instructs him to bluff with constant probability 1/3 (underbidding is not available to him). These results compare

A Bit of Nomenclature

A Bit of Nomenclature

- The nodes of the game tree are often called **histories** (will be denoted with letter h)

A Bit of Nomenclature

- The nodes of the game tree are often called **histories** (will be denoted with letter h)
- The collection of information sets for a given player is called the **information partition** of the player

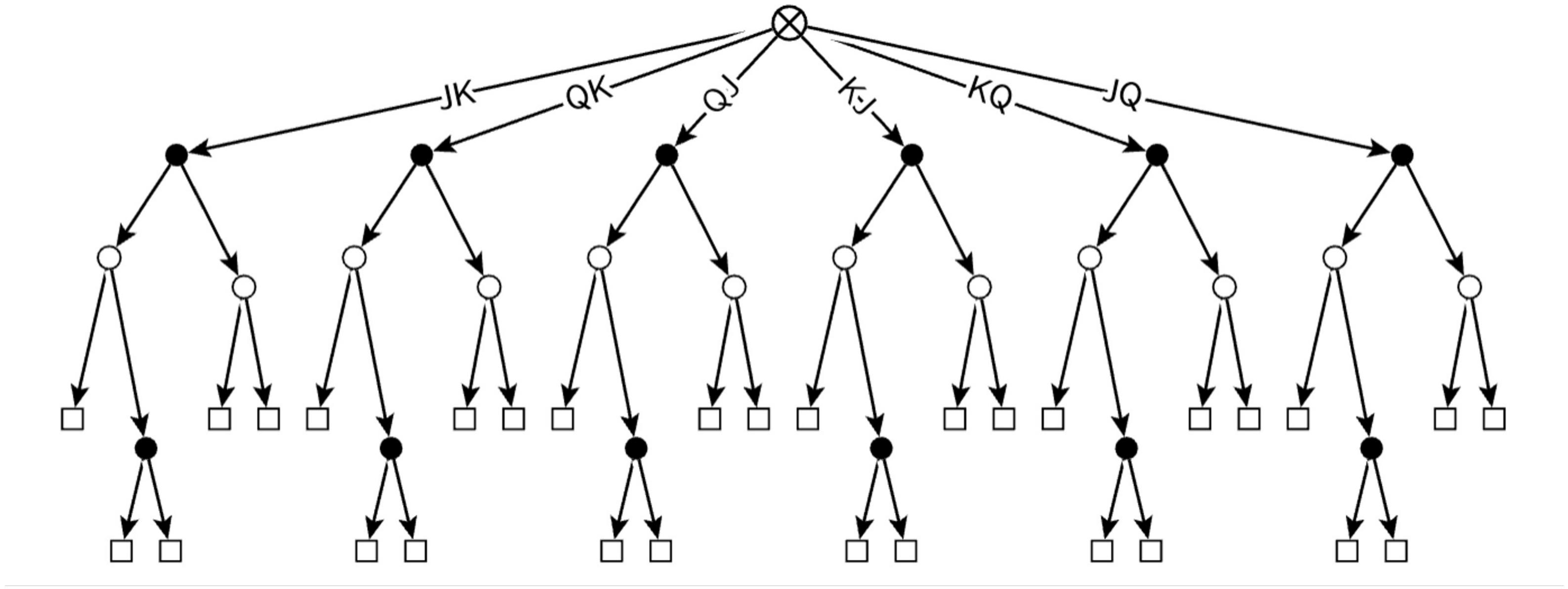
A Bit of Nomenclature

- The nodes of the game tree are often called **histories** (will be denoted with letter h)
- The collection of information sets for a given player is called the **information partition** of the player
- The game has **perfect information** if all information sets are singleton

The structure of Information

■ **First variation**

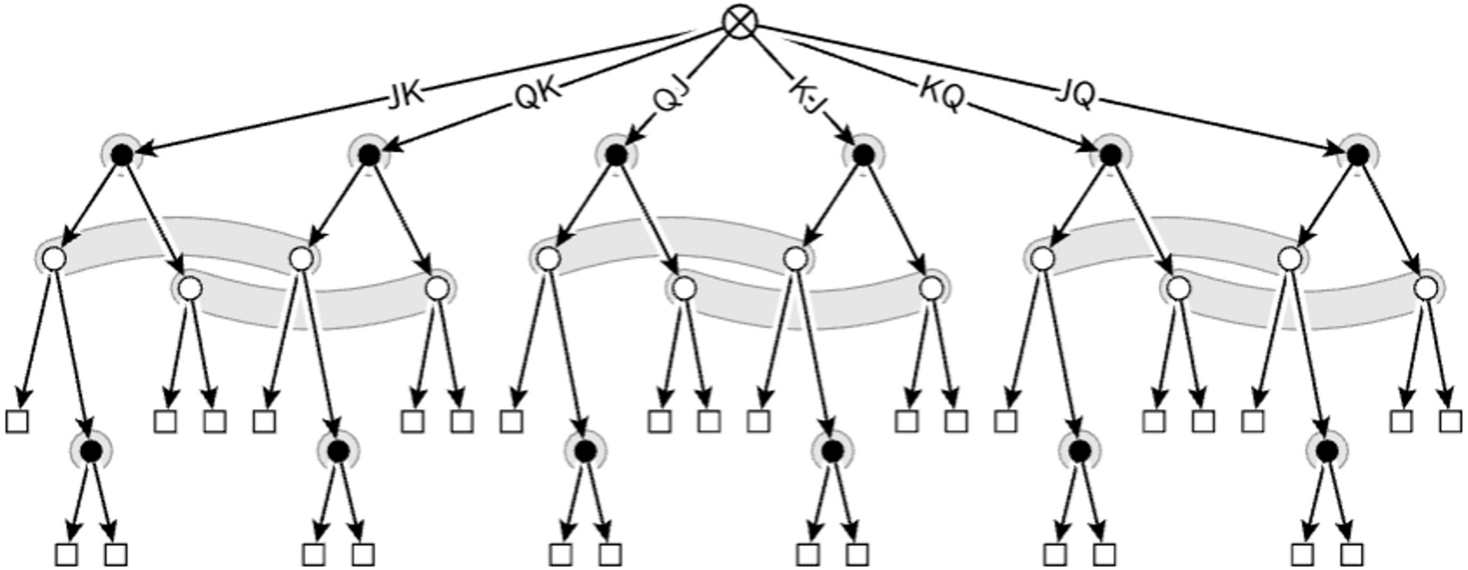
Player 1 is revealed the private card of Player 2 by the dealer.



The structure of Information

■ **First variation**

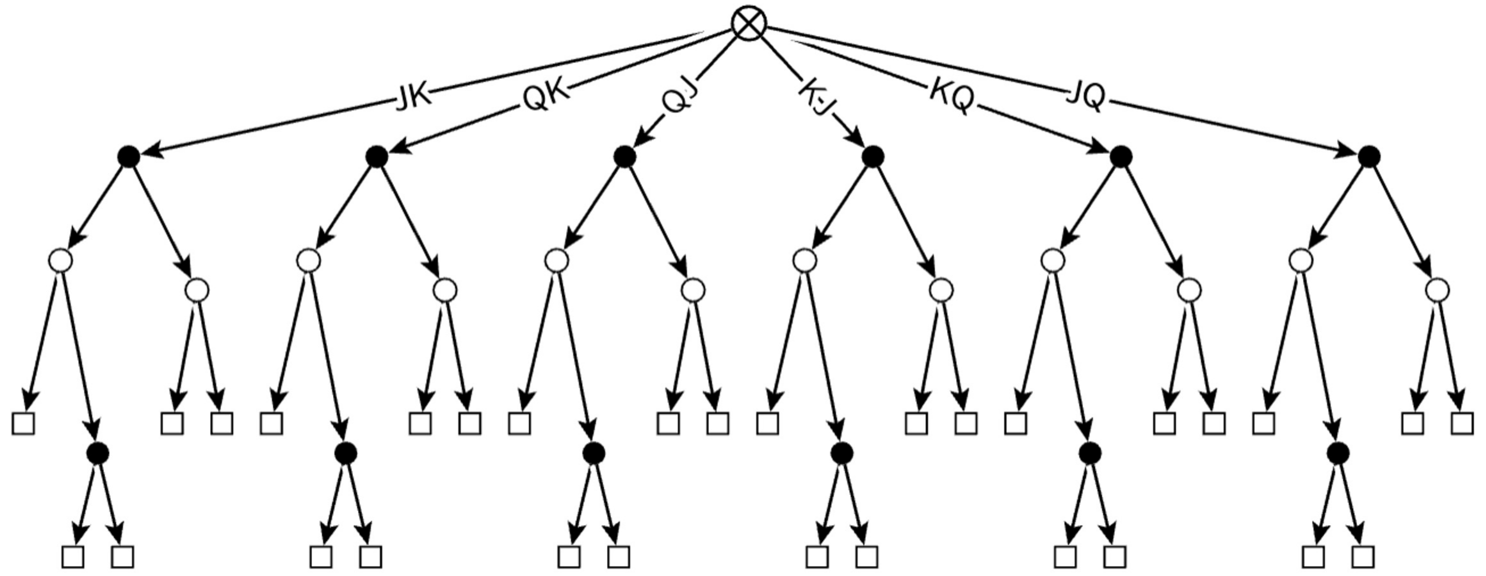
Player 1 is revealed the private card of Player 2 by the dealer.



The structure of Information

- **Second variation**

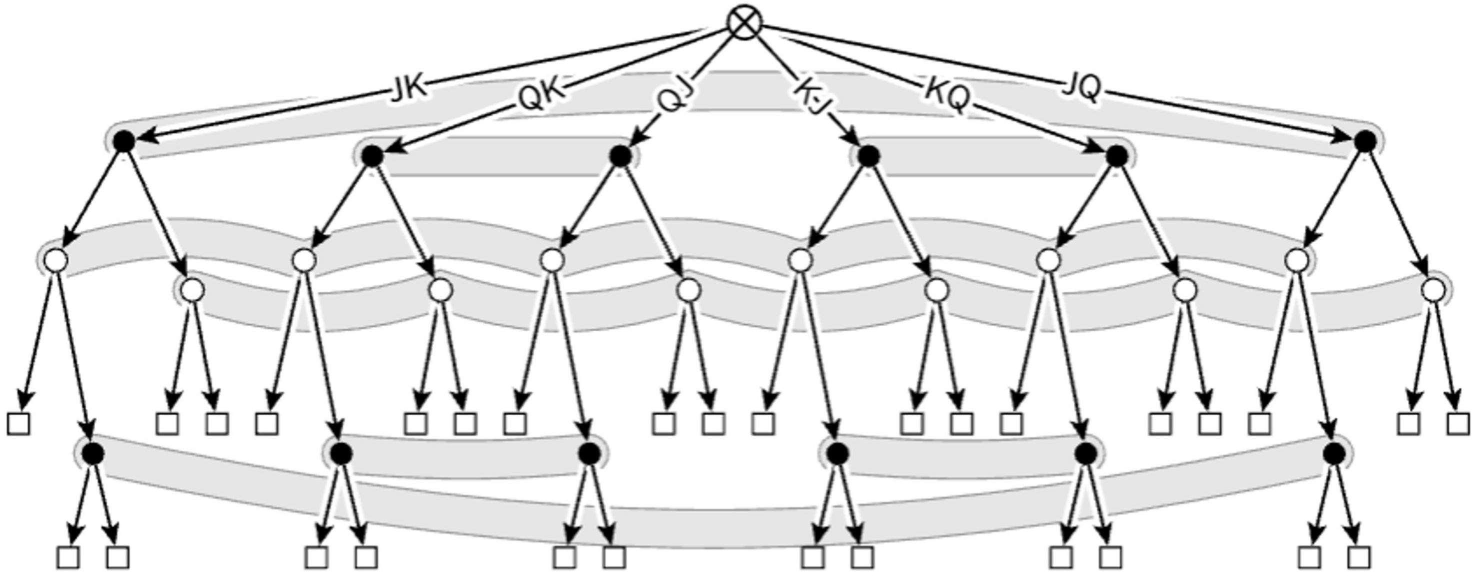
Player 2 does not get to observe her private card.



The structure of Information

■ **Second variation**

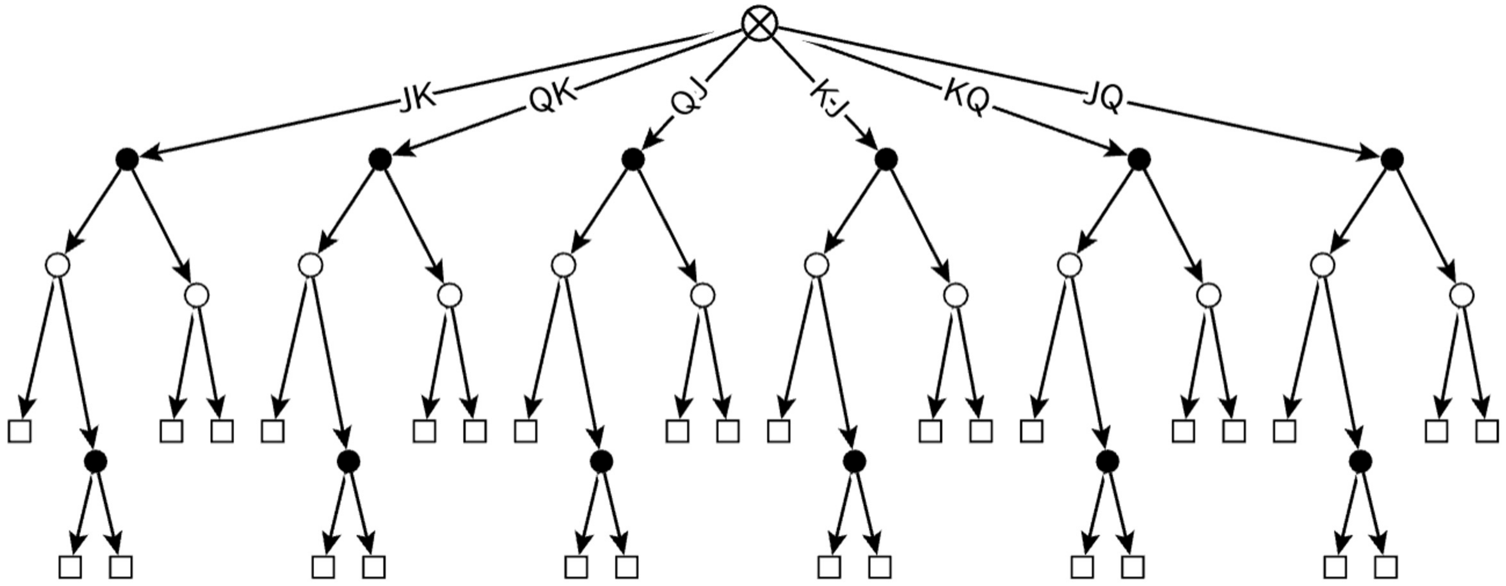
Player 2 does not get to observe her private card.



The structure of Information

■ **Third variation**

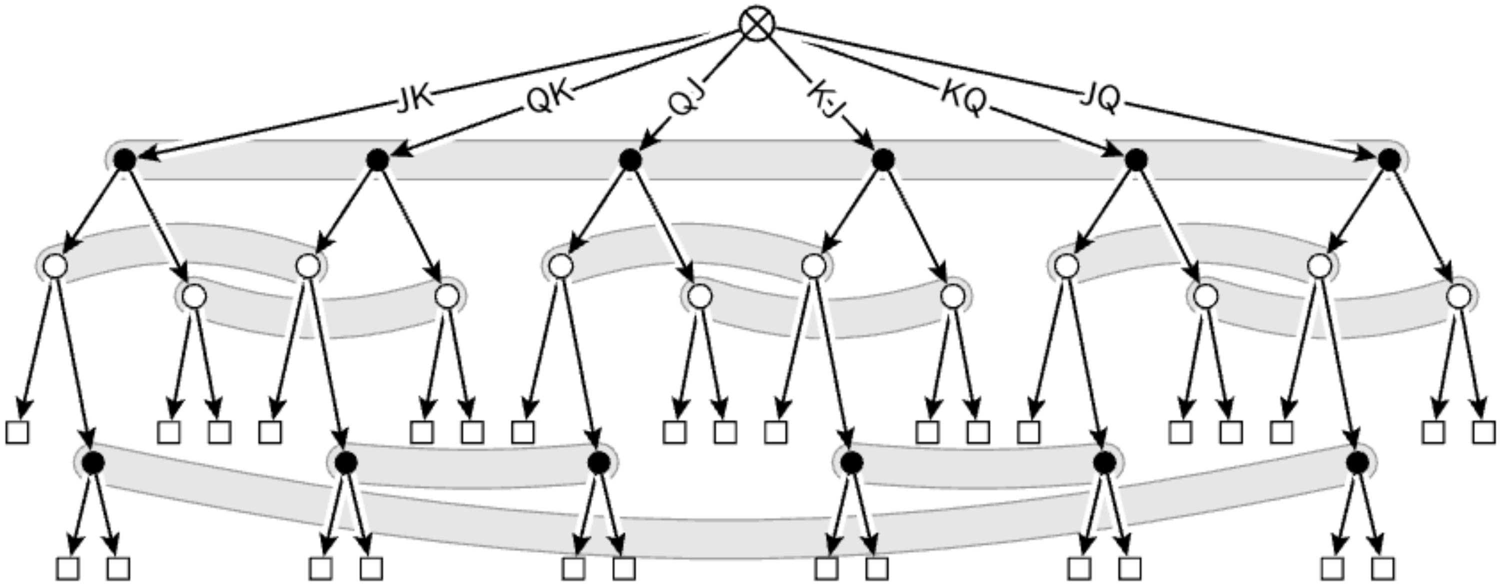
Player 1 is allowed to look at his private card only if he decides to check.



The structure of Information

■ **Third variation**

Player 1 is allowed to look at his private card only if he decides to check.



Perfect vs Imperfect Recall

Perfect Recall: information sets satisfy the fact that that no player forgets about their actions, and about information once acquired

Perfect vs Imperfect Recall

Perfect Recall: information sets satisfy the fact that that no player forgets about their actions, and about information once acquired

🚨 **Danger zone™:** unexpected things happen when trying to formalize optimal strategies in the presence of imperfect recall

Sleeping Beauty problem

文A 12 languages

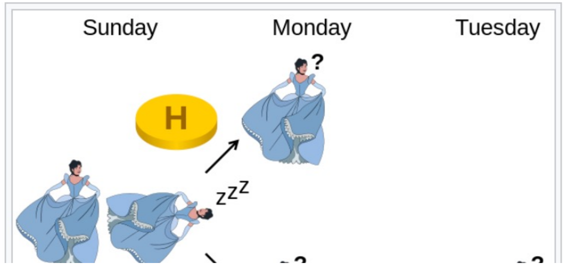
Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

The **Sleeping Beauty problem** is a puzzle in decision theory in which whenever an ideally rational epistemic agent is awoken from sleep, they have no memory of whether they have been awoken before. Upon being told that they have been woken once or twice according to the toss of a coin, once if heads and twice if tails, they are asked their degree of belief for the coin having come up heads.

History [edit]

The problem was originally formulated in unpublished work in the mid-1980s by Arnold Zuboff (the work was later published as "One Self: The Logic of Experience")[1] followed by a paper by Adam Elga.[2] A formal analysis of the problem of belief formation in decision problems with imperfect recall was provided first by Michele Piccione and Ariel Rubinstein in their paper: "On the Interpretation of Decision Problems with Imperfect Recall" where the "paradox of the absent



Perfect vs Imperfect Recall

Perfect Recall: information sets satisfy the fact that that no player forgets about their actions, and about information once acquired

More formally:

Perfect vs Imperfect Recall

Perfect Recall: information sets satisfy the fact that that no player forgets about their actions, and about information once acquired

More formally:

> A player $i \in [\![n]\!]$ is said to have *perfect recall* if, for any information set $I \in \mathcal{I}_i$, for any two histories $h, h' \in I$ the sequence of Player i's actions encountered along the path from the root to h and from the root to h' must coincide (or otherwise Player i would be able to distinguish among the histories, since the player remembers all of the actions they played in the past). The game is perfect recall if all players have perfect recall.

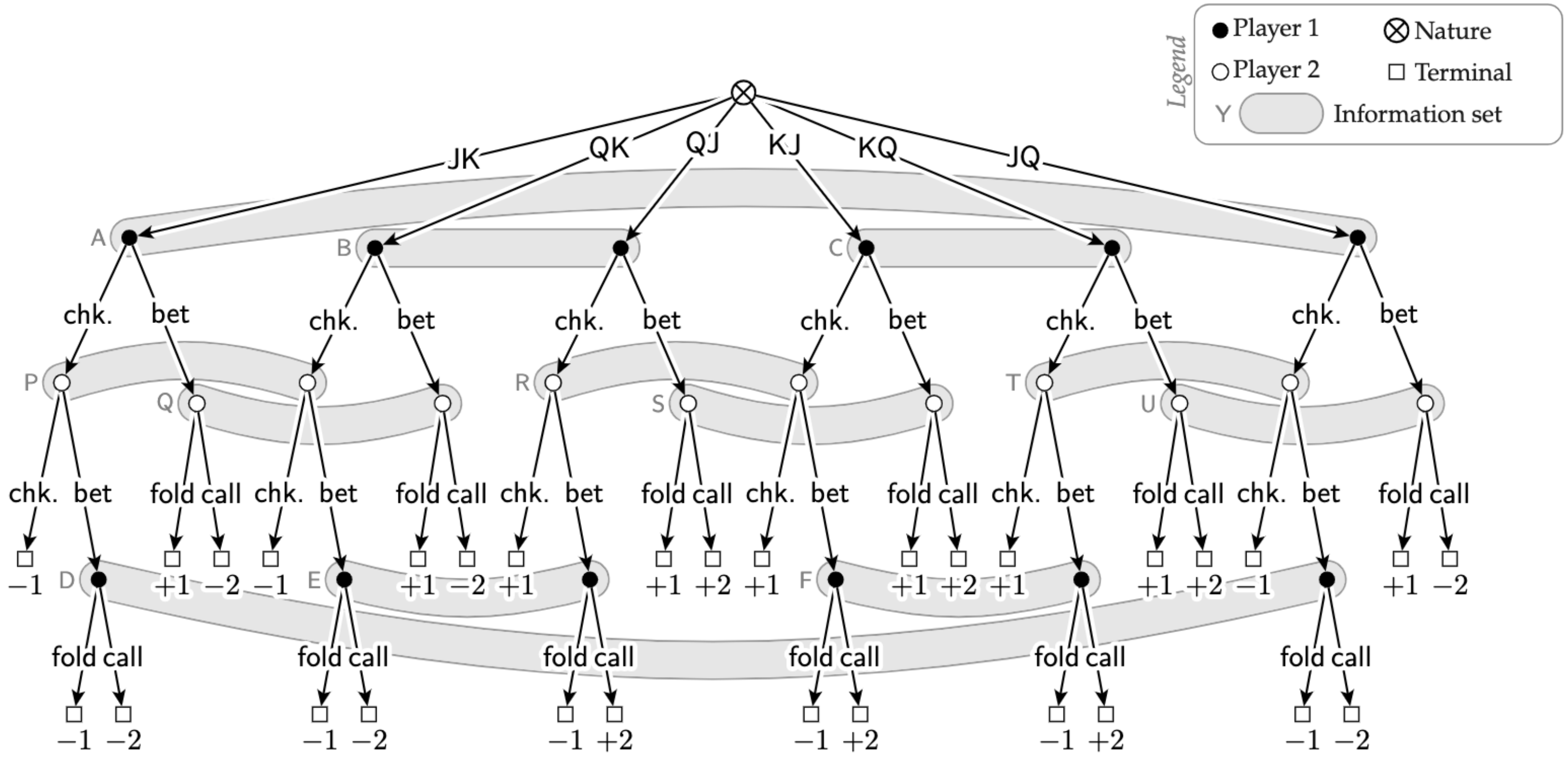
Strategies in Extensive-Form Games

Approach 1: Convert to Normal-Form Game
(aka "reduced normal-form representation")

Approach 2: The RL way:
"Behavioral Strategies"

Strategic Form

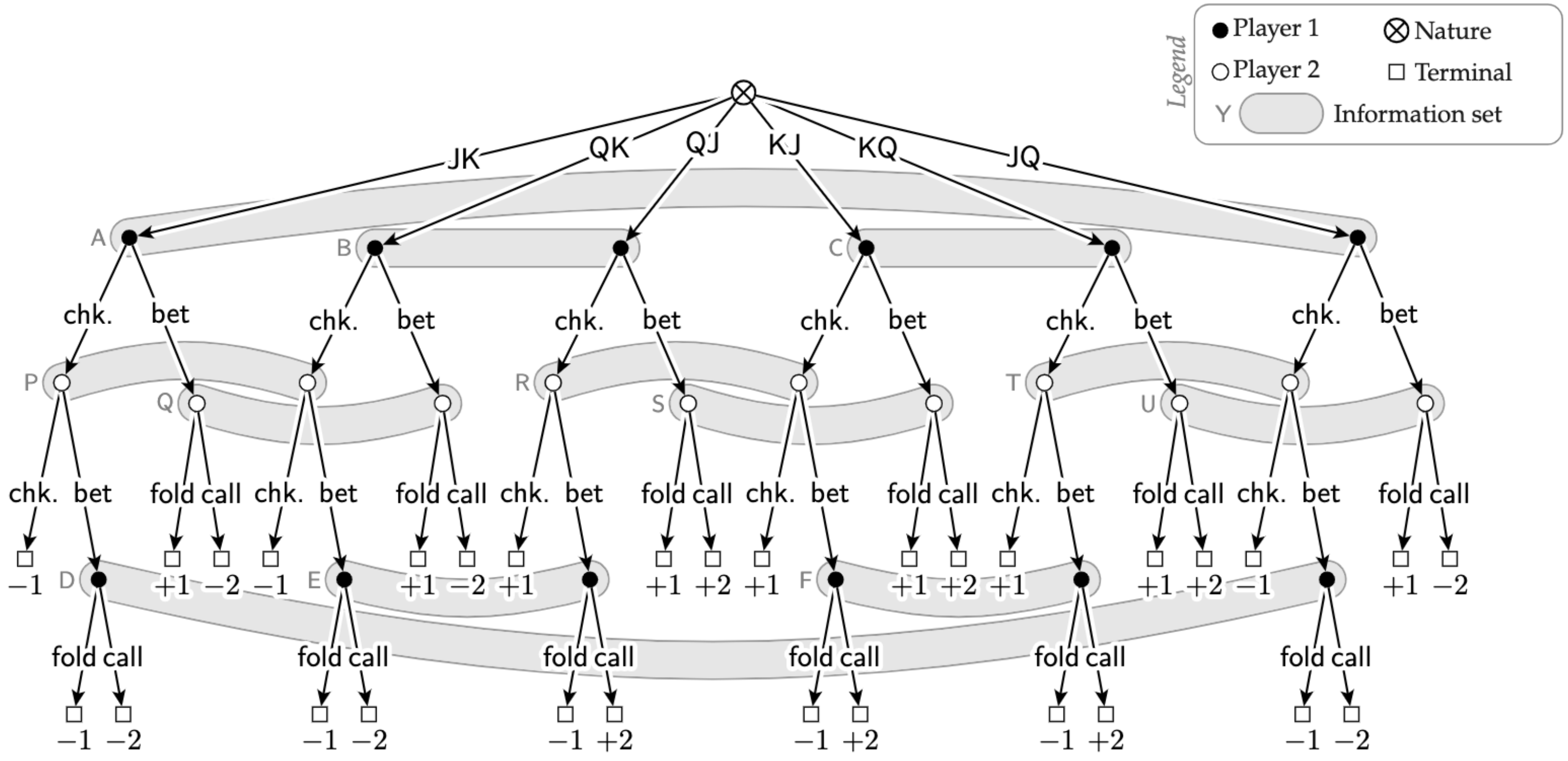
Idea: Strategy = randomize a deterministic contingency plan



Each player constructs a list of all possible assignments of actions at each information set

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

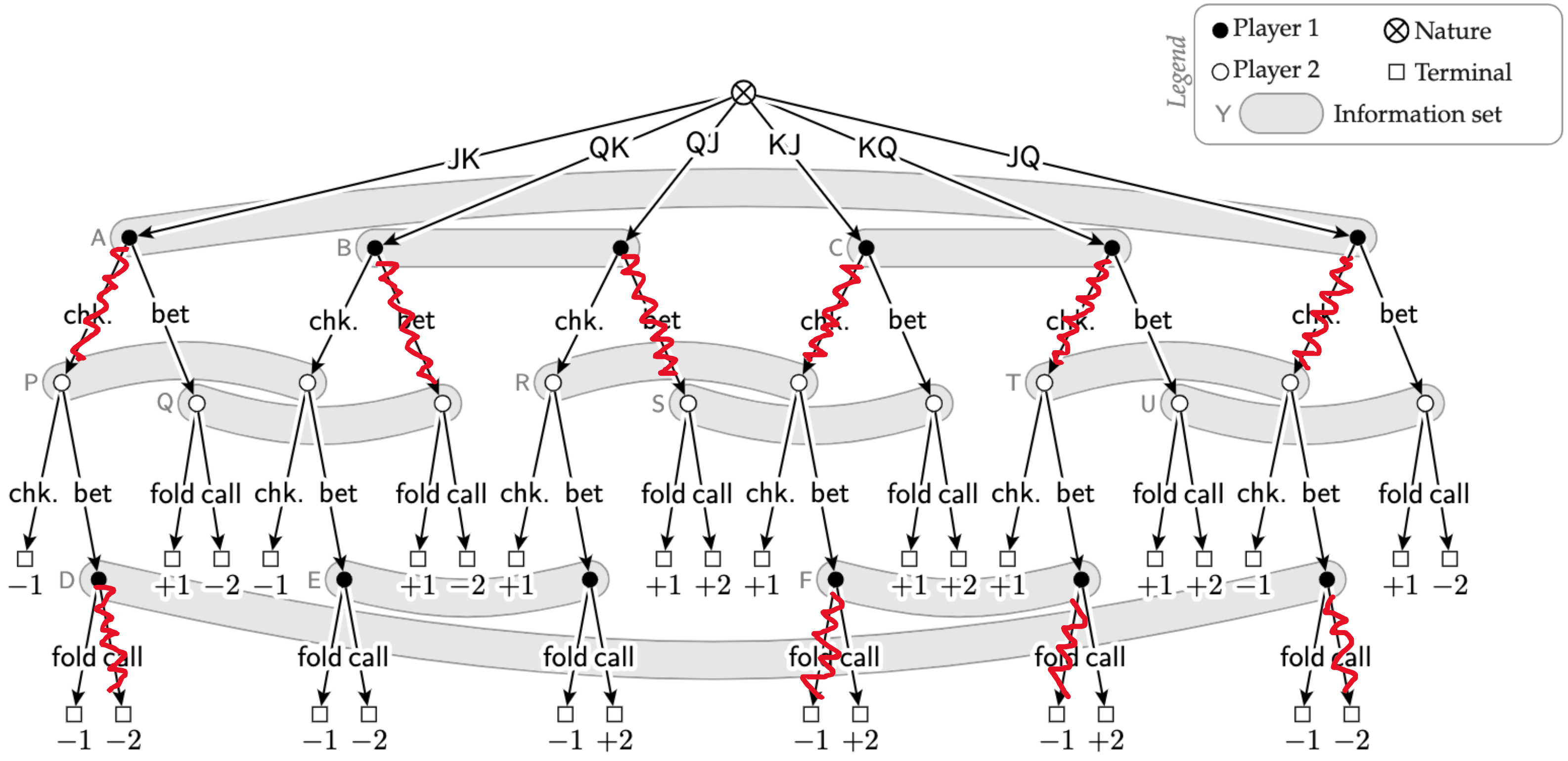


Each player constructs a list of all possible assignments of actions at each information set

Histories in the same information must get assigned the same action

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

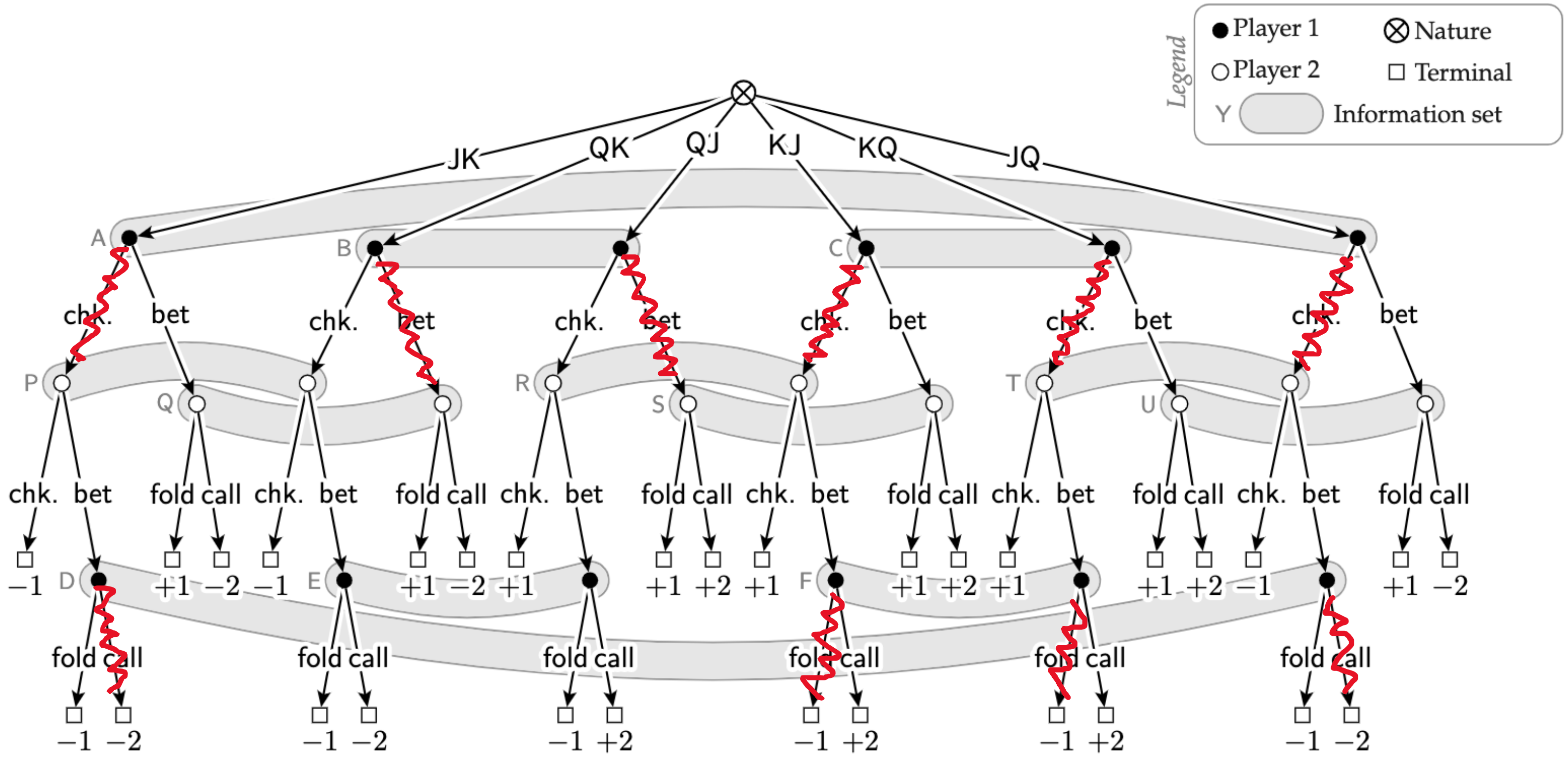


Each player constructs a list of all possible assignments of actions at each information set

Histories in the same information must get assigned the same action

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan



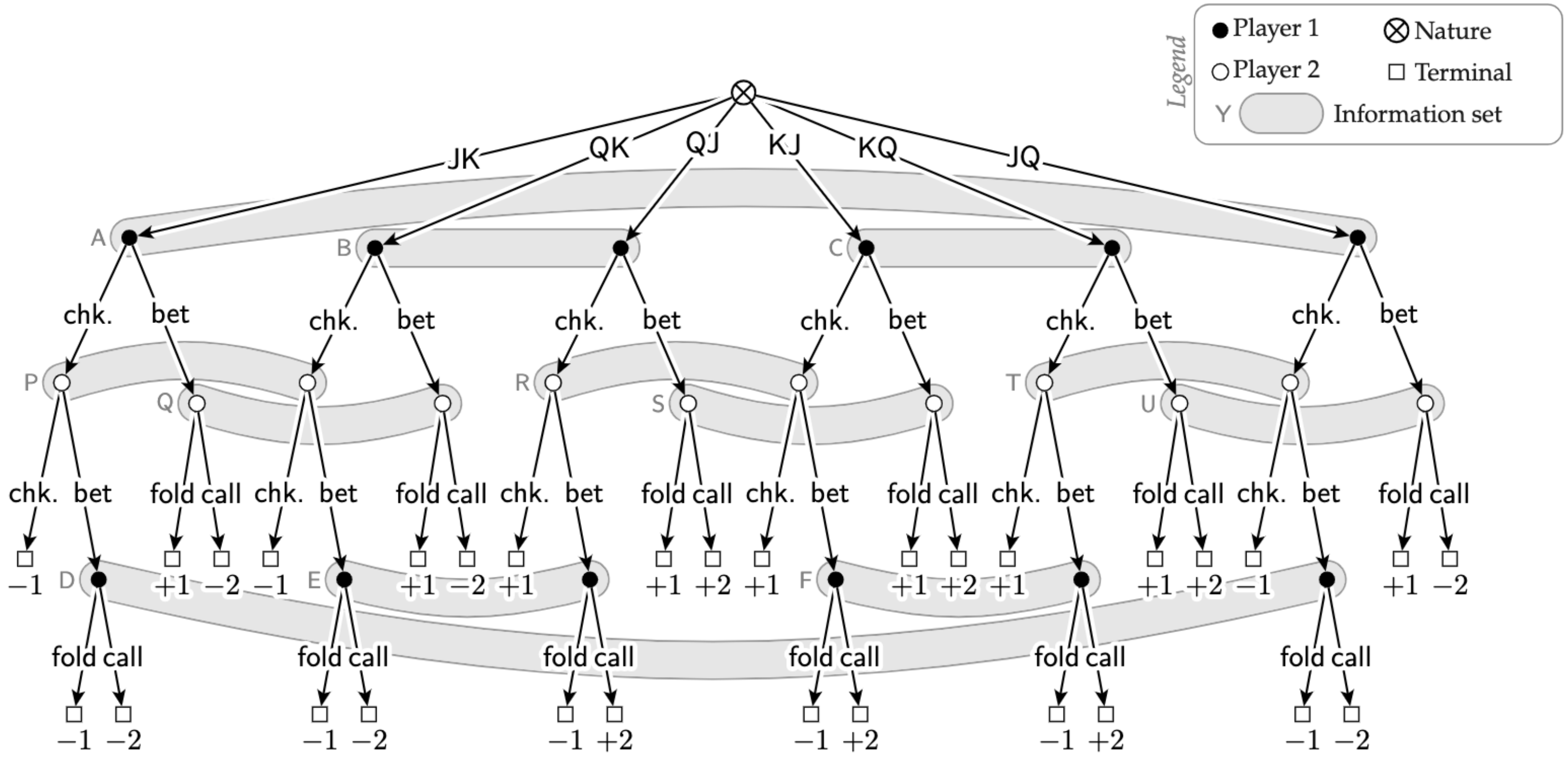
Each player constructs a list of all possible assignments of actions at each information set

Histories in the same information must get assigned the same action

No need to specify actions at histories that are for sure unreachable

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

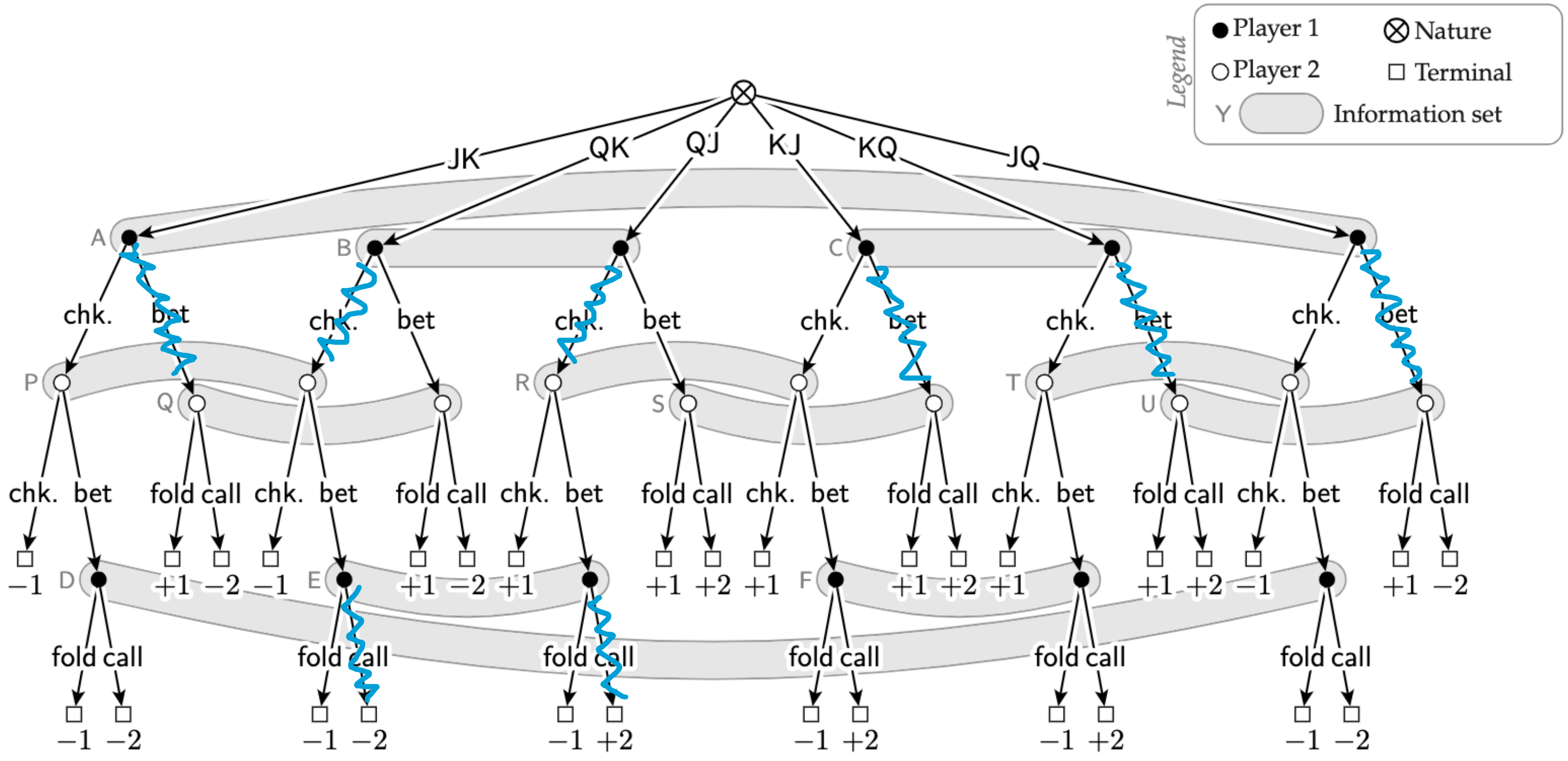


Each player constructs a list of all possible assignments of actions at each information set

(Histories in the same information must get assigned the same action)

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

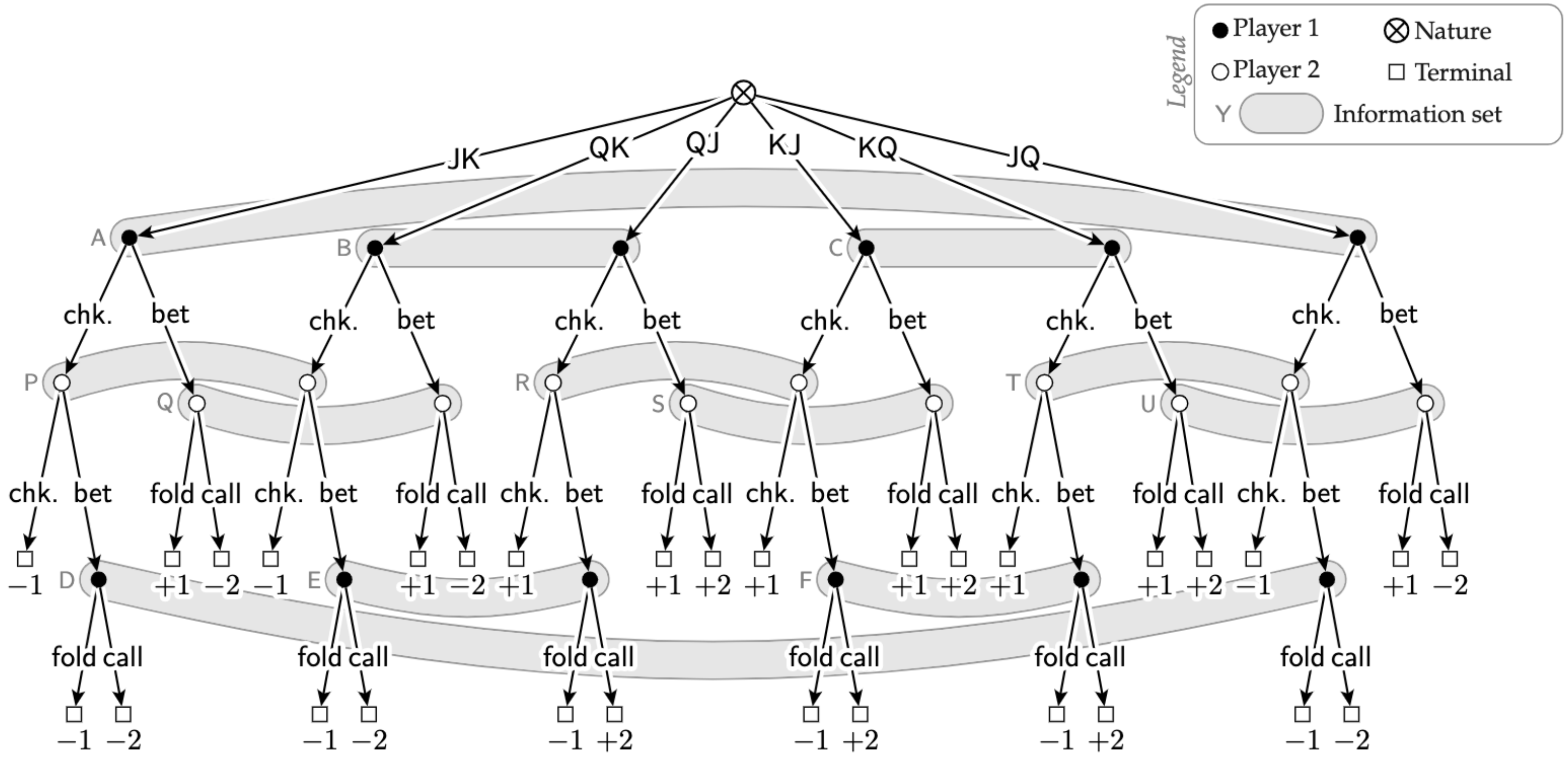


Each player constructs a list of all possible assignments of actions at each information set

(Histories in the same information must get assigned the same action)

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

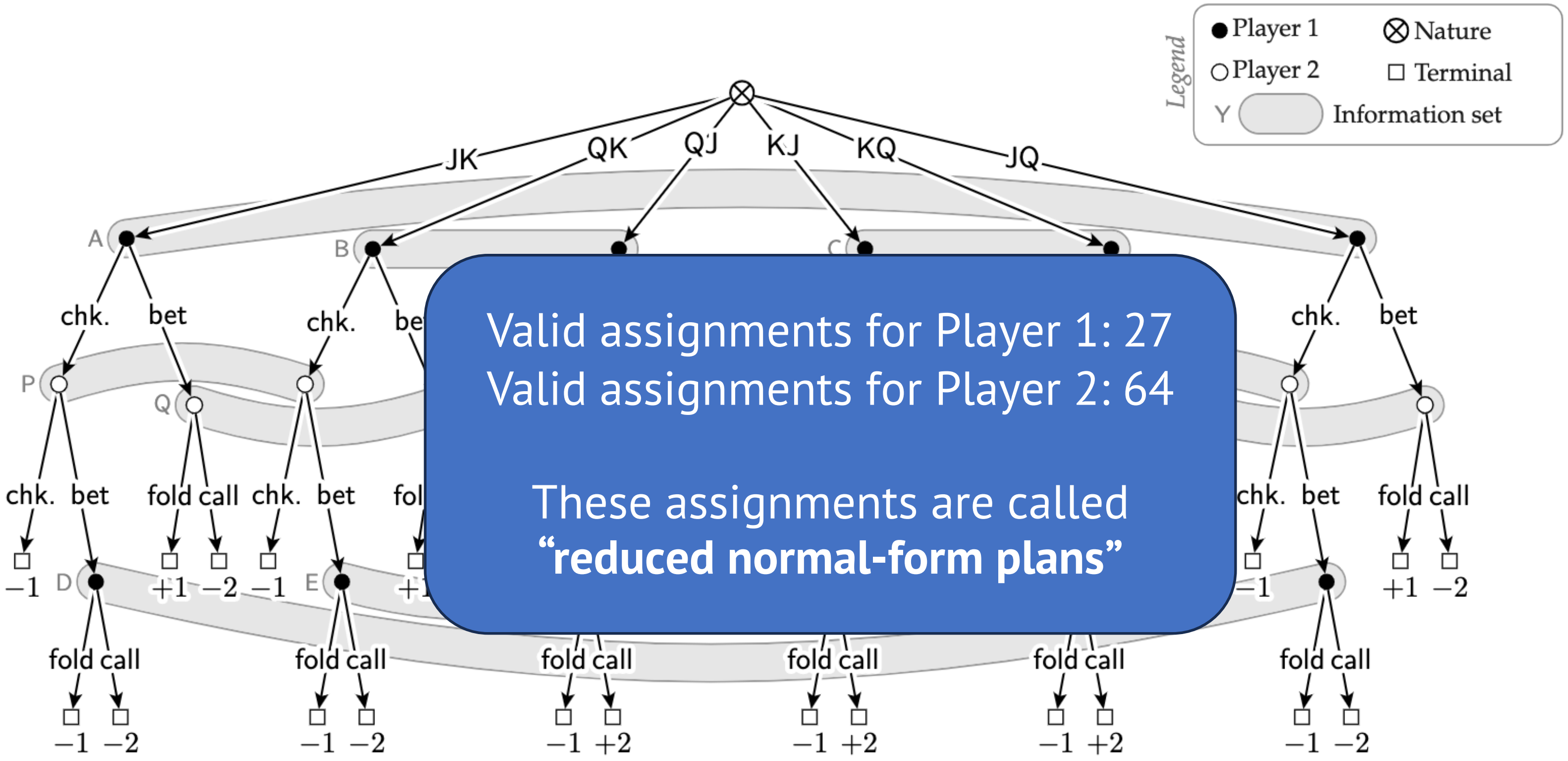


Each player constructs a list of all possible assignments of actions at each information set

(Histories in the same information must get assigned the same action)

Strategic Form

Idea: Strategy = randomize a deterministic contingency plan



Valid assignments for Player 1: 27
Valid assignments for Player 2: 64

These assignments are called **"reduced normal-form plans"**

Each player constructs a list of all possible assignments of actions at each information set

(Histories in the same information must get assigned the same action)

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0	-1/3	…	1/2
0	1/3	0	…	0
-1/3	2/3	1/2	…	0
⋮	⋮	⋮		⋮
1/2	0	-2/3	…	-1/2

(27 x 64 matrix)

Payoff matrix: Each cell contains the expected utility when players use that combination of reduced normal-form plans

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0	-1/3	⋯	1/2
0	1/3	0	⋯	0
-1/3	2/3	1/2	⋯	0
⋮	⋮	⋮		⋮
1/2	0	-2/3	⋯	-1/2

(27 x 64 matrix)

Payoff matrix: Each cell contains the expected utility when players use that combination of reduced normal-form plans

Don't forget nature moves

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0	-1/3	…	1/2
0	1/3	0	…	0
-1/3	2/3	1/2	…	0
⋮	⋮	⋮		⋮
1/2	0	-2/3	…	-1/2

(27 x 64 matrix)

Payoff matrix: Each cell contains the expected utility when players use that combination of reduced normal-form plans

Don't forget nature moves

With this, we have reduced the extensive-form game to a normal-form game ("**reduced normal form of the extensive-form game**")

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0	-1/3	⋯	1/2
0	1/3	0	⋯	0
-1/3	2/3	1/2	⋯	0
⋮	⋮	⋮		⋮
1/2	0	-2/3	⋯	-1/2

(27 x 64 matrix)

Payoff matrix: Each cell contains the expected utility when players use that combination of reduced normal-form plans

Don't forget nature moves

With this, we have reduced the extensive-form game to a normal-form game ("**reduced normal form of the extensive-form game**")

Inherit notions of Nash, correlated equilibrium, coarse correlated equilibrium, ...

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0	-1/3	⋯	1/2
0	1/3	0	⋯	0
-1/3	2/3	1/2	⋯	0
⋮	⋮	⋮		⋮
1/2	0	-2/3	⋯	-1/2

(27 x 64 matrix)

Example: Nash equilibrium in Kuhn poker:

$$\max_{x} \min_{y} x^T A y$$

Distribution over the 27 plans of Player 1

Distribution over the 64 plans of Player 2

Payoff matrix on the left

You can use any technique for normal-form games: learning, linear programming, …

Equivalent Normal-Form Game

Reduced normal-form plans for Player 2

Reduced normal-form plans for Player 1

1/3	0			
0	1/3			
-1/3	2/3			
⋮	⋮			
1/2	0	-2/3	…	-1/2

(27 x 64 matrix)

Big issue: the number of reduced normal-form plans scales exponentially with the game tree size!

This approach is not scalable beyond very small games

We need better techniques

Example: Nash equilibrium in Kuhn

er:

n $x^T Ay$

Payoff matrix on the left

the 27 plans of Player 1

Distribution over the 64 plans of Player 2

You can use any technique for normal-form games: learning, linear programming, …

Quick Aside

Recent discovery: for certain algorithms, we can actually get around the exponential size and still operate in this exponential representation implicitly via a kernel trick

[Farina et al., 2022] Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Gam

Quick Aside

Recent discovery: for certain algorithms, we can actually get around the exponential size and still operate in this exponential representation implicitly via a kernel trick

Specifically, this applies to the multiplicative weights update (MWU) algorithm.

[Farina et al., 2022] Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Gam

Quick Aside

Recent discovery: for certain algorithms, we can actually get around the exponential size and still operate in this exponential representation implicitly via a kernel trick

Specifically, this applies to the multiplicative weights update (MWU) algorithm.

Takeaway

Running MWU on the reduced normal-form representation of an extensive-form game can be done in linear time per iteration in the size of the game tree (as opposed to linear in the number of reduced normal-form plans)

[Farina et al., 2022] Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Gam

Quick Aside

Recent discovery: for certain algorithms, we can actually get around the exponential size and still operate in this exponential representation implicitly via a kernel trick

Specifically, this applies to the multiplicative weights update (MWU) algorithm.

Takeaway

Running MWU on the reduced normal-form representation of an extensive-form game can be done in linear time per iteration in the size of the game tree (as opposed to linear in the number of reduced normal-form plans)

We can use this technique to compute Nash eq. (in two-player zero-sum games) and coarse correlated equilibrium

[Farina et al., 2022] Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Gam

Recap on Normal-Form Strategies

	Idea	Obvious downsides	Good news
(Reduced) Normal-form strategies	Distribution over deterministic strategies $\mu \in \Delta(Plans)$	Exponentially-sized object	In rare cases, it's possible to operate implicitly on the exponential object via a kernel trick

Behavioral Strategies

Idea: Strategy = choice of distribution over available actions at each "decision point"

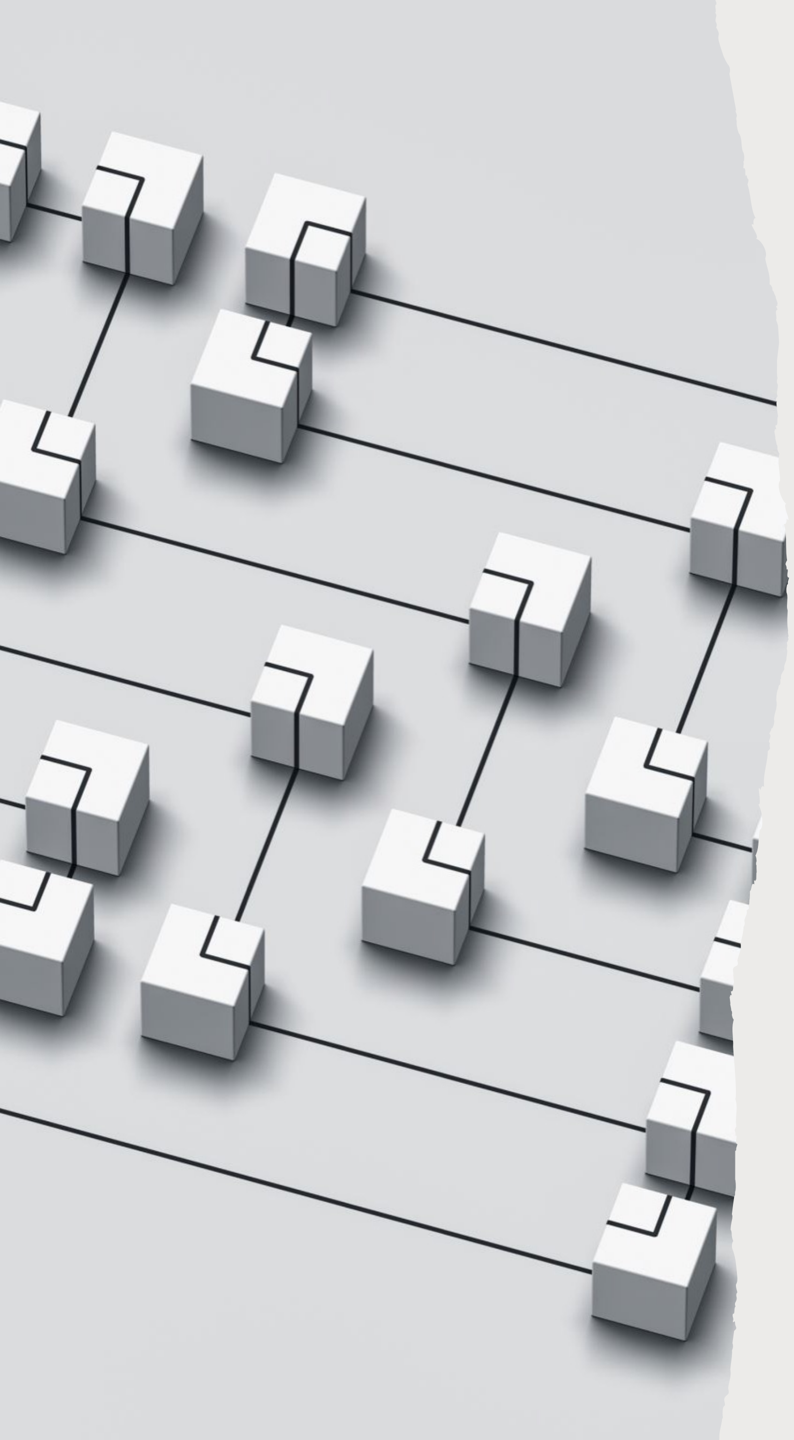
Let's introduce some notation for the tree-form decision process faced by each player…

Behavioral Strategies

Idea: Strategy = choice of distribution over available actions at each "decision point"

Information set

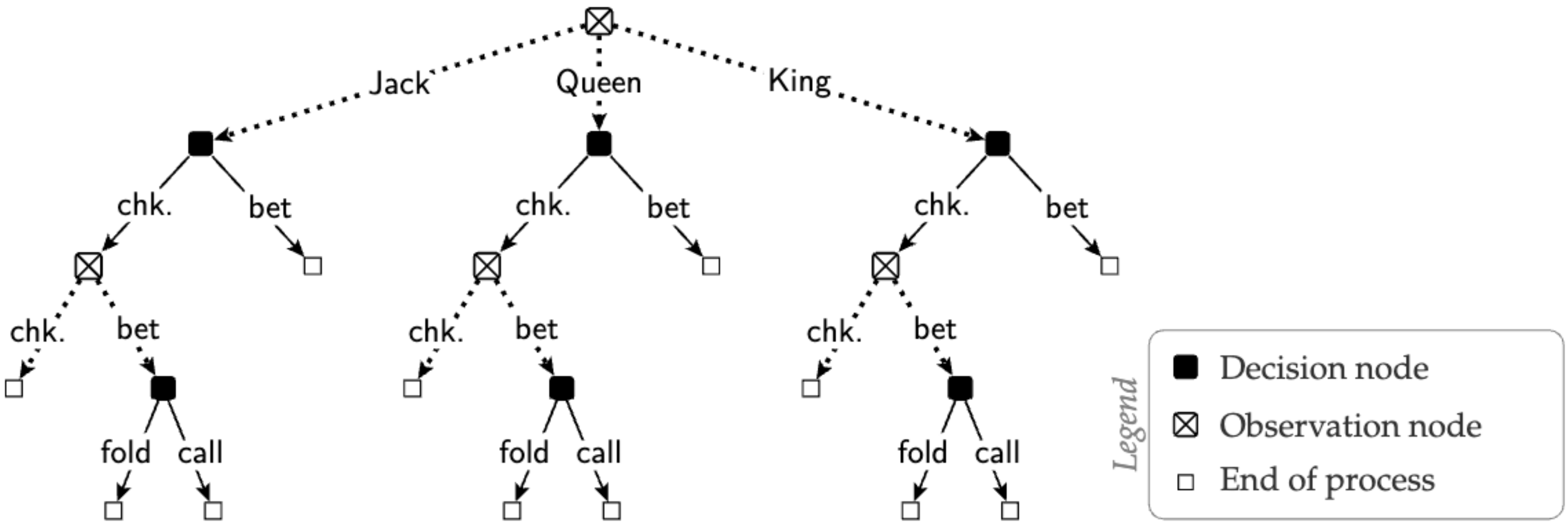
Let's introduce some notation for the tree-form decision process faced by each player…



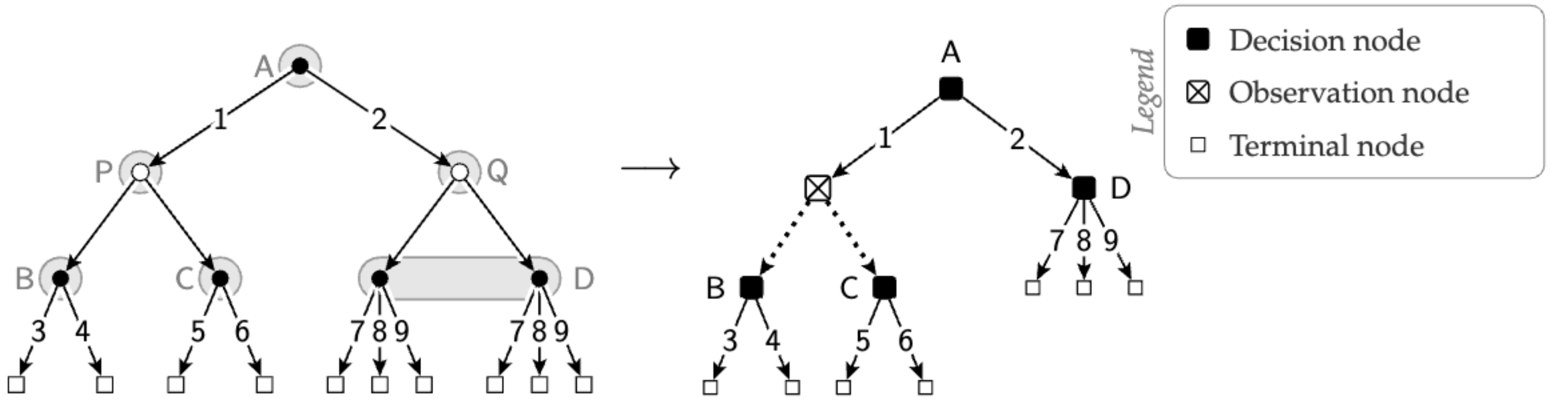
Tree-form Decision Processes

- The **game tree** is a description of the global dynamics of the game, without taking the side of any player in particular
- The problem faced by an individual player is called a tree-form decision process
- TFDP provides a more natural formalism for defining player-specific quantities and procedures, such as strategies and learning algorithms, that inherently refer to the decision space that one player faces while playing the game
- From the point of view of each player, two types of nodes: **decision points** and **observation points**

Example in Kuhn Poker (Player 1)

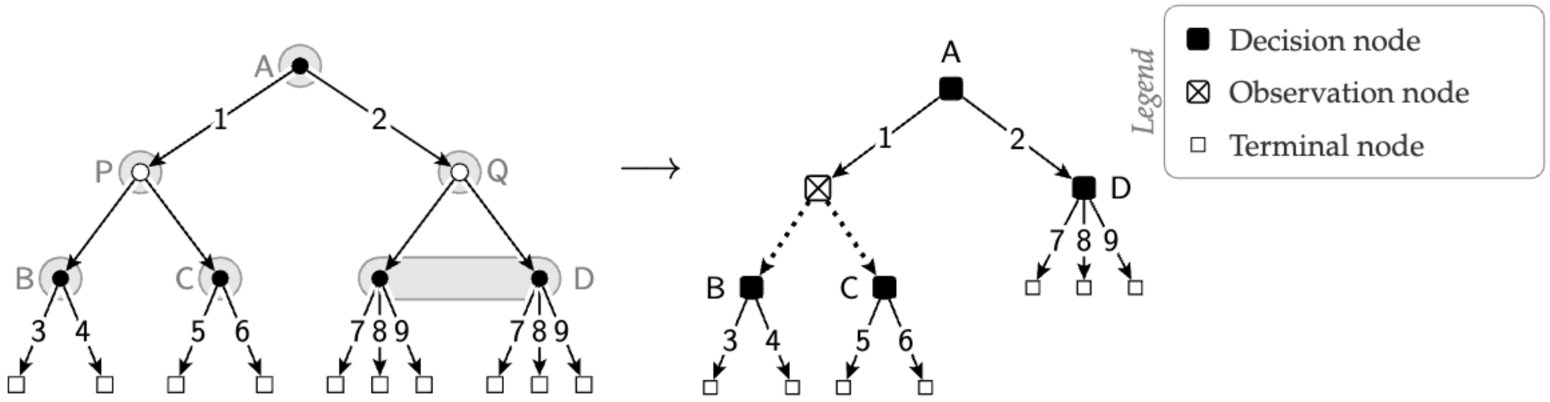


Another Example



Algorithm for constructing the tree-form decision process of a player:

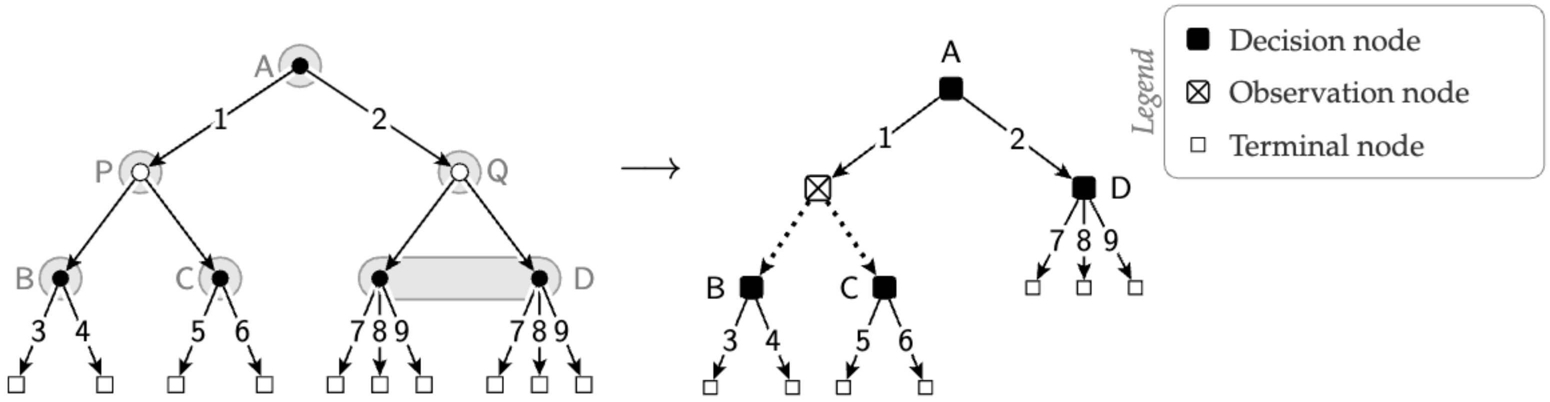
Another Example



Algorithm for constructing the tree-form decision process of a player:

1. For each information set of the player, construct a corresponding decision node

Another Example



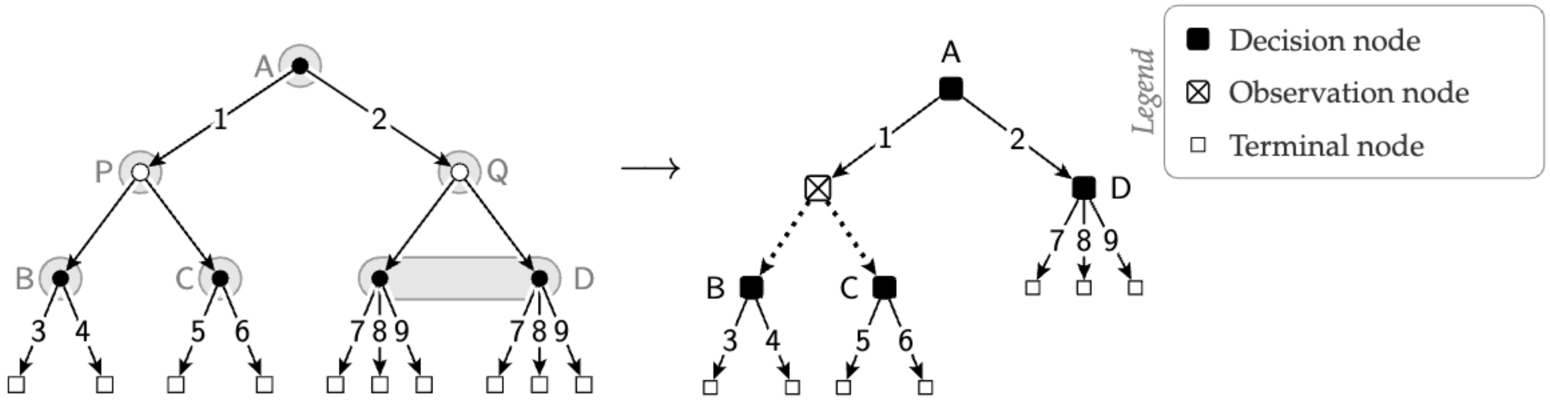
Algorithm for constructing the tree-form decision process of a player:

1. For each information set of the player, construct a corresponding decision node

2. The parent of each decision node is the last action of the player on the path from the root of the game tree to any node of the information set

💡 Does not matter which one when the player has perfect recall! (why?)

Another Example



Algorithm for constructing the tree-form decision process of a player:

1. For each information set of the player, construct a corresponding decision node

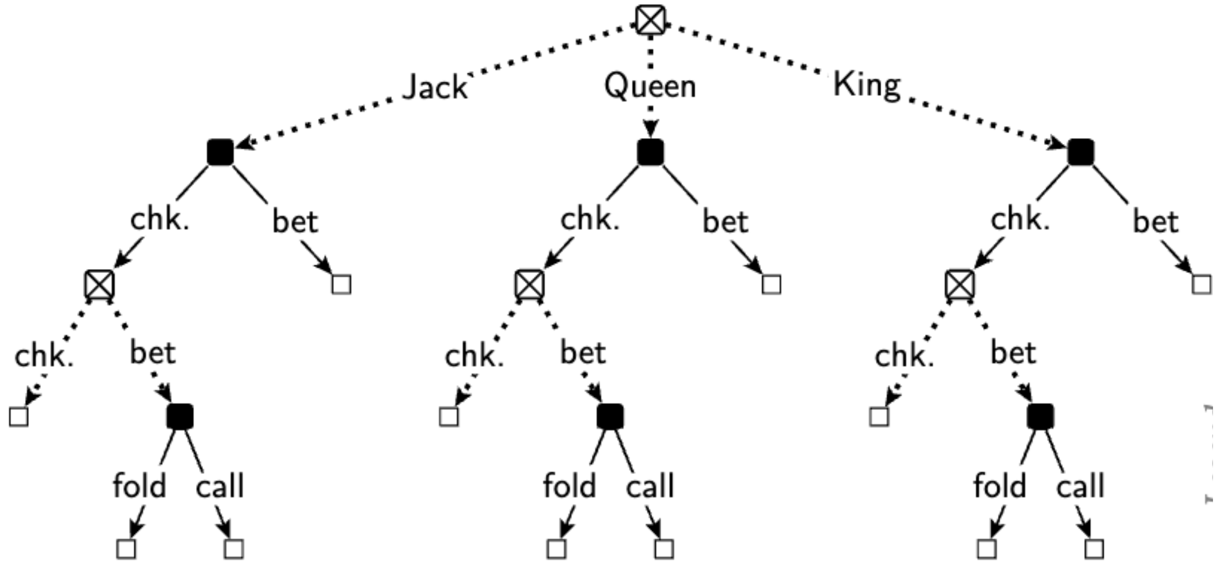
2. The parent of each decision node is the last action of the player on the path from the root of the game tree to any node of the information set

💡 Does not matter which one when the player has perfect recall! (why?)

3. If multiple decision nodes want to have the same parent action, connect with an observation node

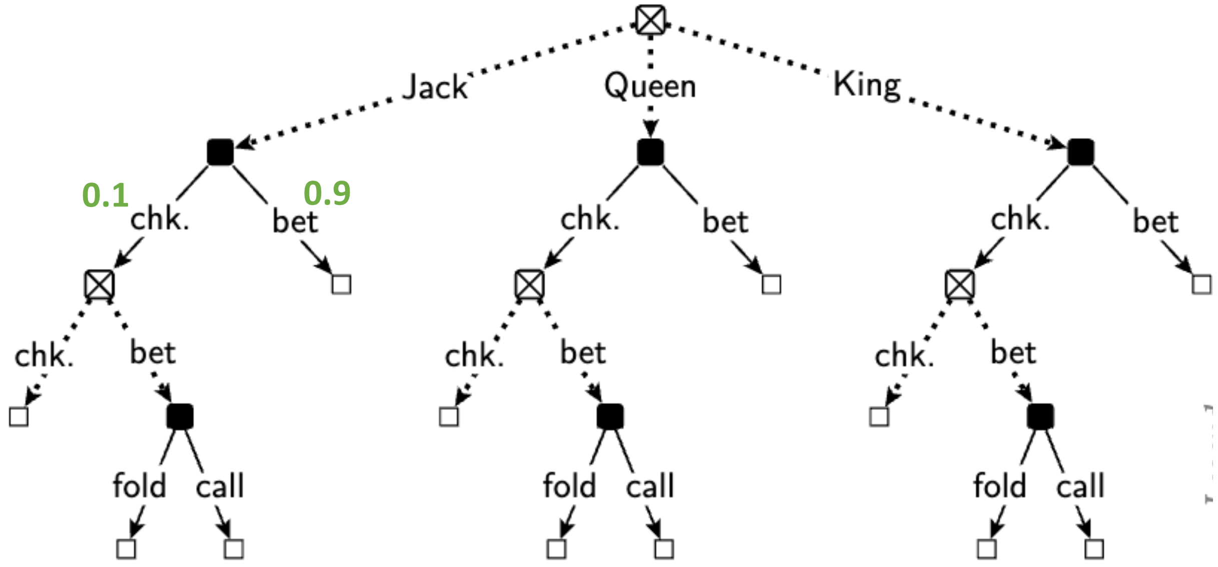
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



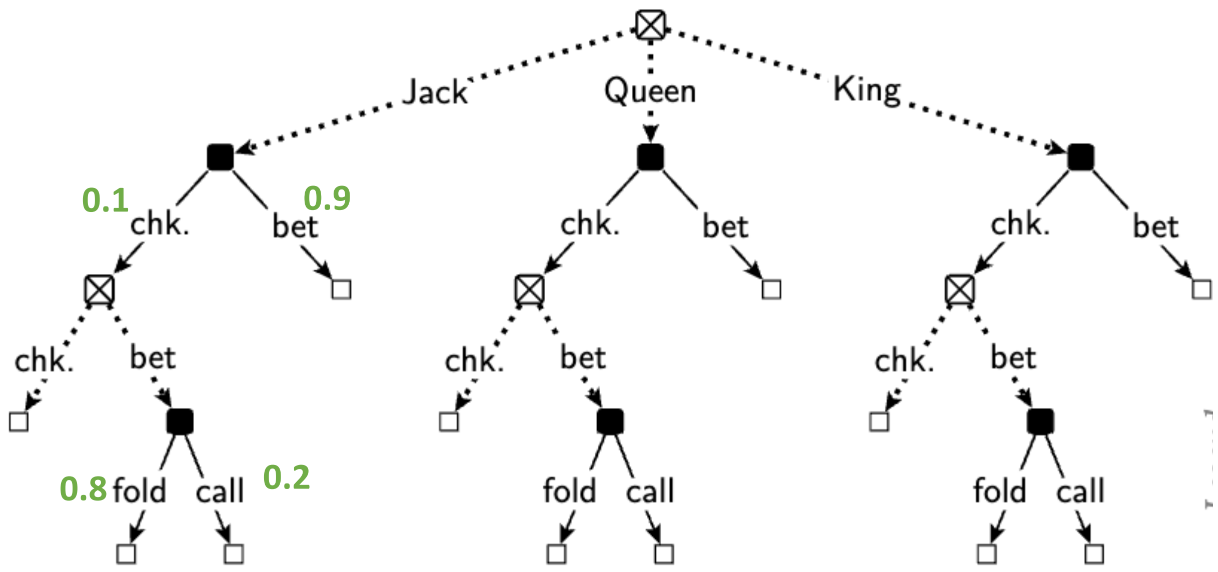
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



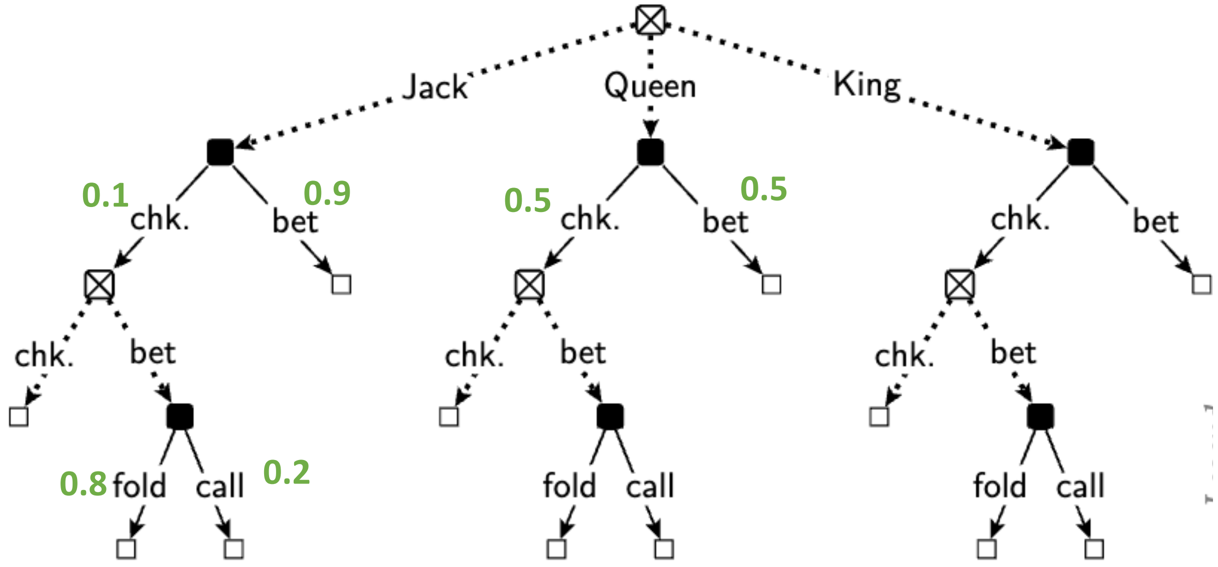
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



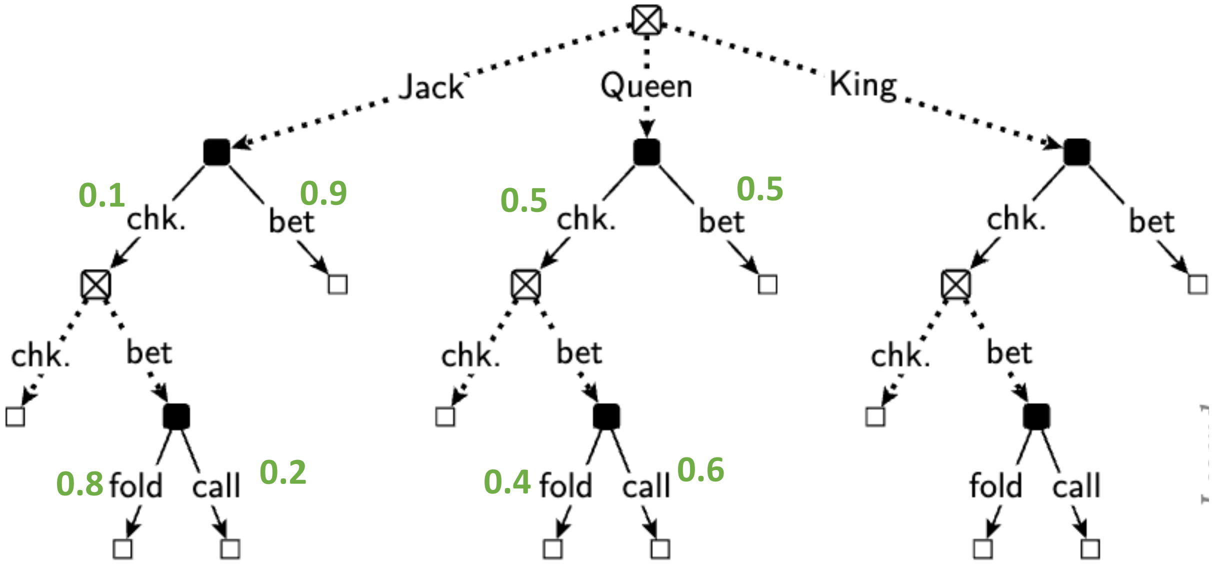
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



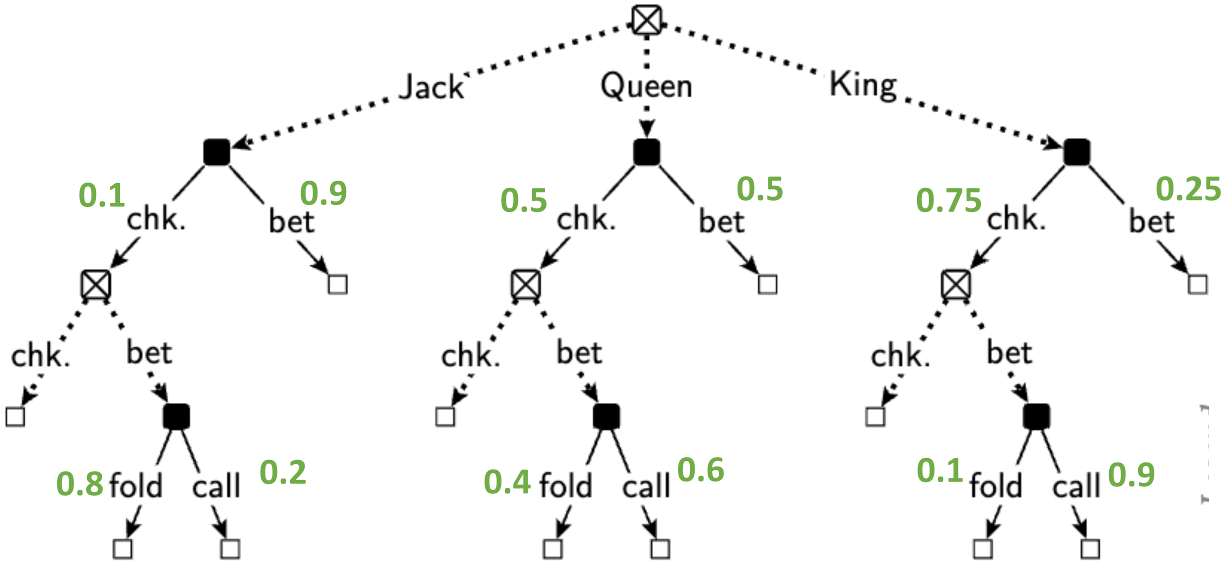
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



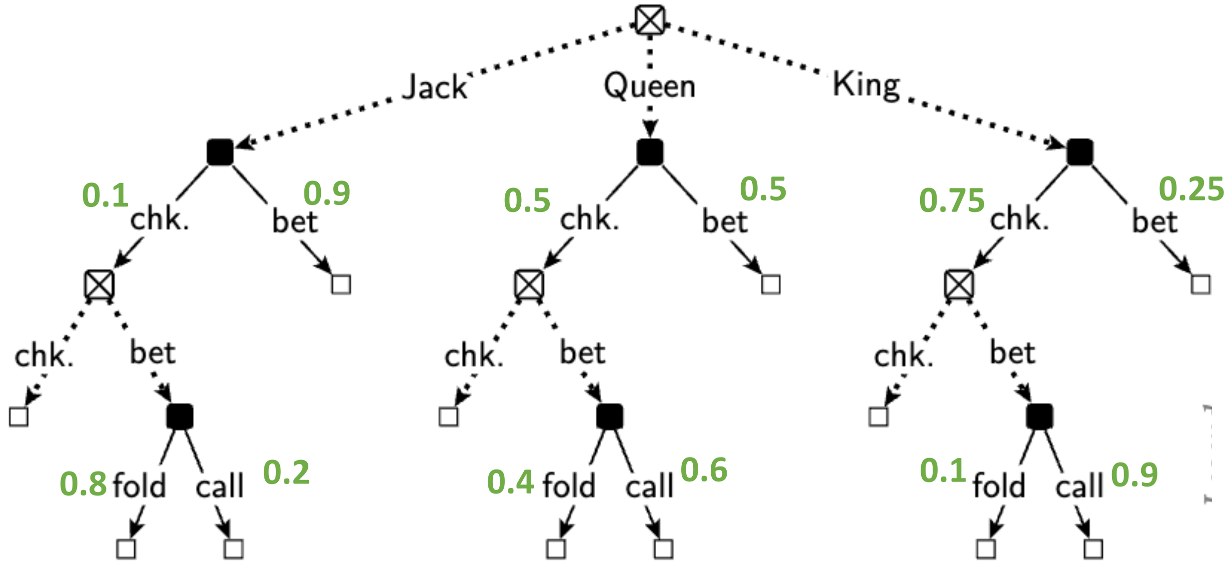
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



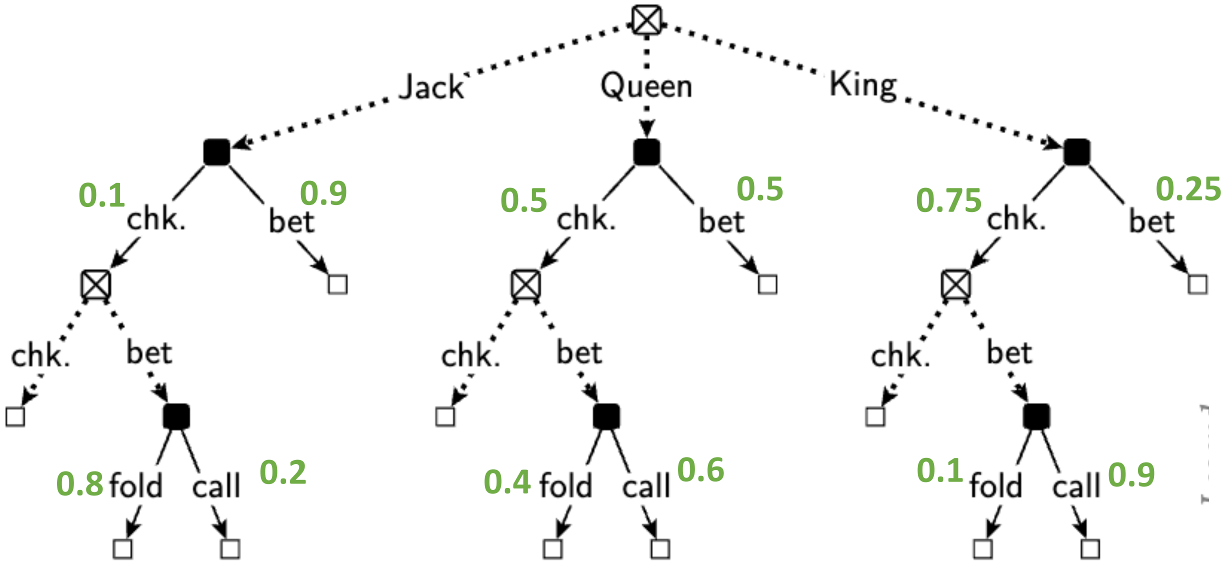
Behavioral strategies

Idea: Strategy = choice of distribution over available actions at each **decision point**



✔ Set of strategies is convex

Behavioral strategies



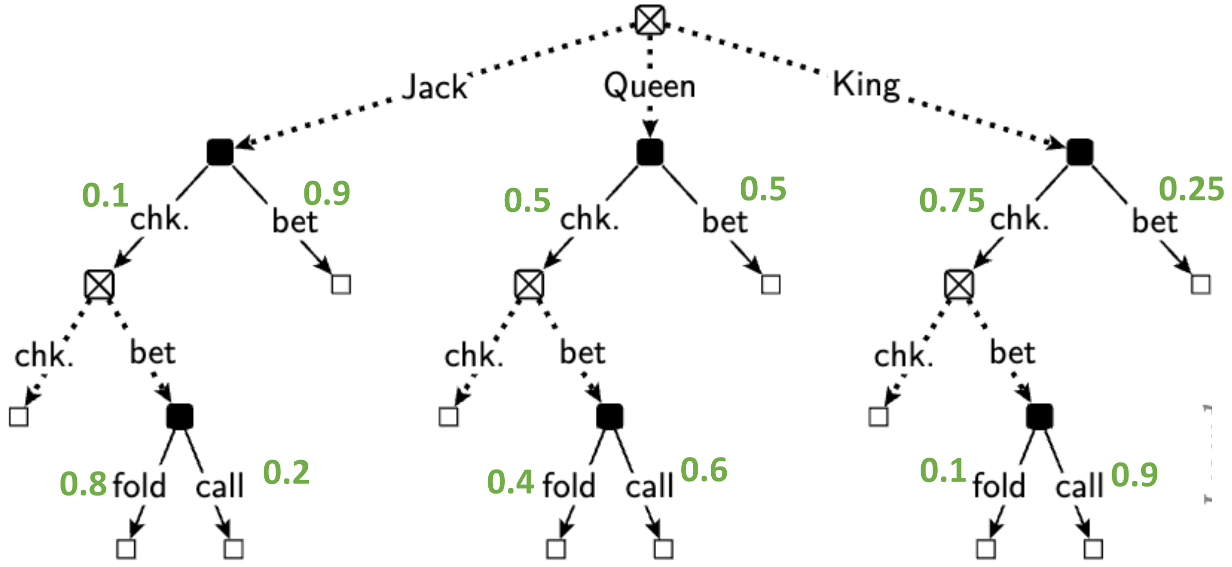
Idea: Strategy = choice of distribution over available actions at each **decision point**

✔ Set of strategies is convex

✘ Expected utility is **not** linear in this representation

Reason: prob. of reaching a terminal state is **product** of variables

Behavioral strategies



Idea: Strategy = choice of distribution over available actions at each **decision point**

✓ Set of strategies is convex

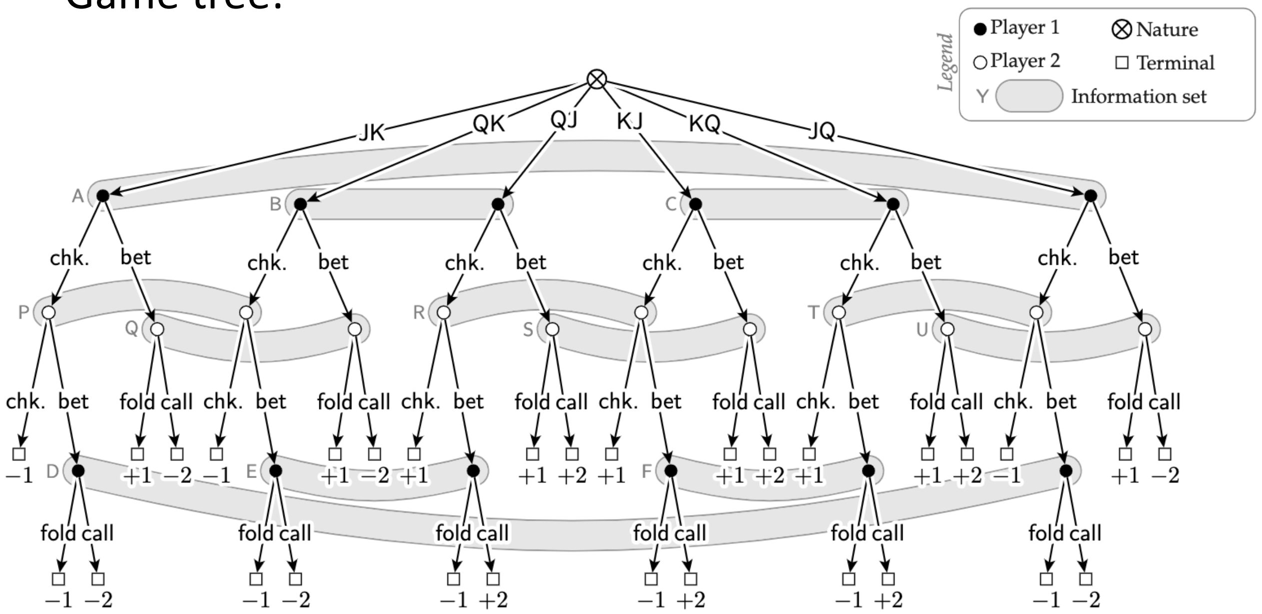
✗ Expected utility is **not** linear in this representation

Reason: prob. of reaching a terminal state is **product** of variables

Products = non-convexity 😪

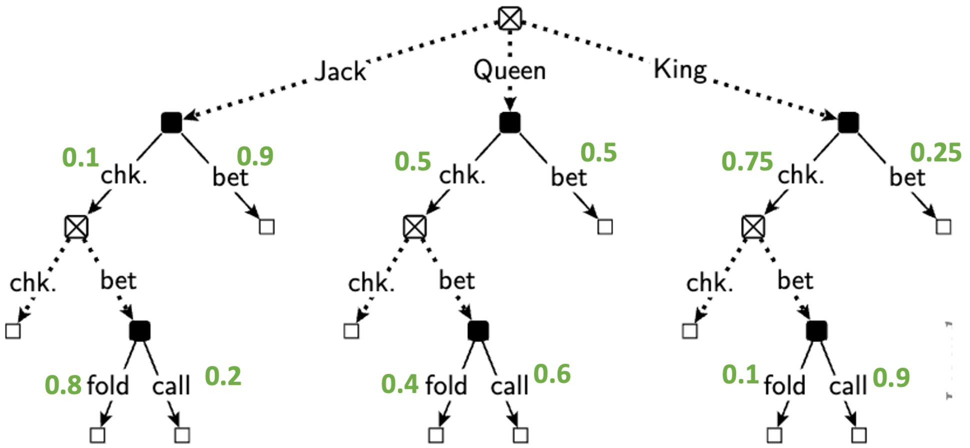
Expected Utility

Game tree:

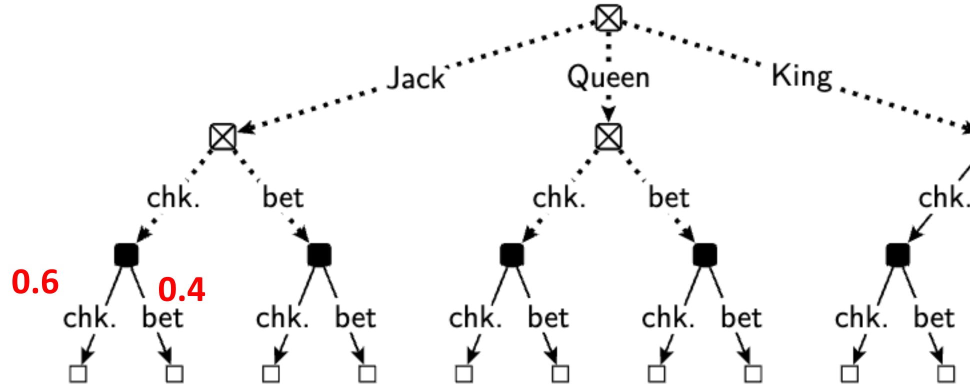


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

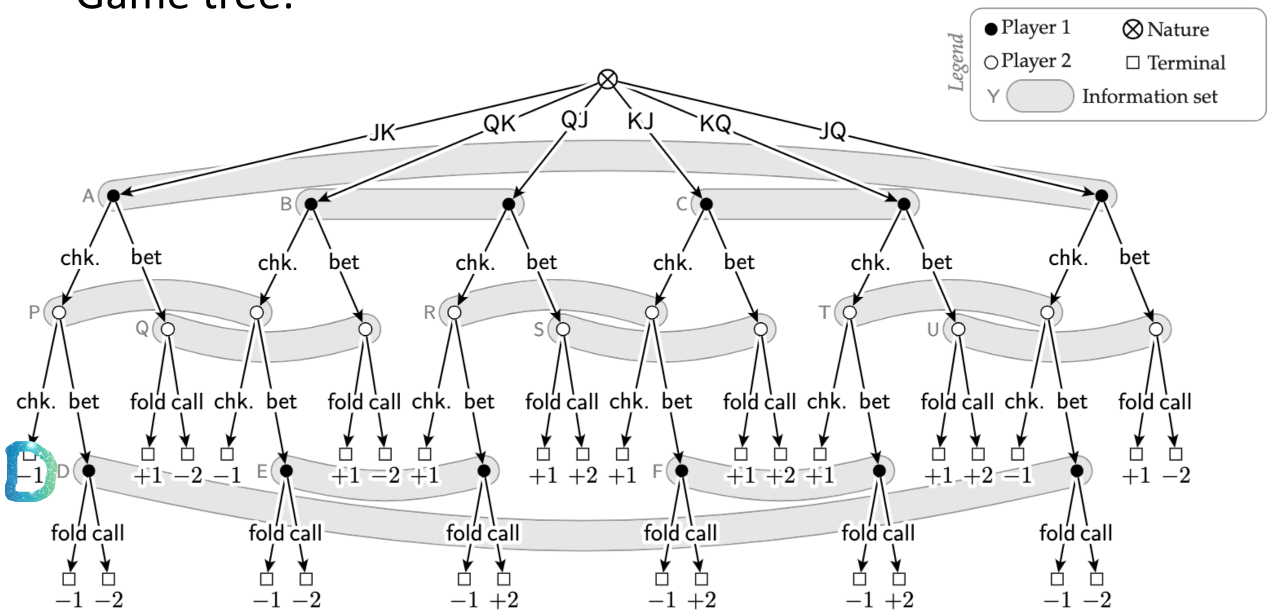


Decision problem and behavioral strategy of Player 2



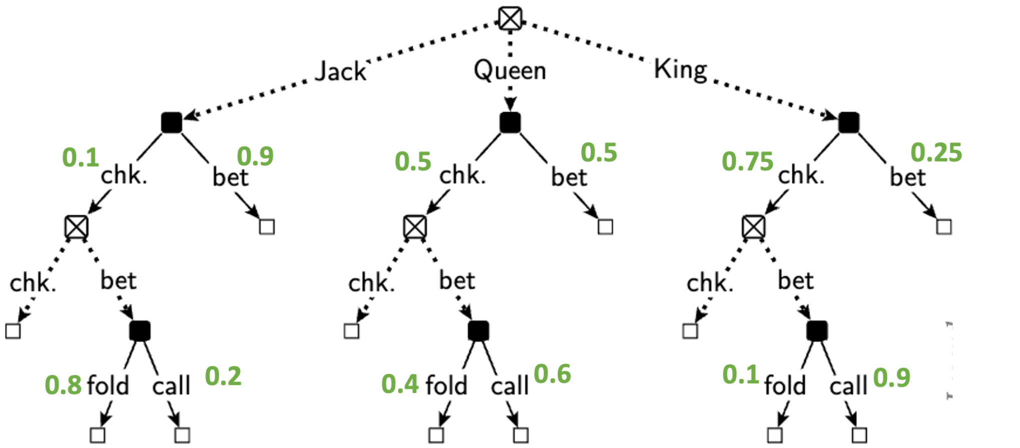
Expected Utility

Game tree:

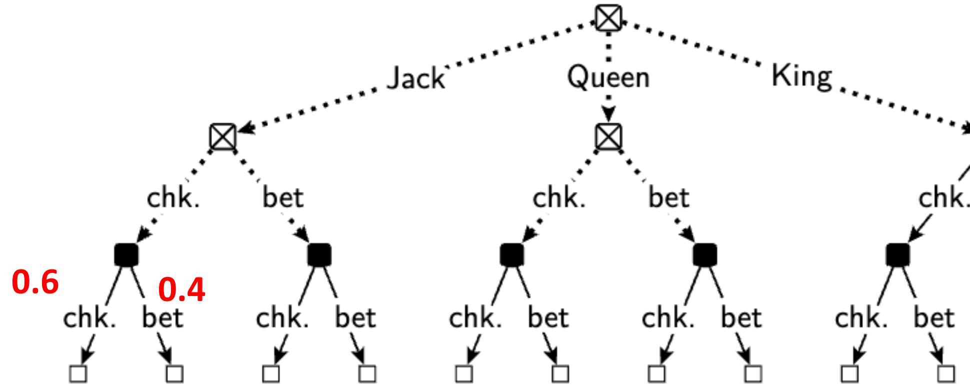


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

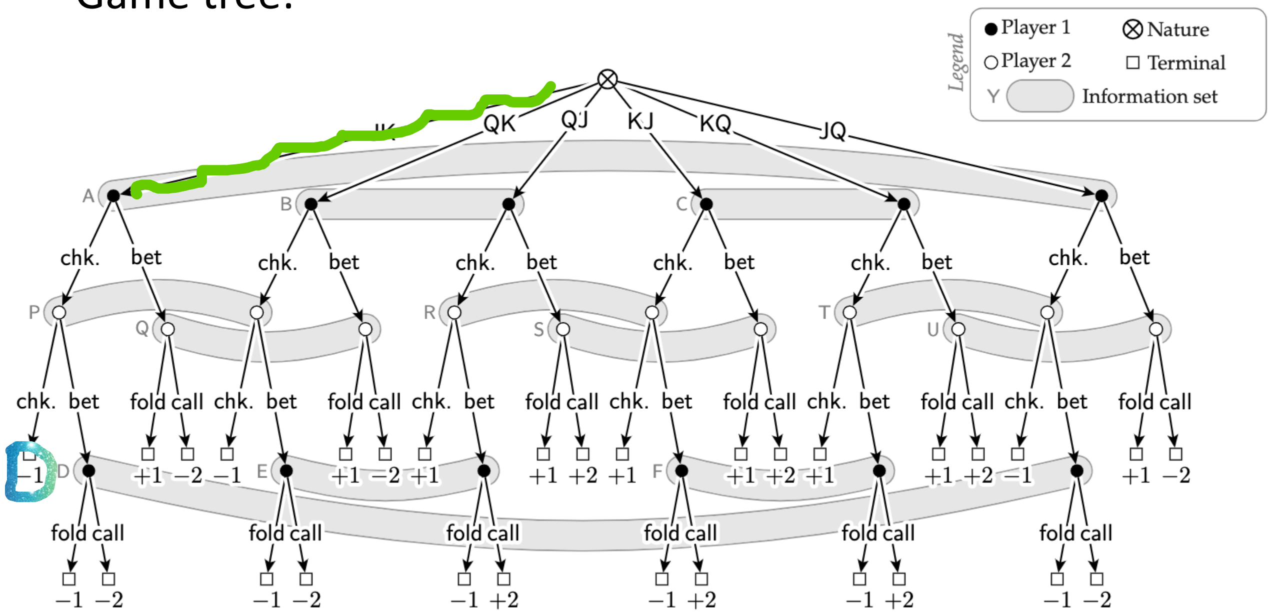


Decision problem and behavioral strategy of Player 2

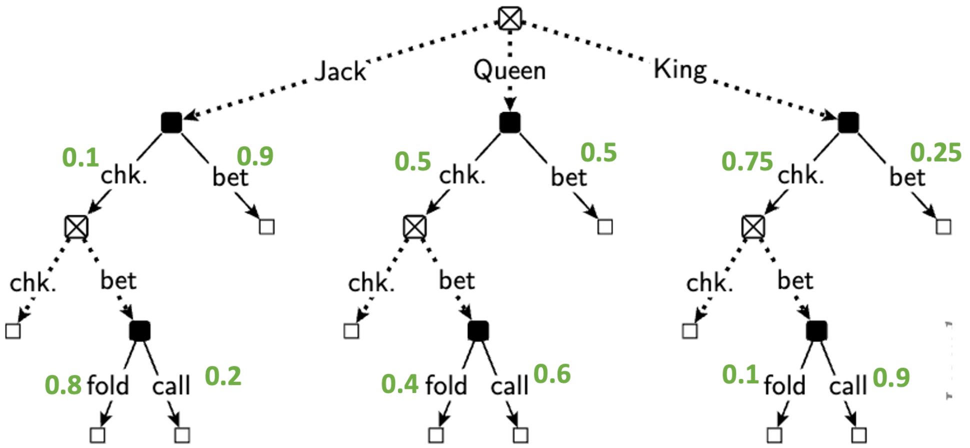


Expected Utility

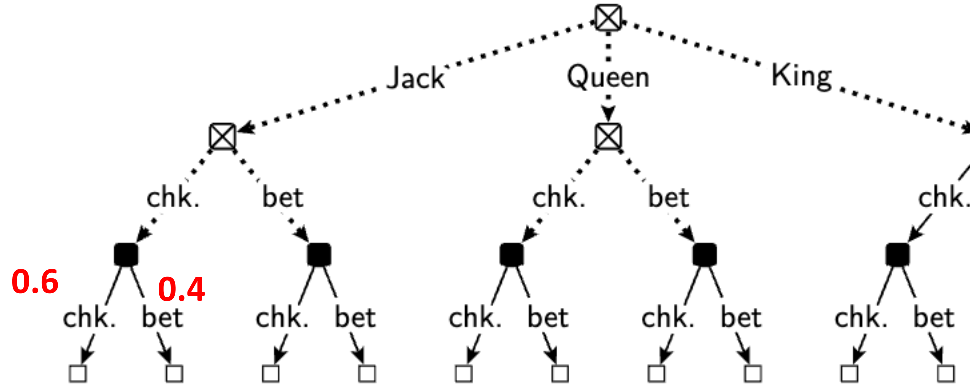
Game tree:



Decision problem and behavioral strategy of Player 1



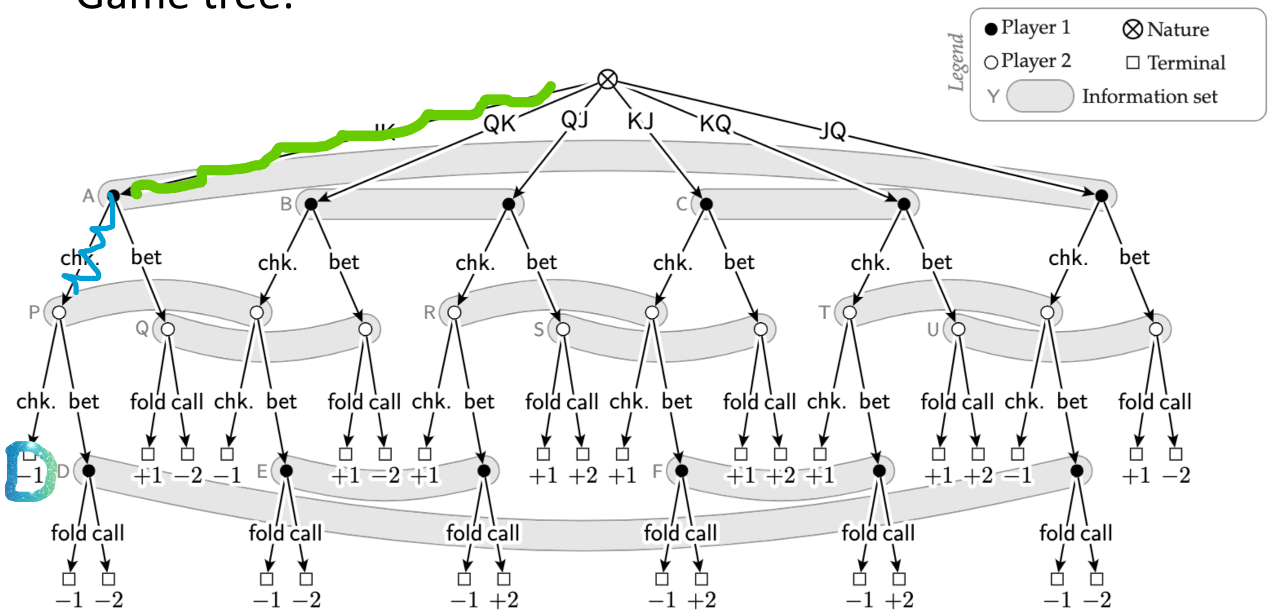
Decision problem and behavioral strategy of Player 2



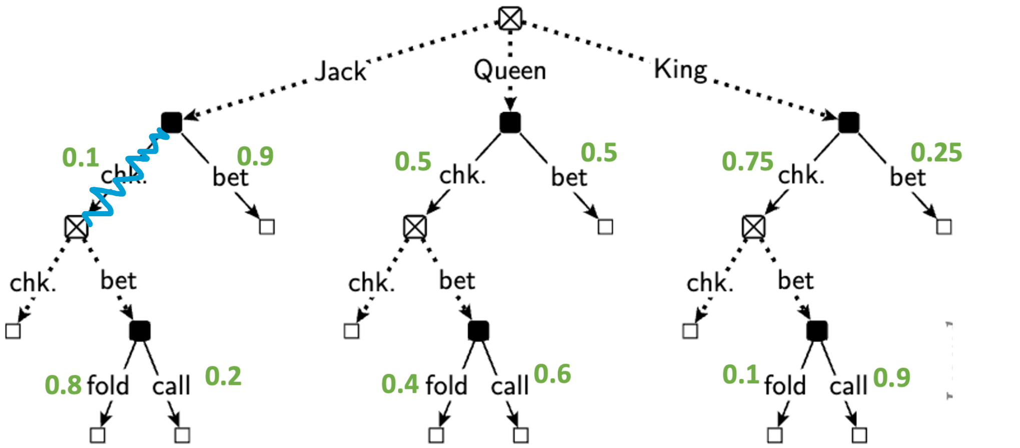
Prob of reaching this terminal state: 1/6 (Nature)

Expected Utility

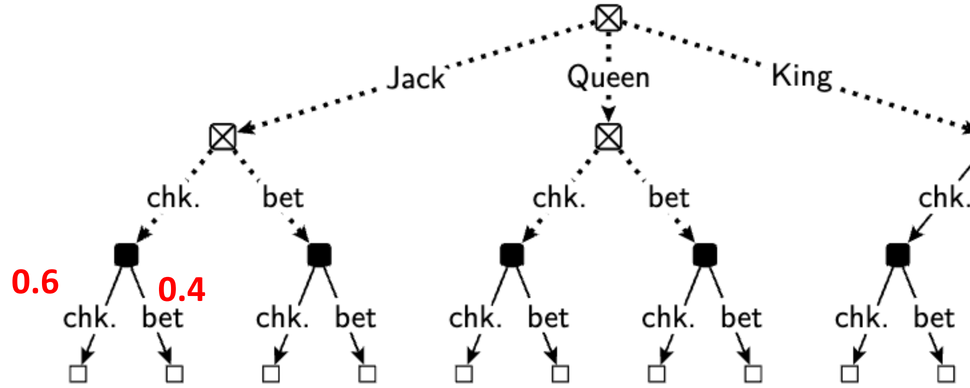
Game tree:



Decision problem and behavioral strategy of Player 1



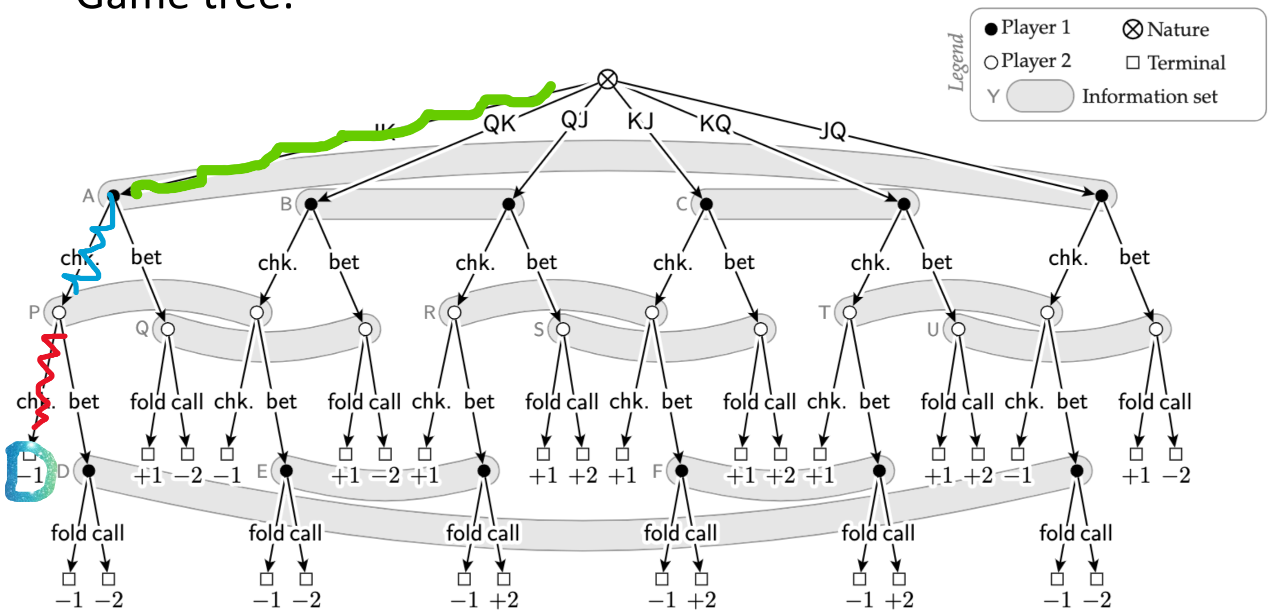
Decision problem and behavioral strategy of Player 2



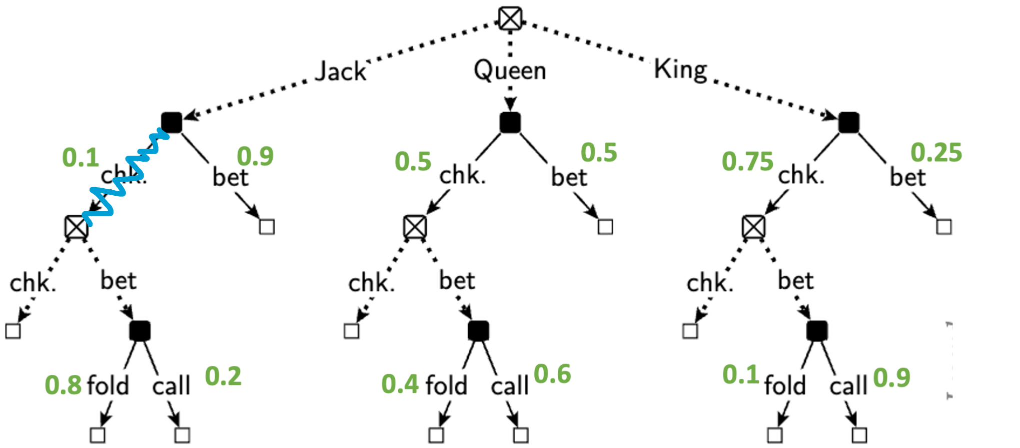
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1)

Expected Utility

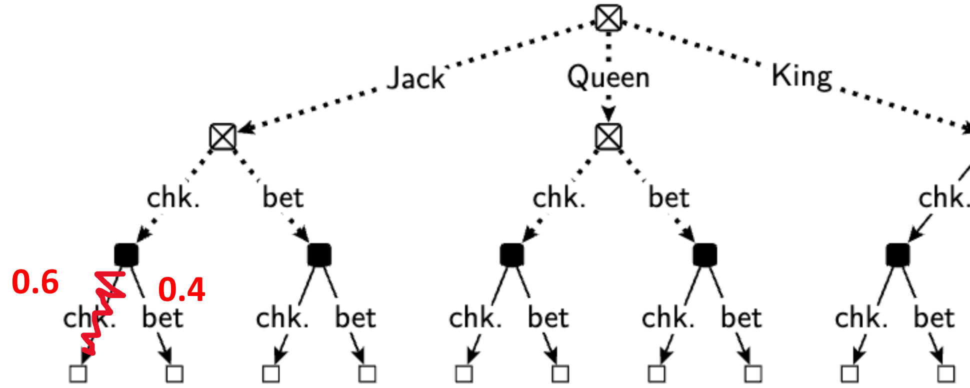
Game tree:



Decision problem and behavioral strategy of Player 1



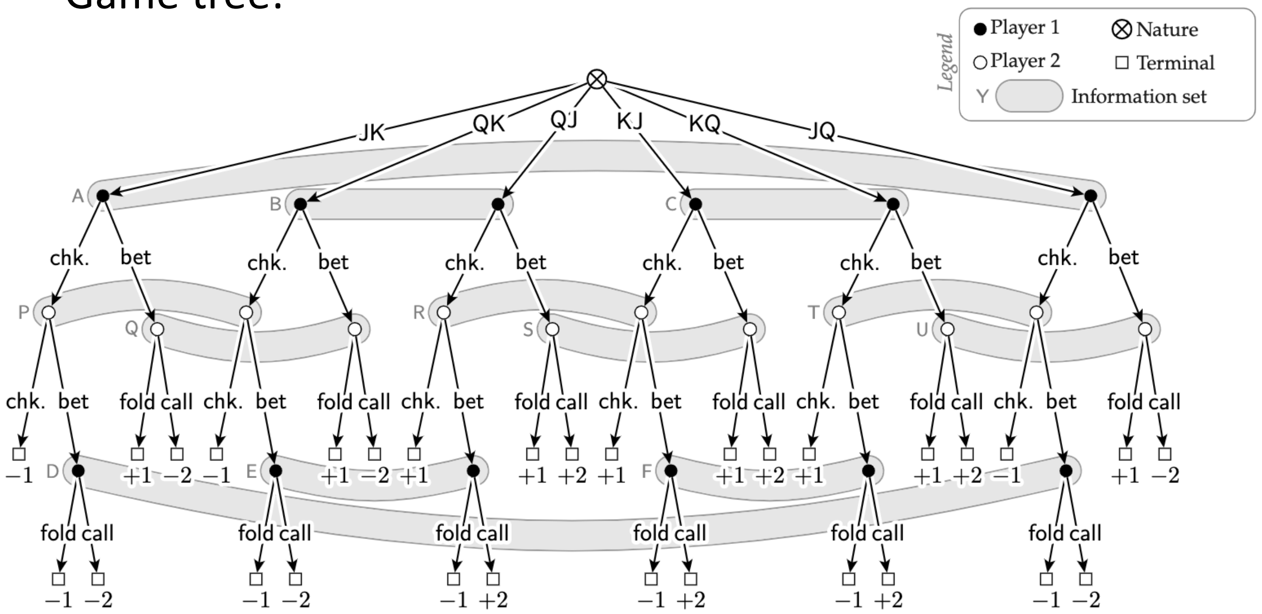
Decision problem and behavioral strategy of Player 2



Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.6 (Pl2)

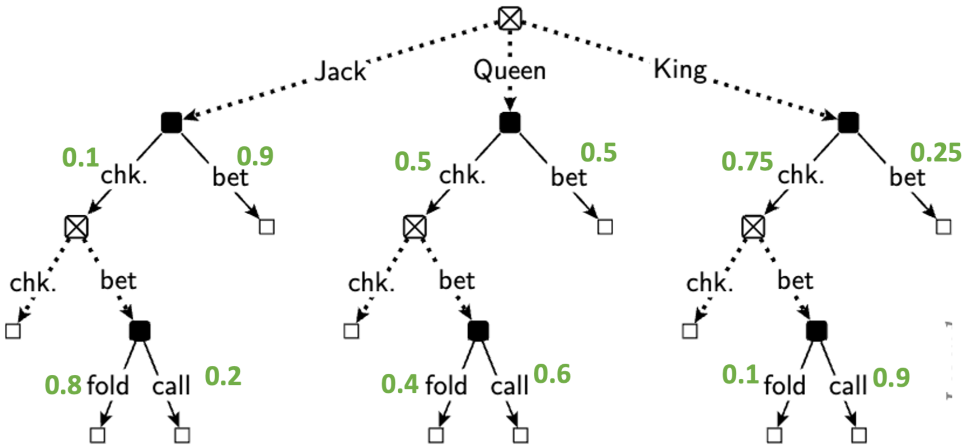
Expected Utility

Game tree:

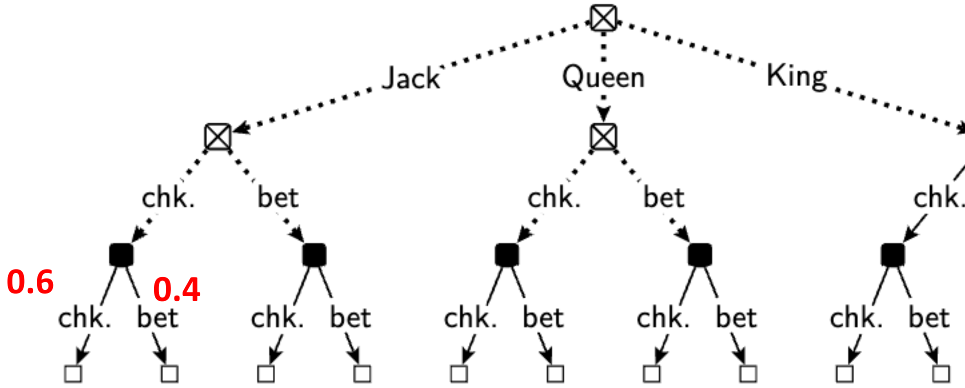


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

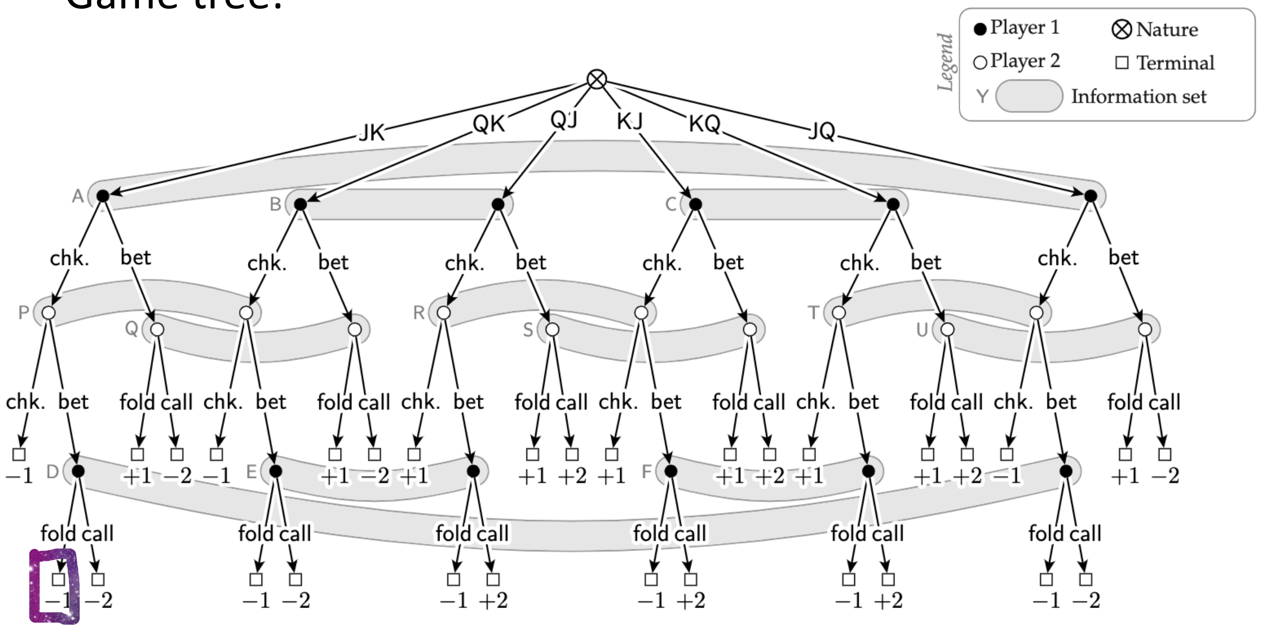


Decision problem and behavioral strategy of Player 2



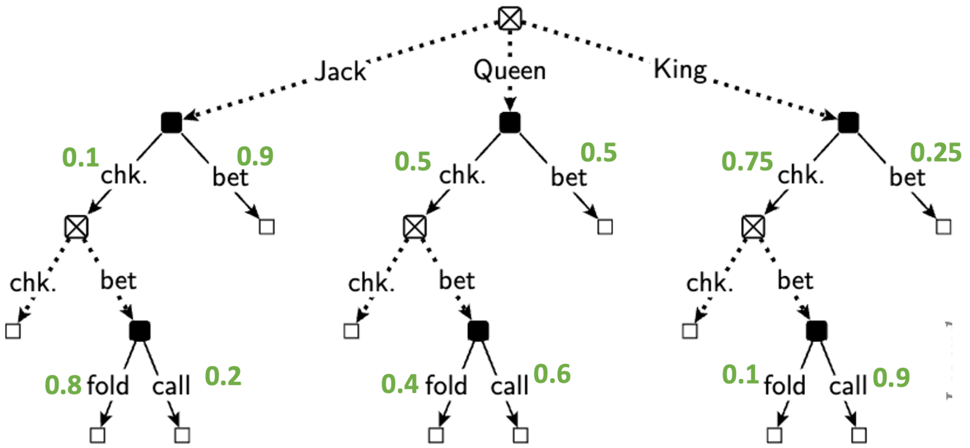
Expected Utility

Game tree:

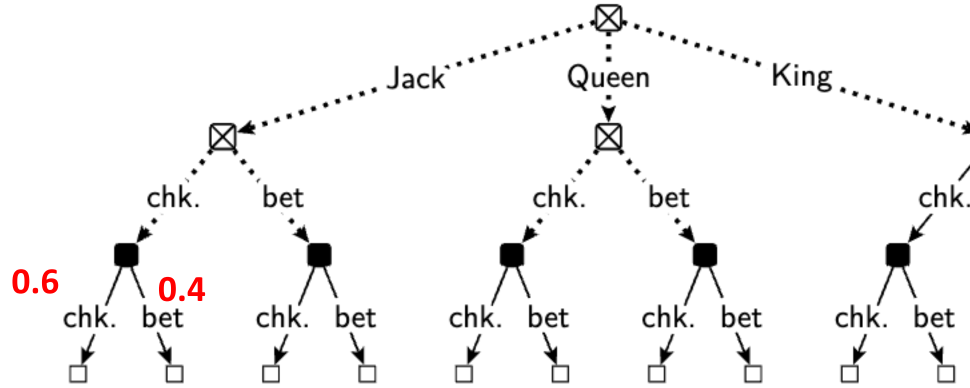


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

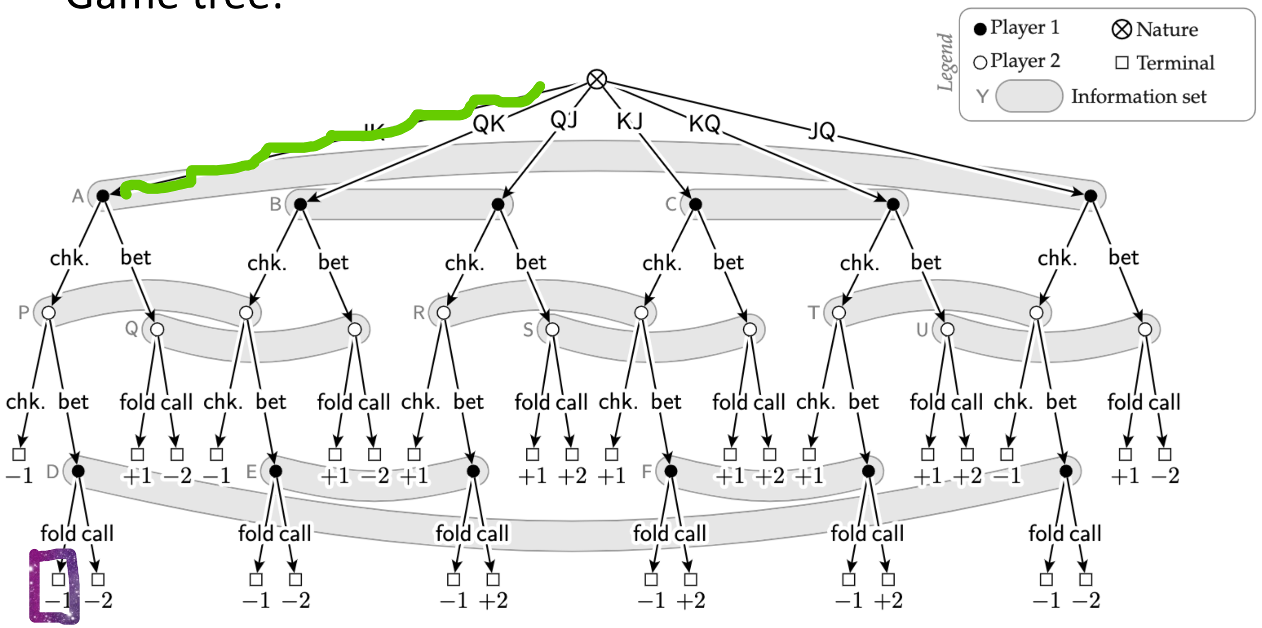


Decision problem and behavioral strategy of Player 2



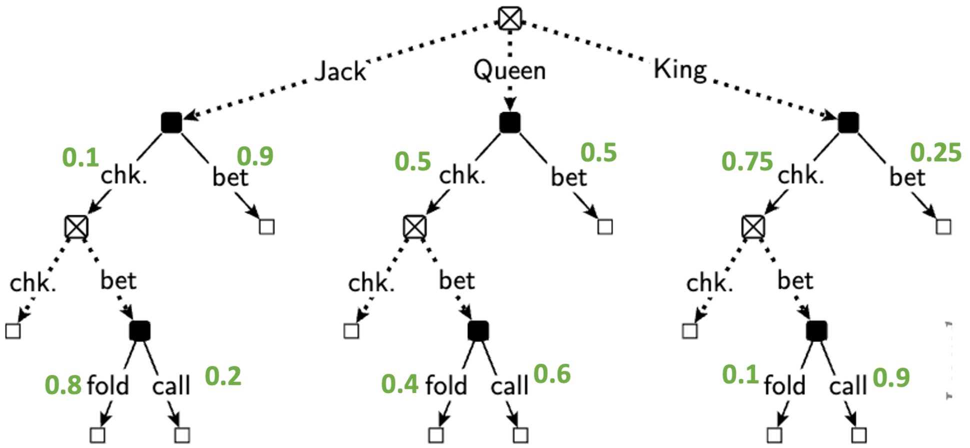
Expected Utility

Game tree:

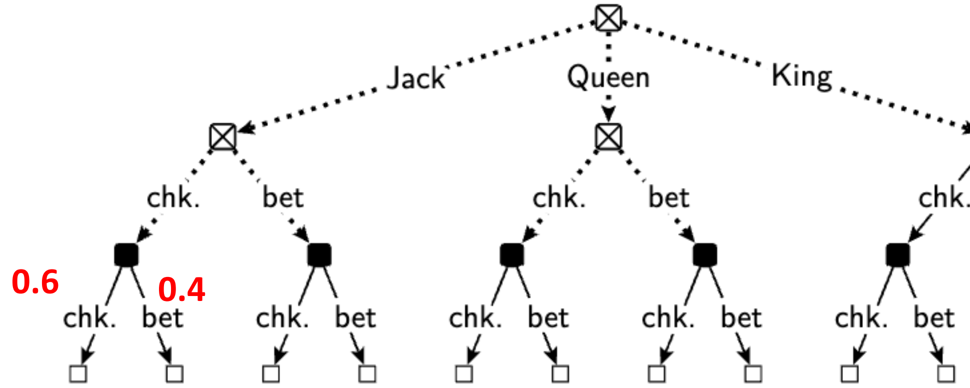


Prob of reaching this terminal state: 1/6 (Nature)

Decision problem and behavioral strategy of Player 1

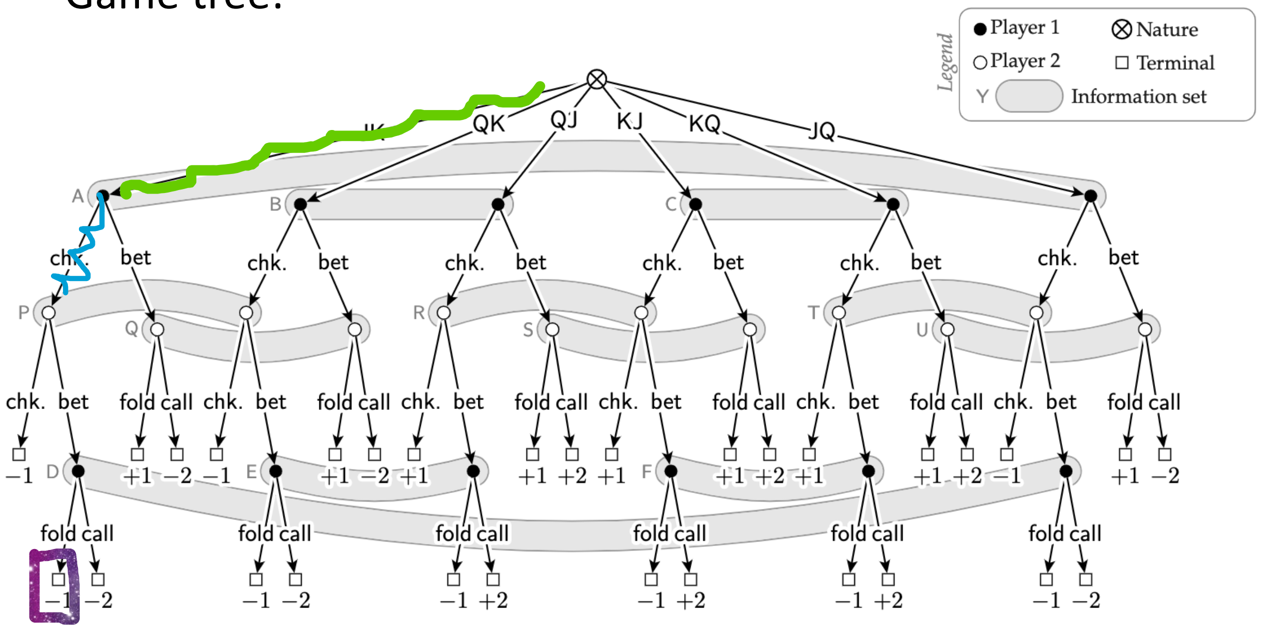


Decision problem and behavioral strategy of Player 2

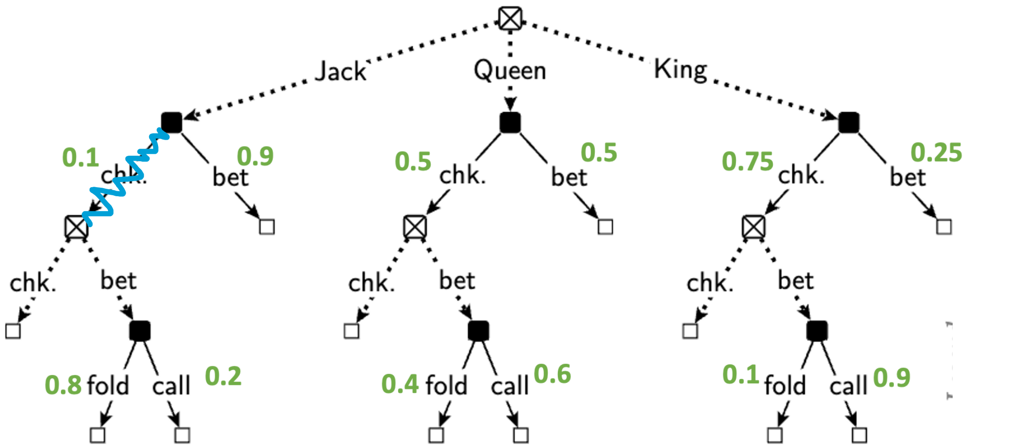


Expected Utility

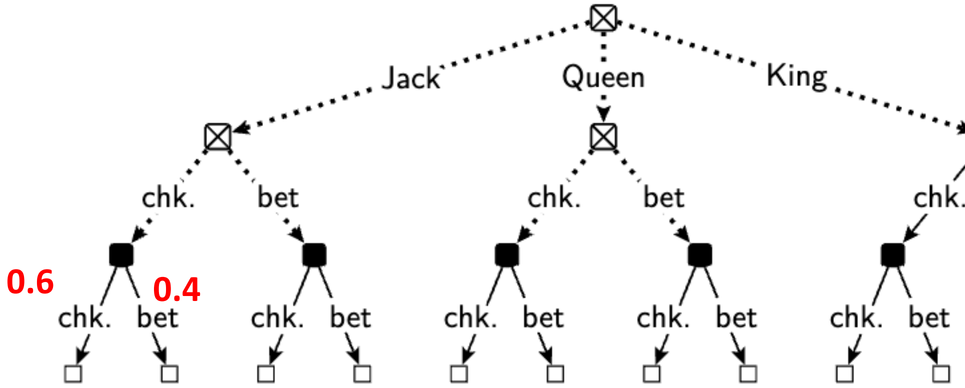
Game tree:



Decision problem and behavioral strategy of Player 1



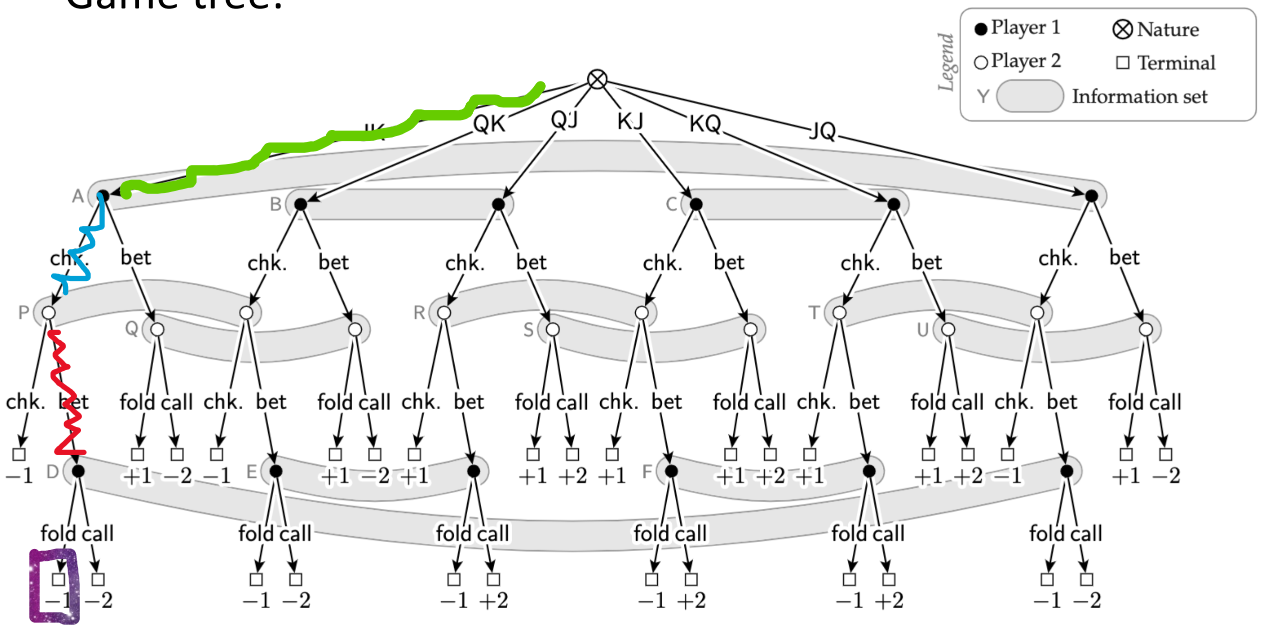
Decision problem and behavioral strategy of Player 2



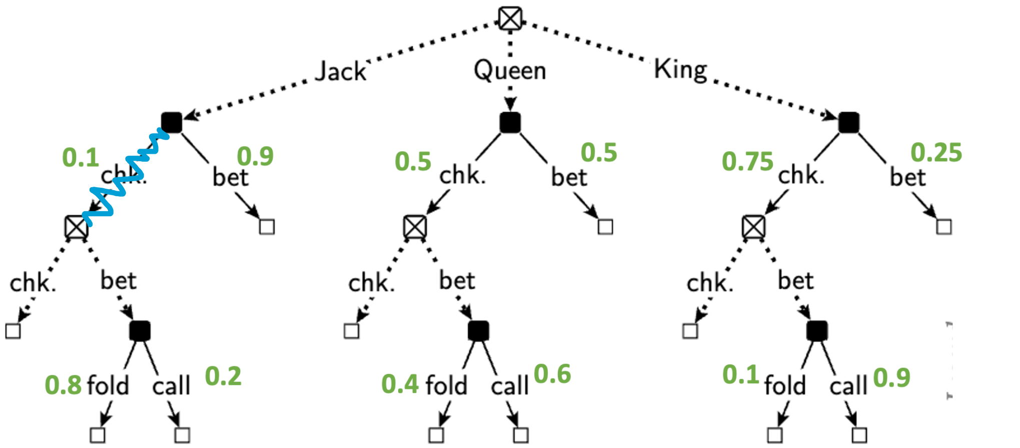
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1)

Expected Utility

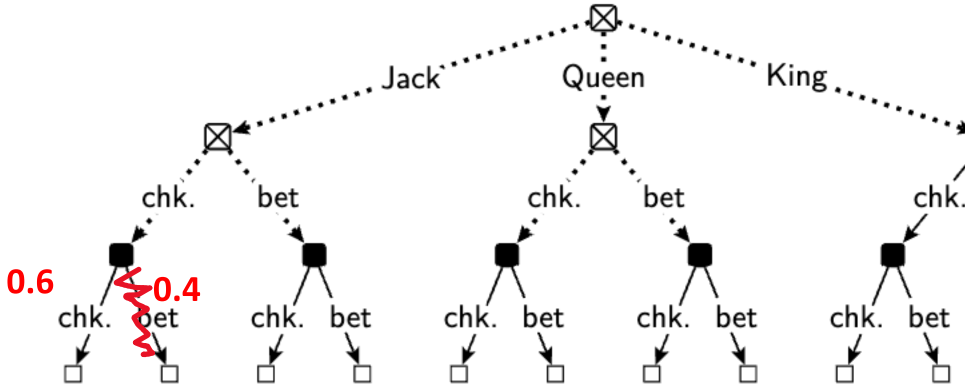
Game tree:



Decision problem and behavioral strategy of Player 1



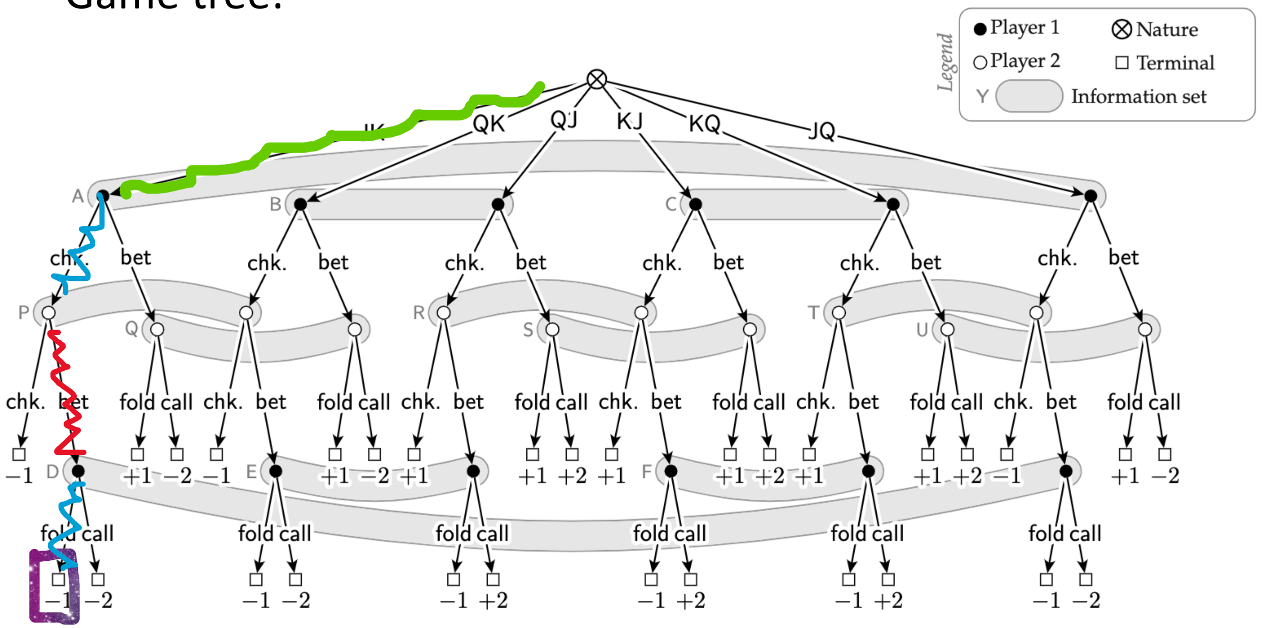
Decision problem and behavioral strategy of Player 2



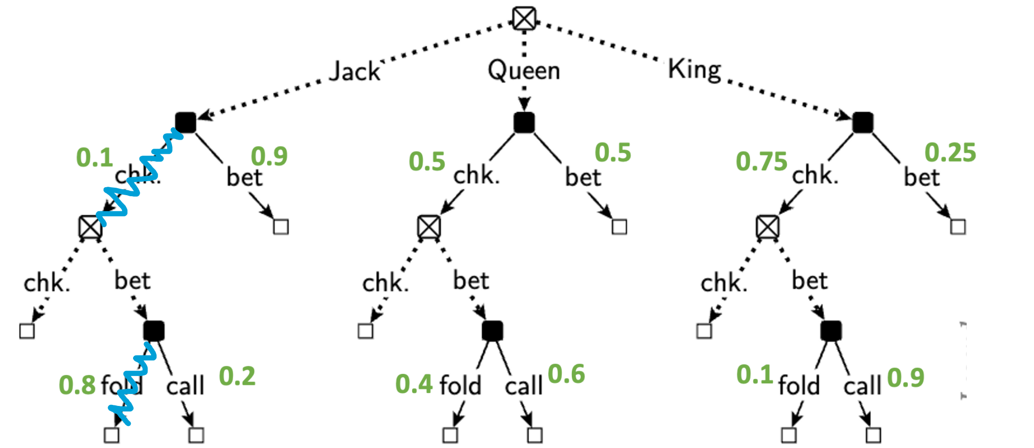
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.4 (Pl2)

Expected Utility

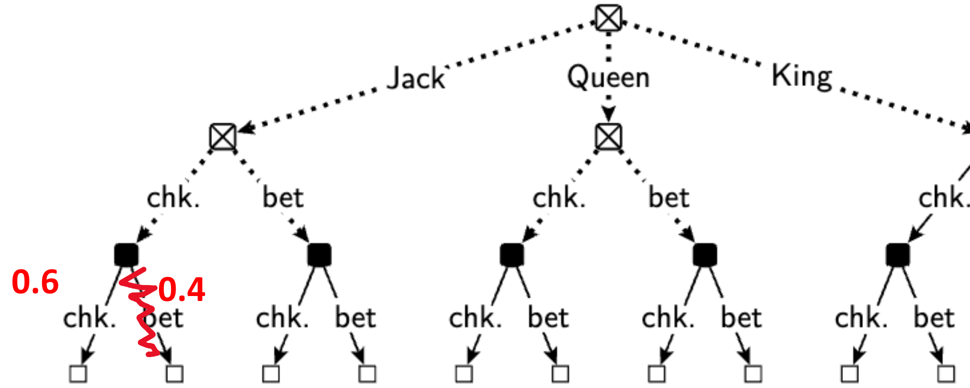
Game tree:



Decision problem and behavioral strategy of Player 1



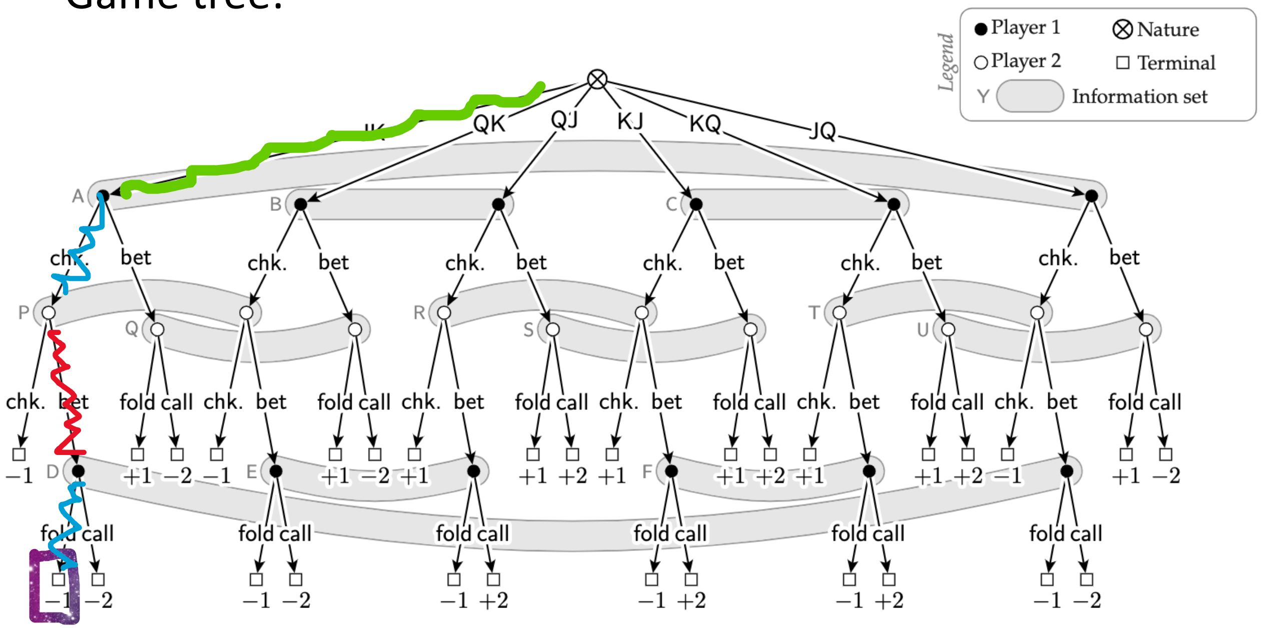
Decision problem and behavioral strategy of Player 2



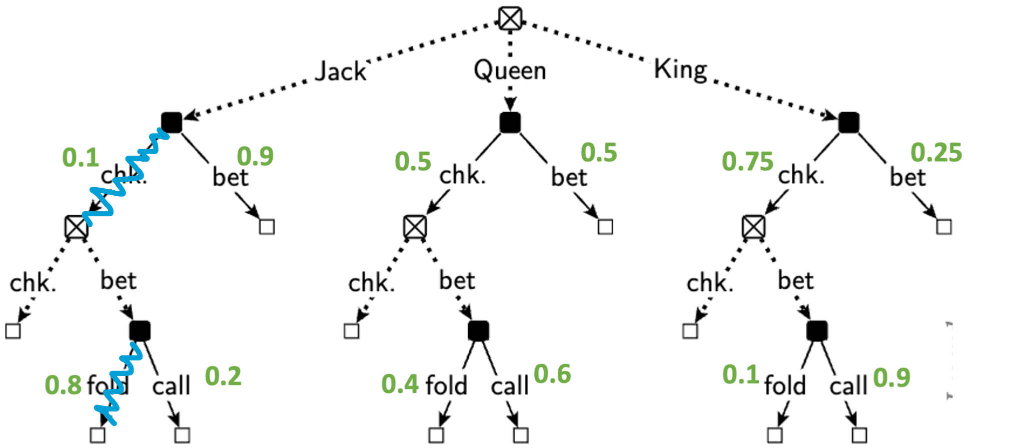
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.4 (Pl2) x 0.8 (Pl1)

Expected Utility

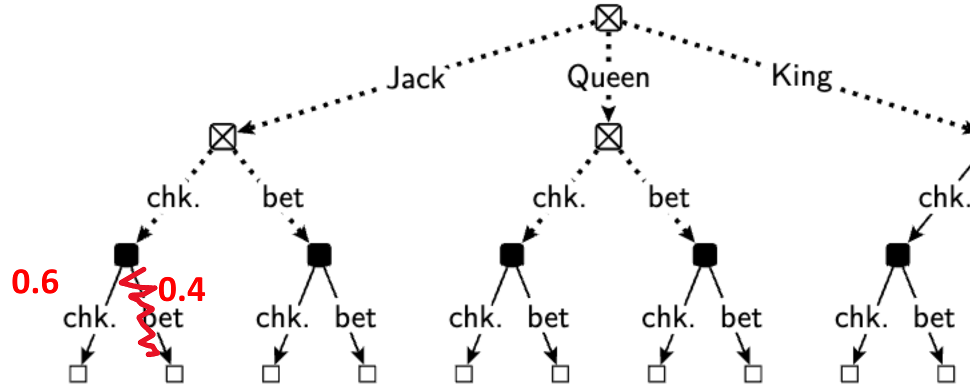
Game tree:



Decision problem and behavioral strategy of Player 1



Decision problem and behavioral strategy of Player 2



Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.4 (Pl2)
x 0.8 (Pl1)

When these are variables being optimized, we have a product! Non-convexity in player's strategy

Kuhn's Theorem

(Under perfect recall assumption)

Normal-form strategies and behavioral strategies are equally powerful

(more formally: they can induce the same distribution over terminal states)

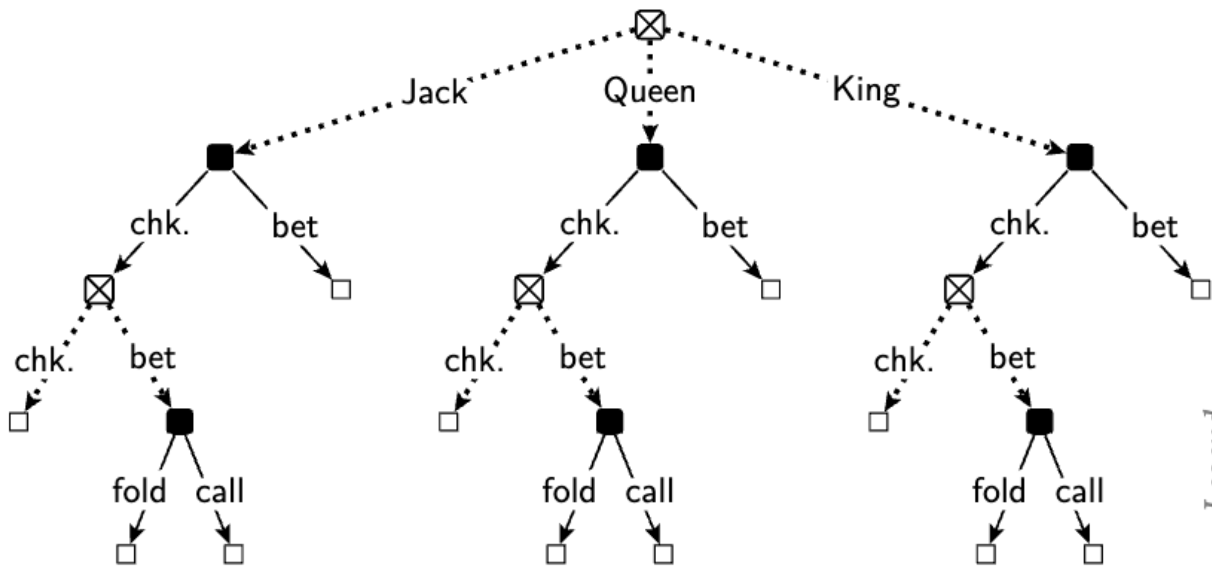
🚨 **Danger zone™:** the theorem is not true anymore if the player does not have perfect recall!

Recap on Behavioral Strategies

	Idea	Obvious downsides	Good news
(Reduced) Normal-form strategies	Distribution over deterministic strategies $\mu \in \Delta(\Pi)$	Exponentially-sized object	In rare cases, it's possible to operate implicitly on the exponential object via a kernel trick
Behavioral strategies	**Local distribution over actions at each decision point** $\boldsymbol{b} \in \times_j \boldsymbol{\Delta}(\boldsymbol{A}_j)$	**Expected utility is nonconvex in the the entries of vector $\boldsymbol{b}$**	**Kuhn's theorem: same power as reduced normal-form strategies**

“Fixing” Behavioral Strategies: Sequence-Form Strategies

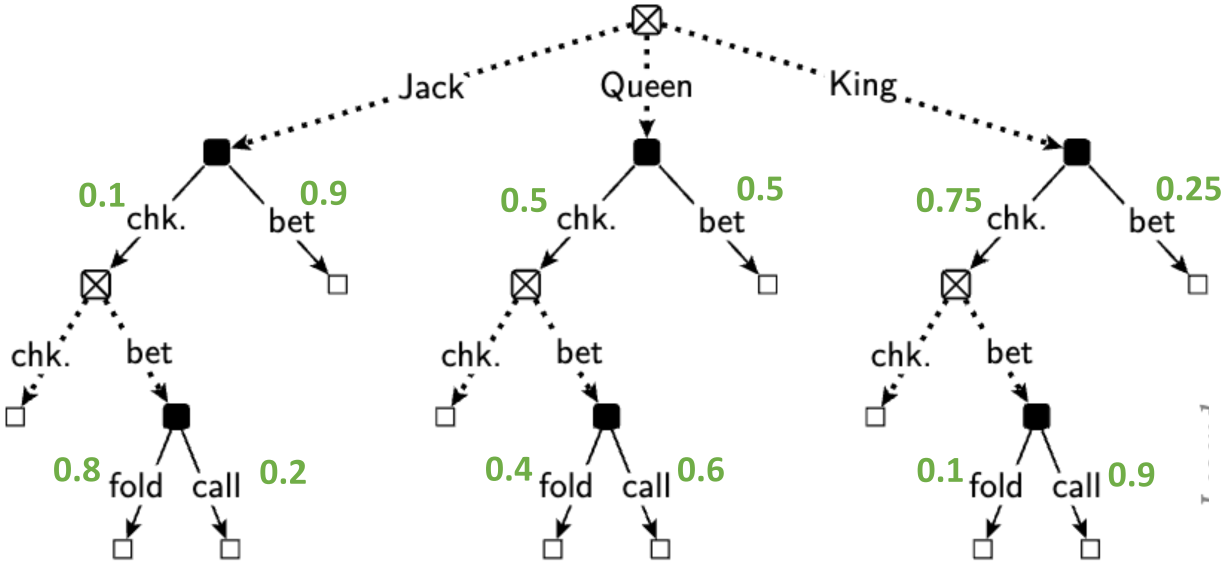
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

“Fixing” Behavioral Strategies: Sequence-Form Strategies

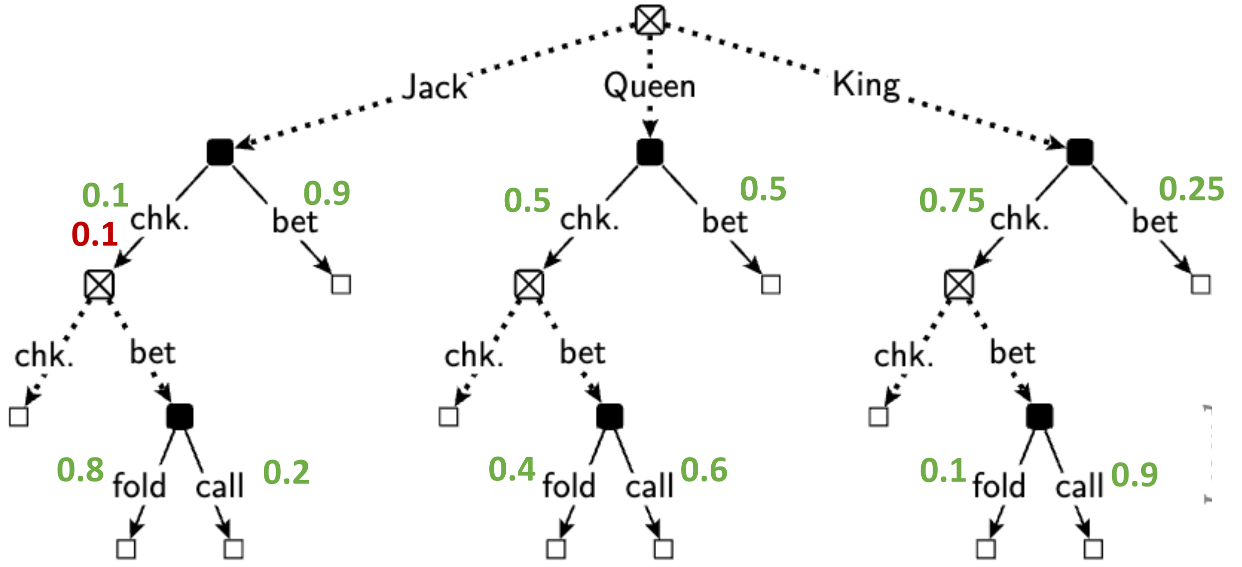
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

“Fixing” Behavioral Strategies: Sequence-Form Strategies

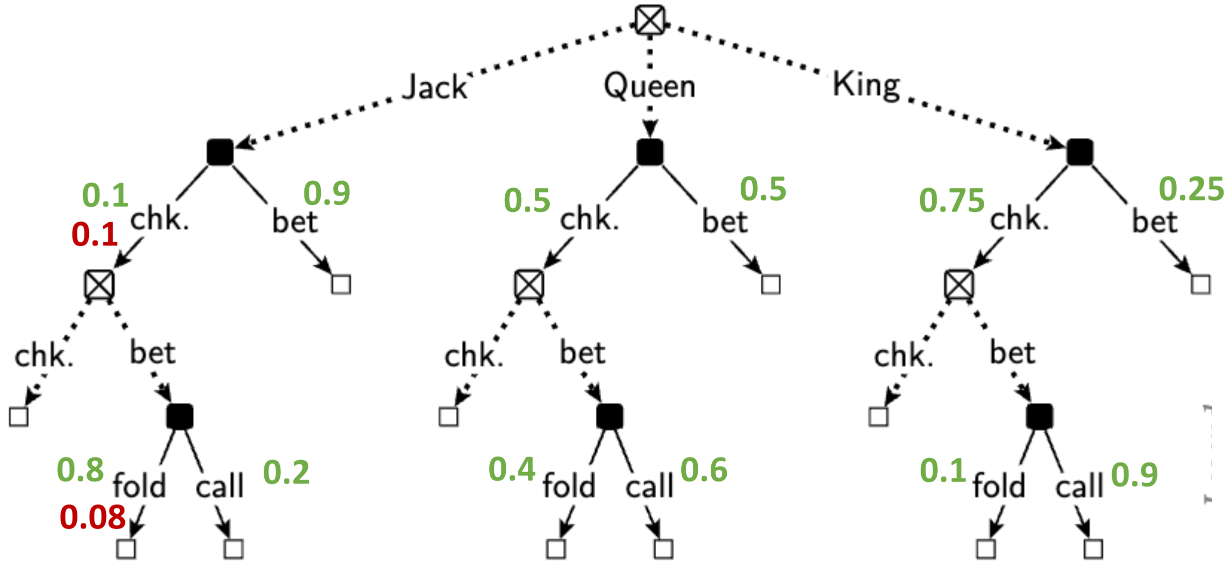
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

"Fixing" Behavioral Strategies: Sequence-Form Strategies

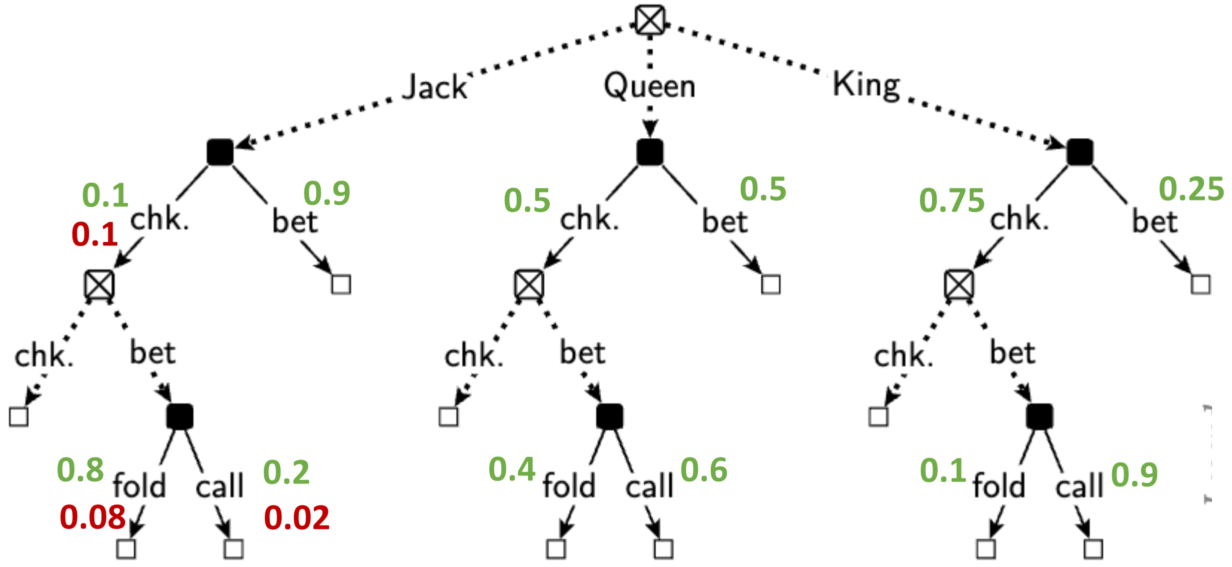
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

“Fixing” Behavioral Strategies: Sequence-Form Strategies

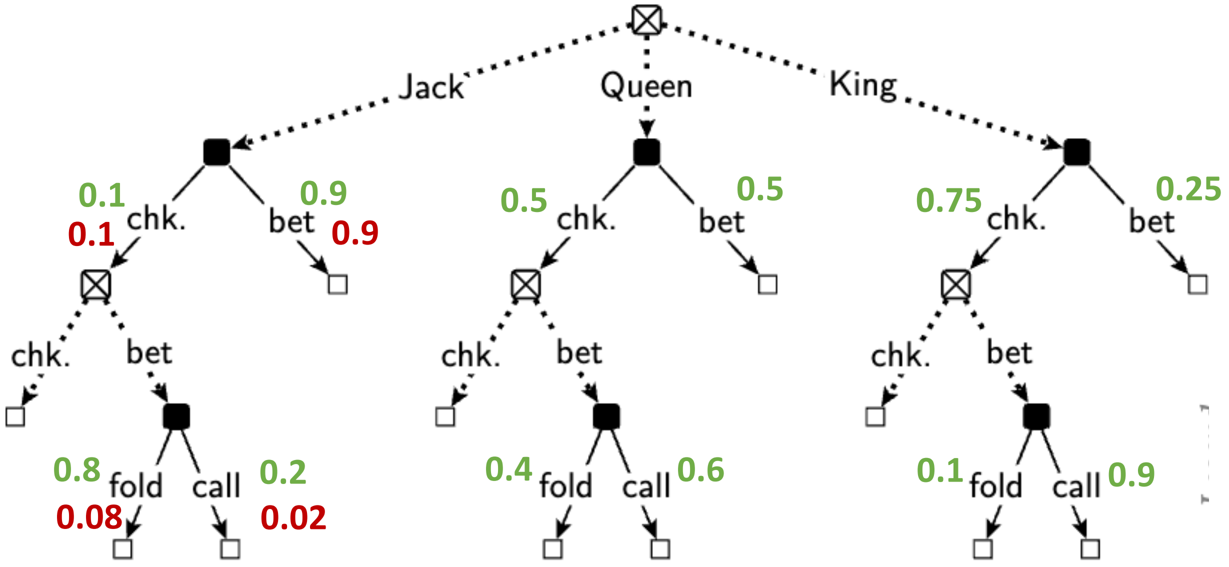
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

“Fixing” Behavioral Strategies: Sequence-Form Strategies

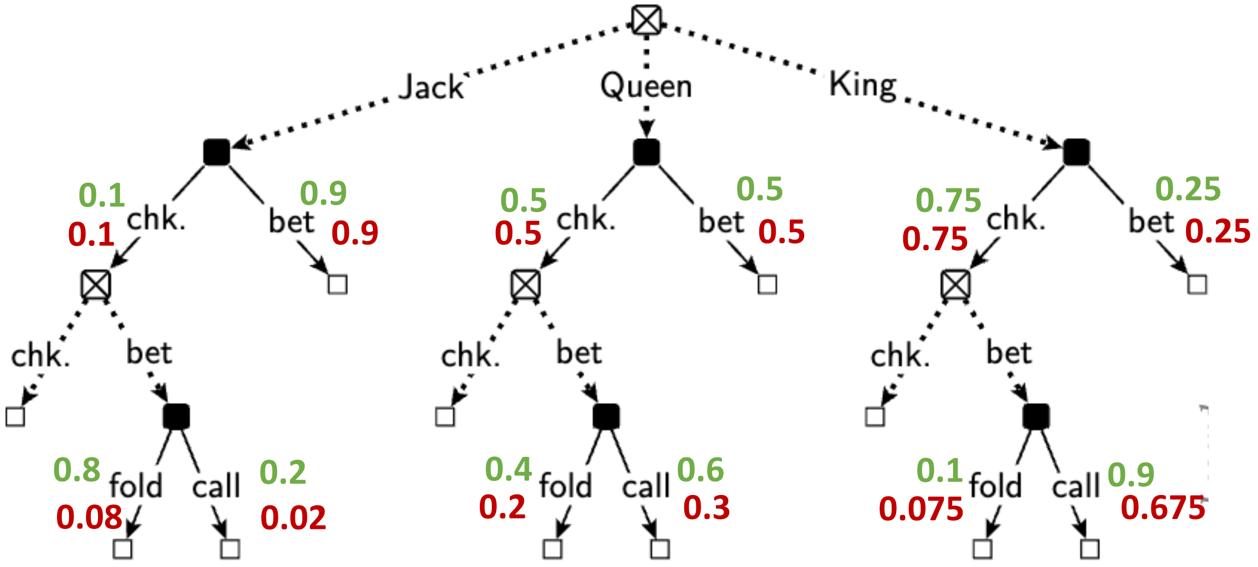
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

“Fixing” Behavioral Strategies: Sequence-Form Strategies

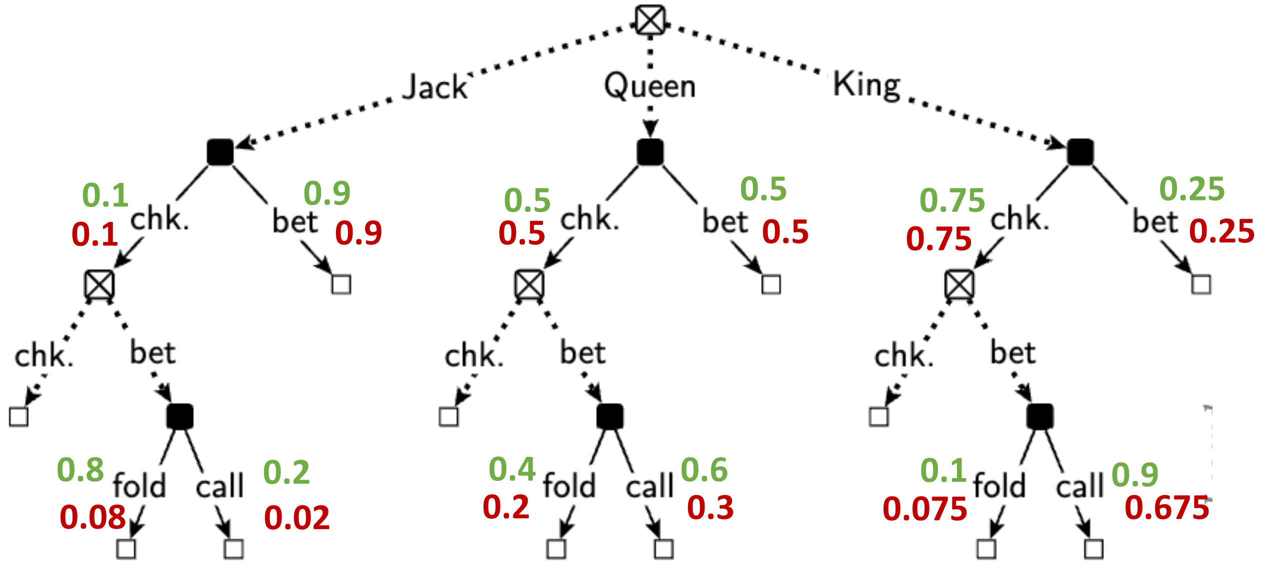
Idea: Store probability for whole sequences of actions



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

"Fixing" Behavioral Strategies: Sequence-Form Strategies

Idea: Store probability for whole sequences of actions

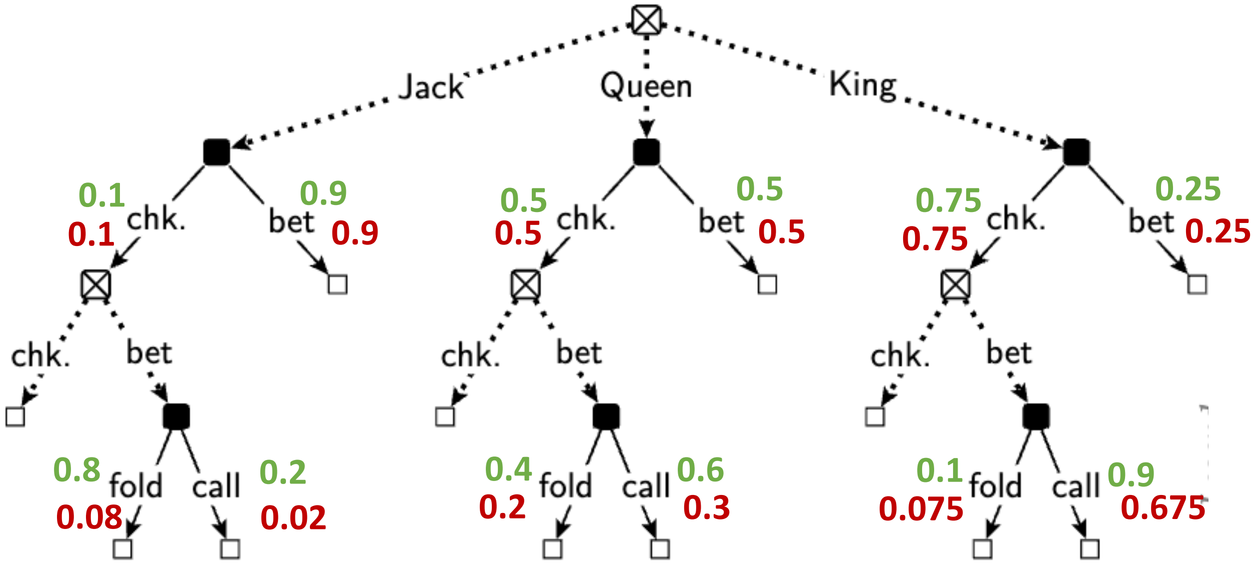


⭐ **Consistency constraints**

Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

"Fixing" Behavioral Strategies: Sequence-Form Strategies

Idea: Store probability for whole sequences of actions



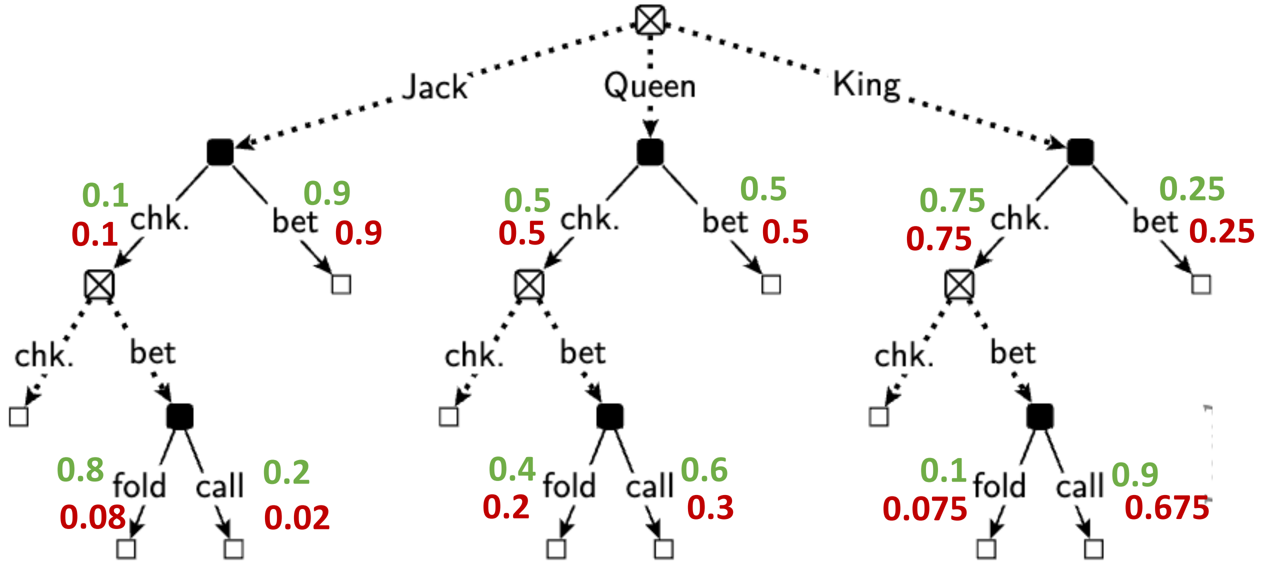
Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

⭐ **Consistency constraints**

1. Entries all non-negative

“Fixing” Behavioral Strategies: Sequence-Form Strategies

Idea: Store probability for whole sequences of actions



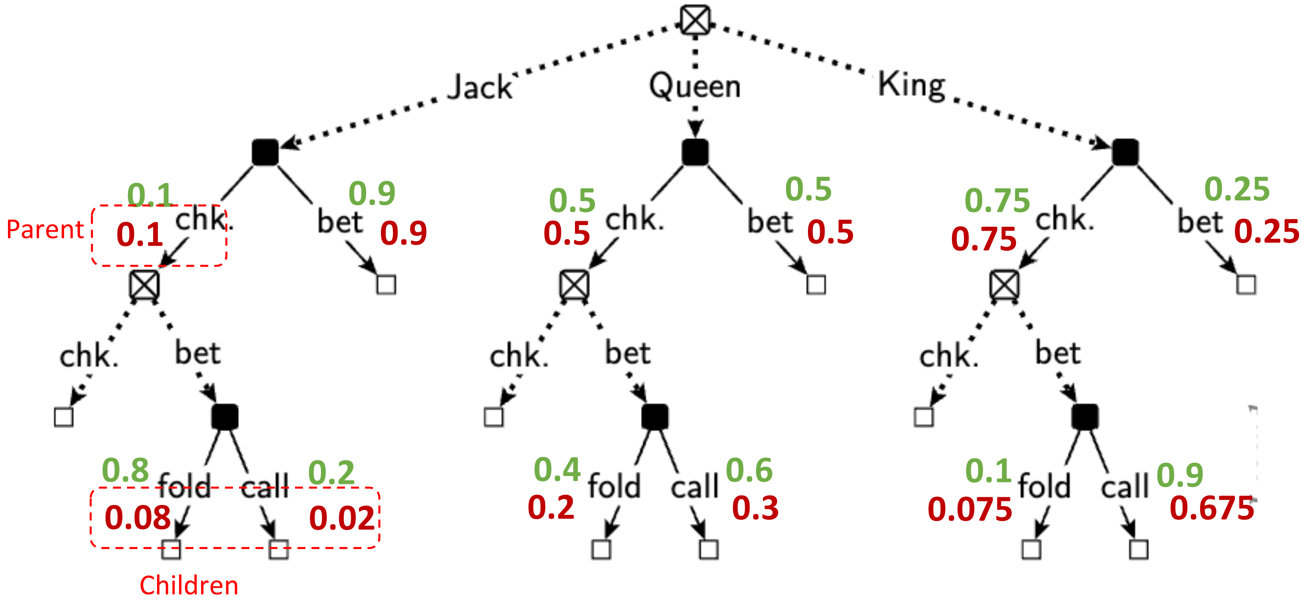
Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

⭐ **Consistency constraints**

1. Entries all non-negative
2. Root sequence has probability 1.0

“Fixing” Behavioral Strategies: Sequence-Form Strategies

Idea: Store probability for whole sequences of actions

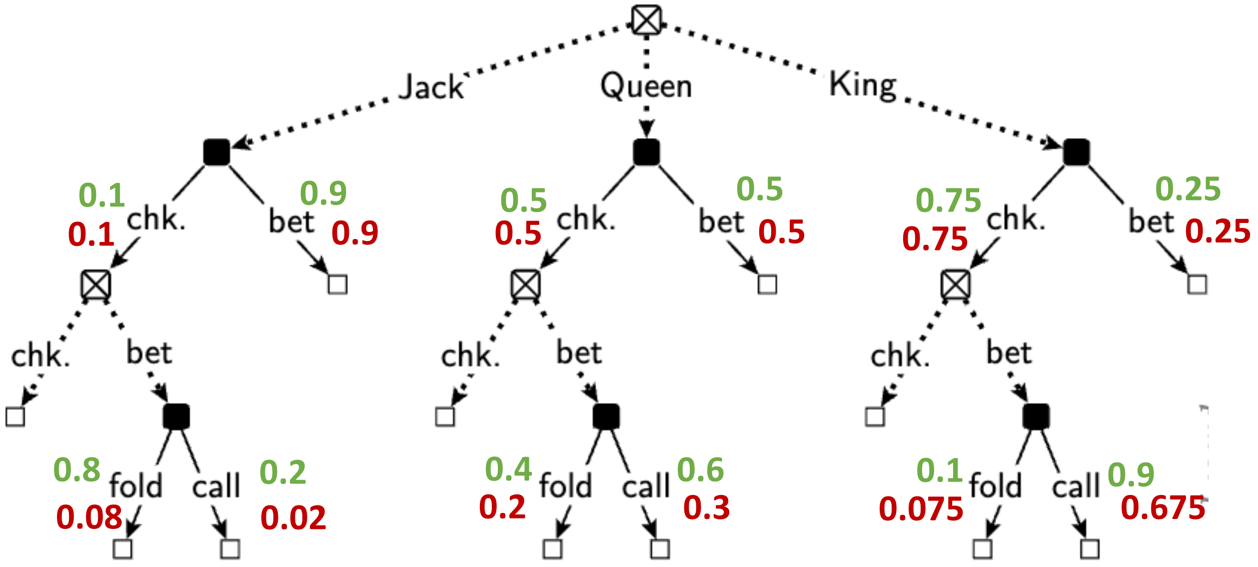


Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

⭐ **Consistency constraints**

1. Entries all non-negative
2. Root sequence has probability 1.0
3. Probability mass conservation

"Fixing" Behavioral Strategies: Sequence-Form Strategies



Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

Idea: Store probability for whole sequences of actions

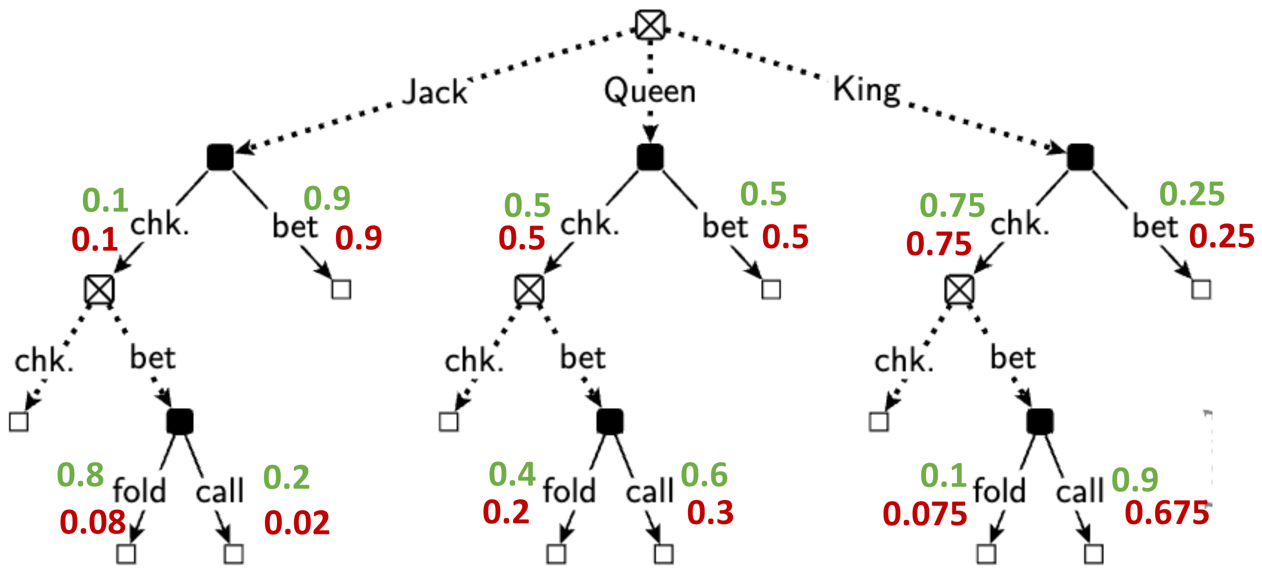
✔ Set of strategies is convex

⭐ **Consistency constraints**

1. Entries all non-negative
2. Root sequence has probability 1.0
3. Probability mass conservation

“Fixing” Behavioral Strategies: Sequence-Form Strategies

Idea: Store probability for whole sequences of actions



✔ Set of strategies is convex

✔ Expected utility is a linear function

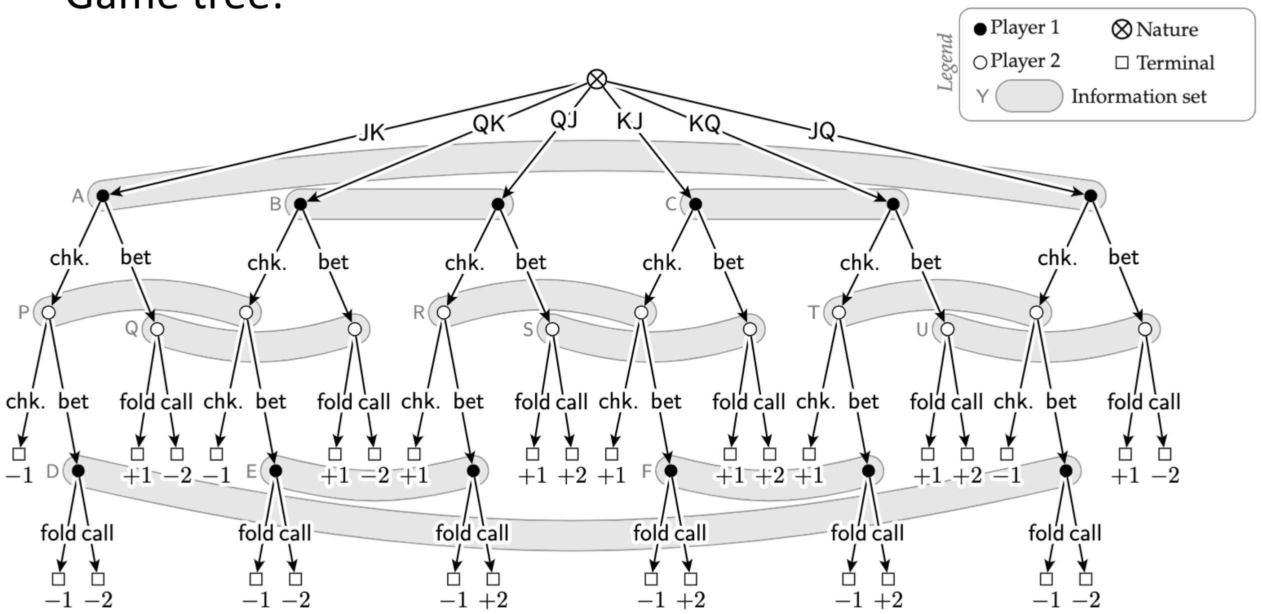
⭐ **Consistency constraints**

1. Entries all non-negative
2. Root sequence has probability 1.0
3. Probability mass conservation

Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

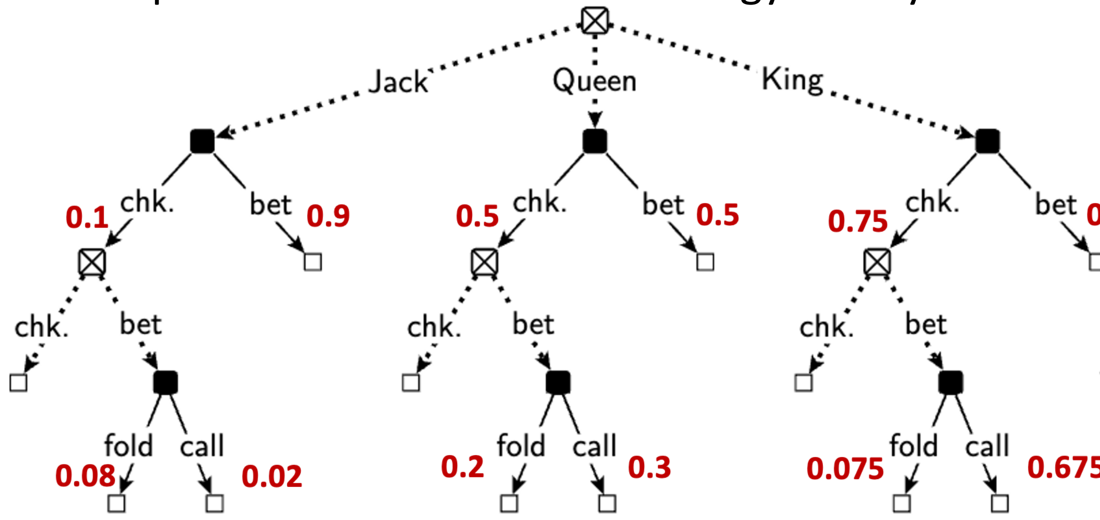
Expected Utility

Game tree:

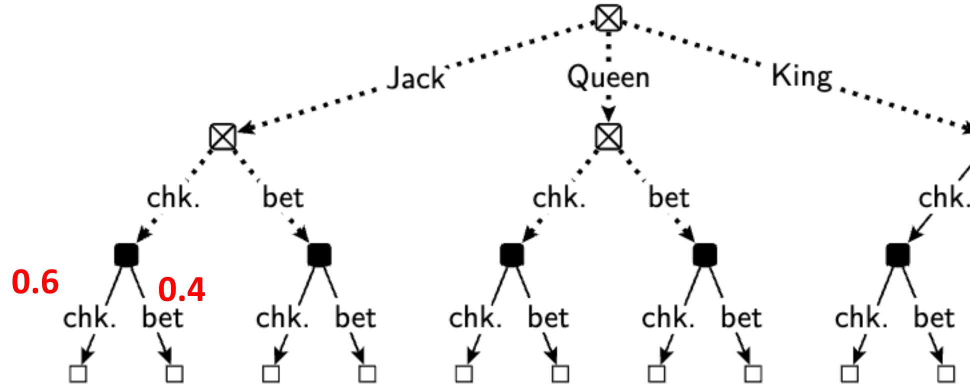


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

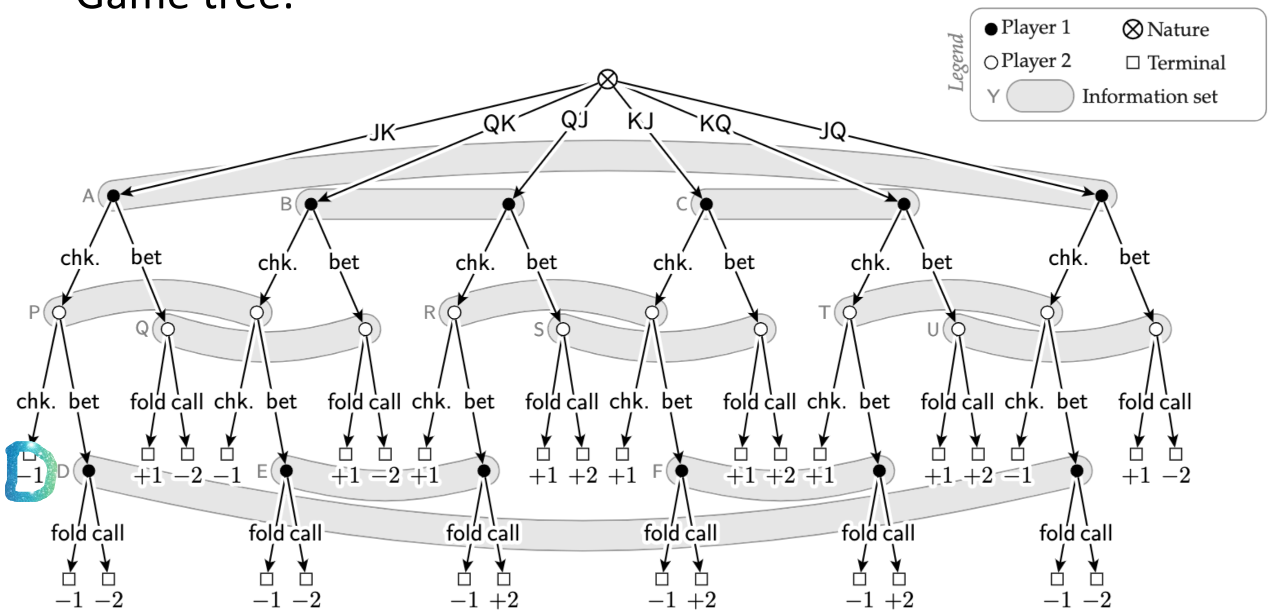


Decision problem and behavioral strategy of Player 2

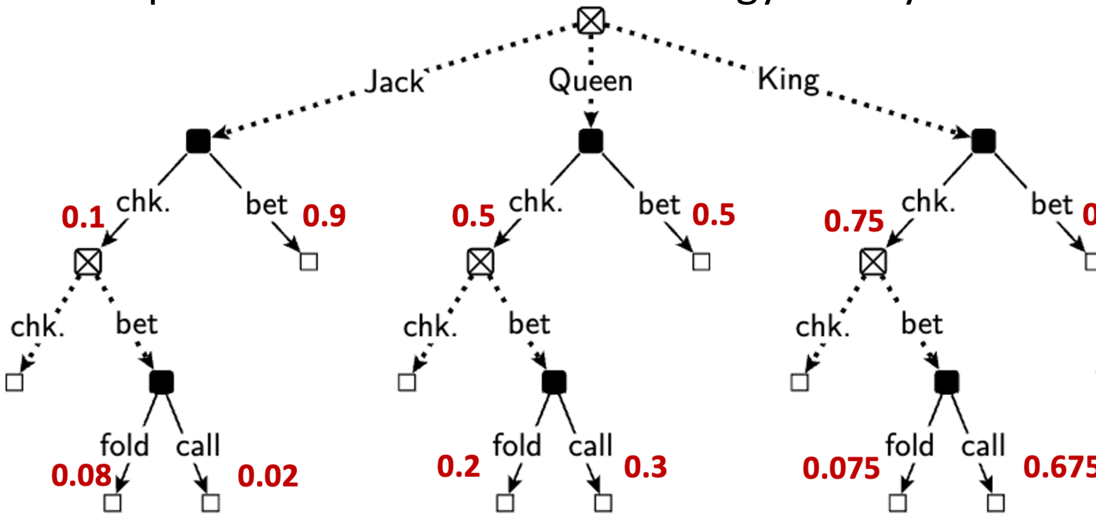


Expected Utility

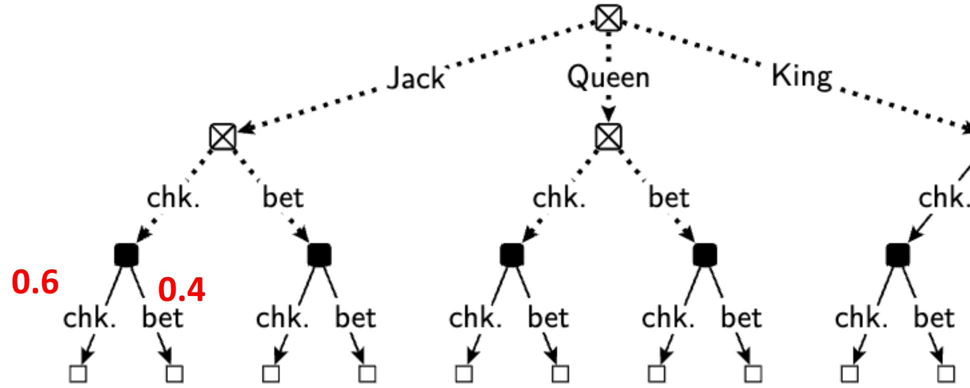
Game tree:



Decision problem and behavioral strategy of Player 1



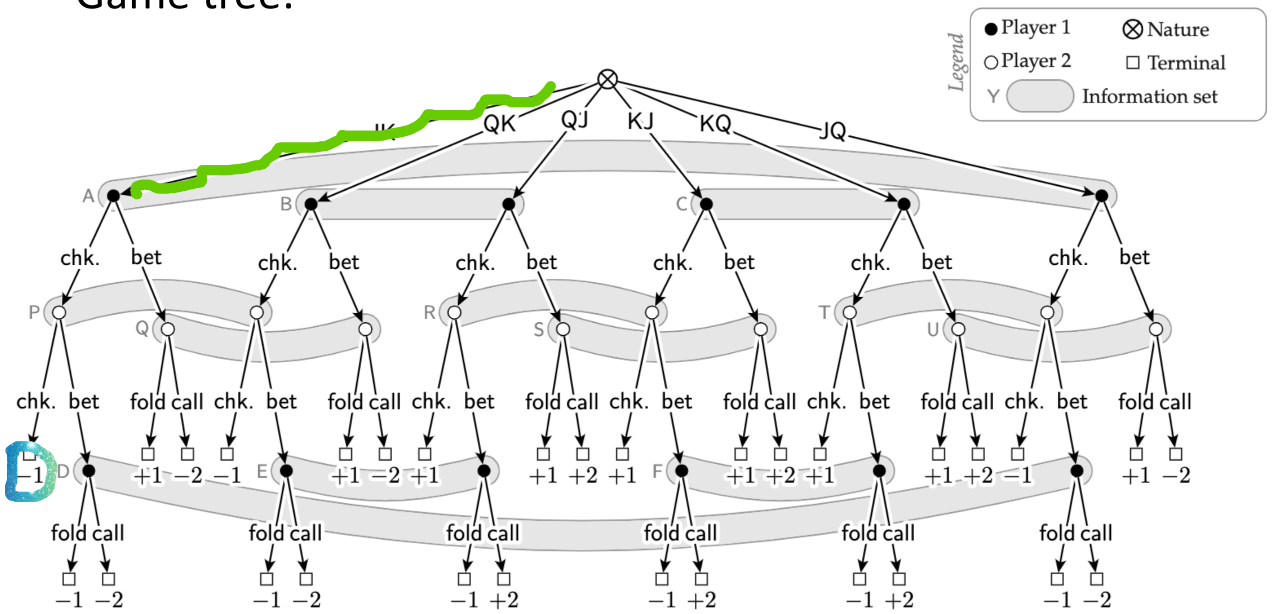
Decision problem and behavioral strategy of Player 2



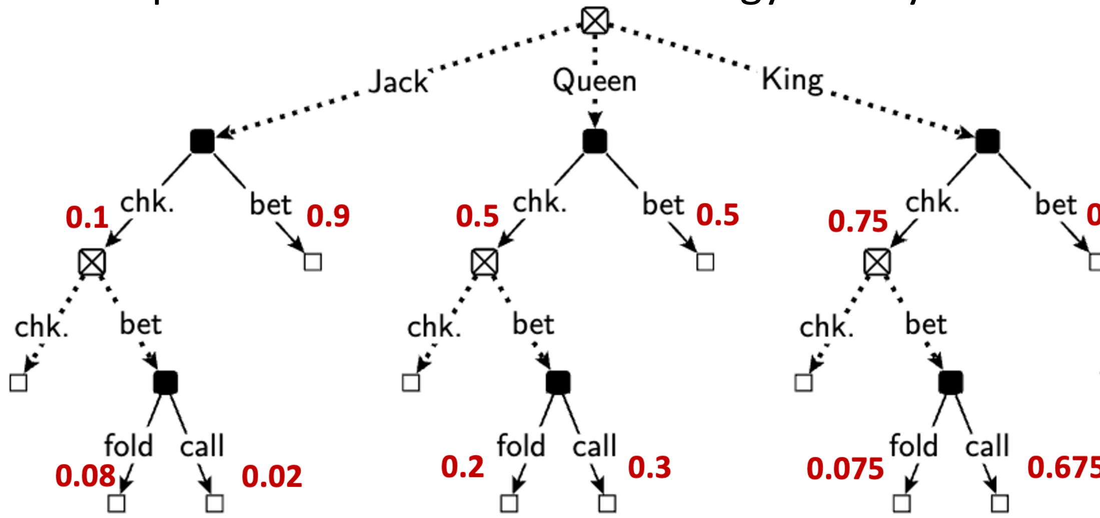
Prob of reaching this terminal state:

Expected Utility

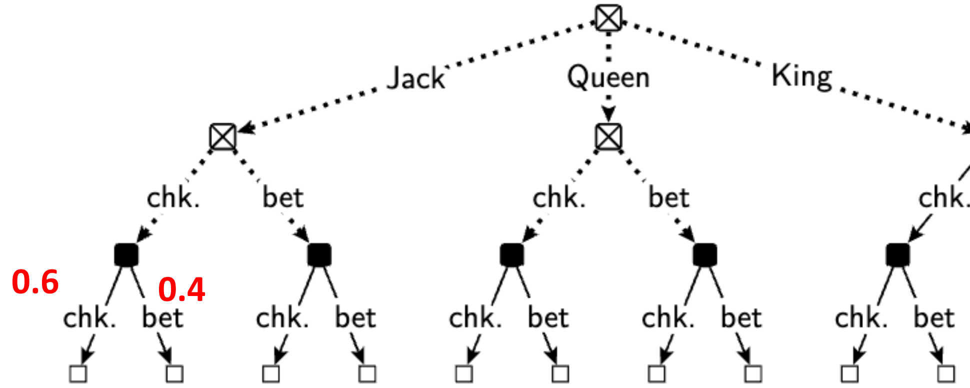
Game tree:



Decision problem and behavioral strategy of Player 1



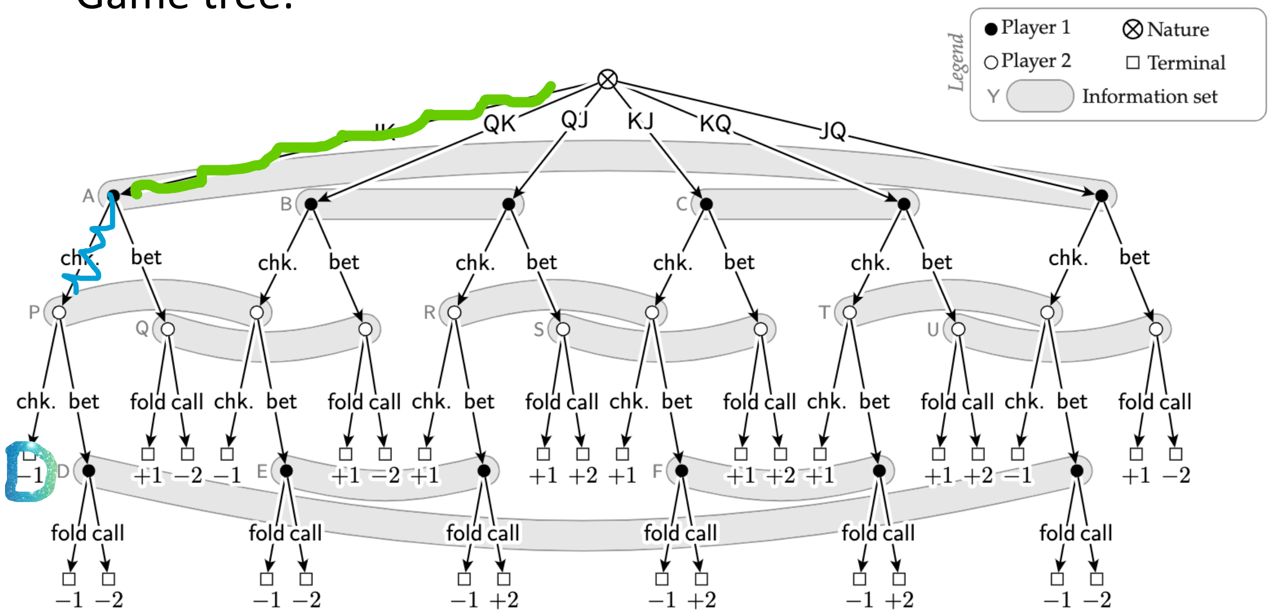
Decision problem and behavioral strategy of Player 2



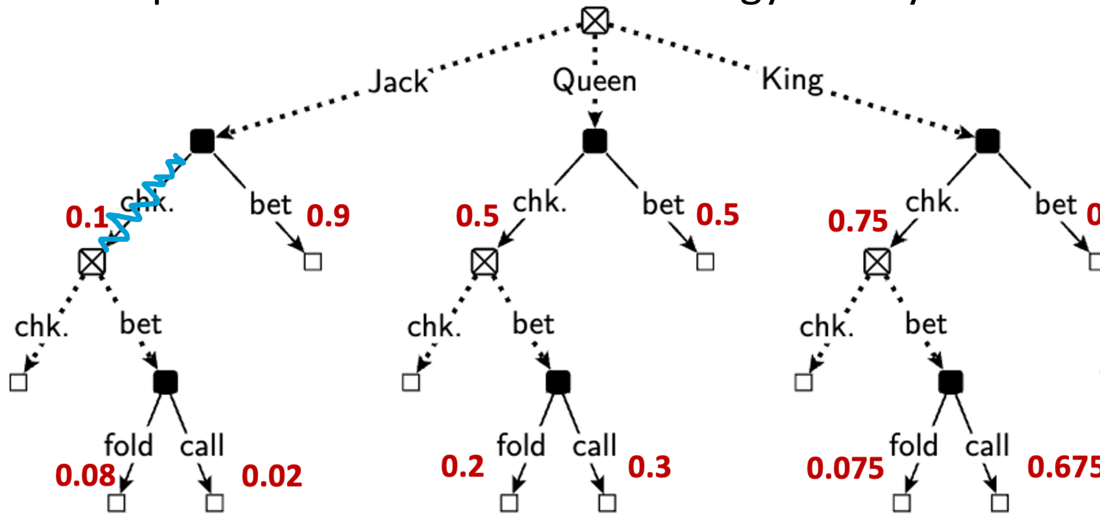
Prob of reaching this terminal state: 1/6 (Nature)

Expected Utility

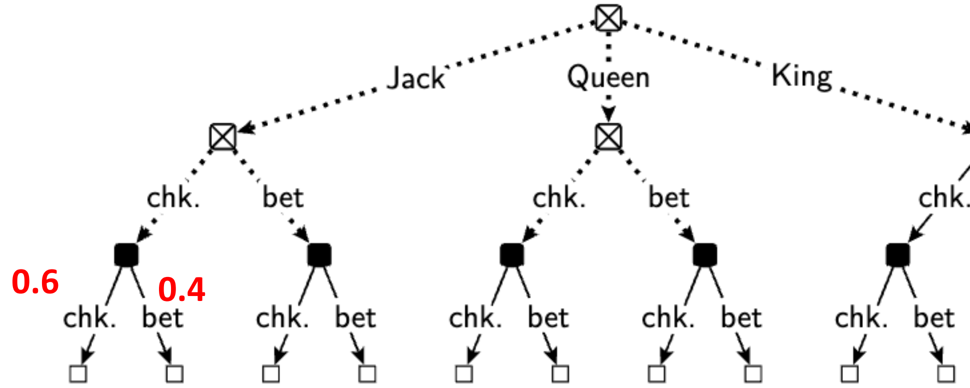
Game tree:



Decision problem and behavioral strategy of Player 1



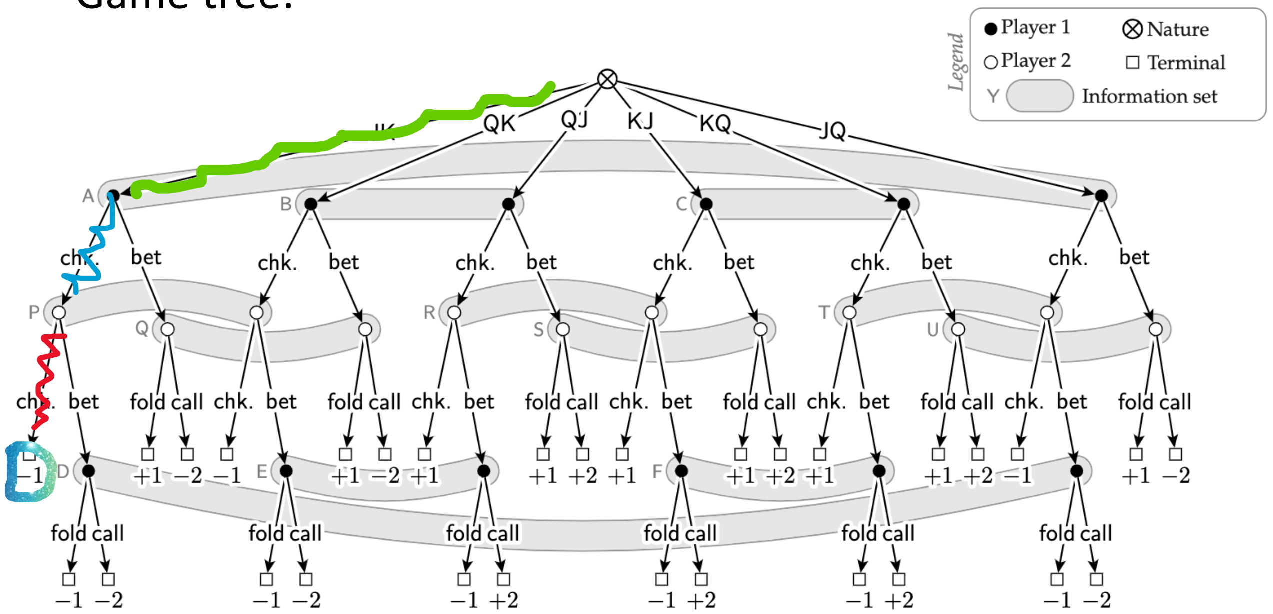
Decision problem and behavioral strategy of Player 2



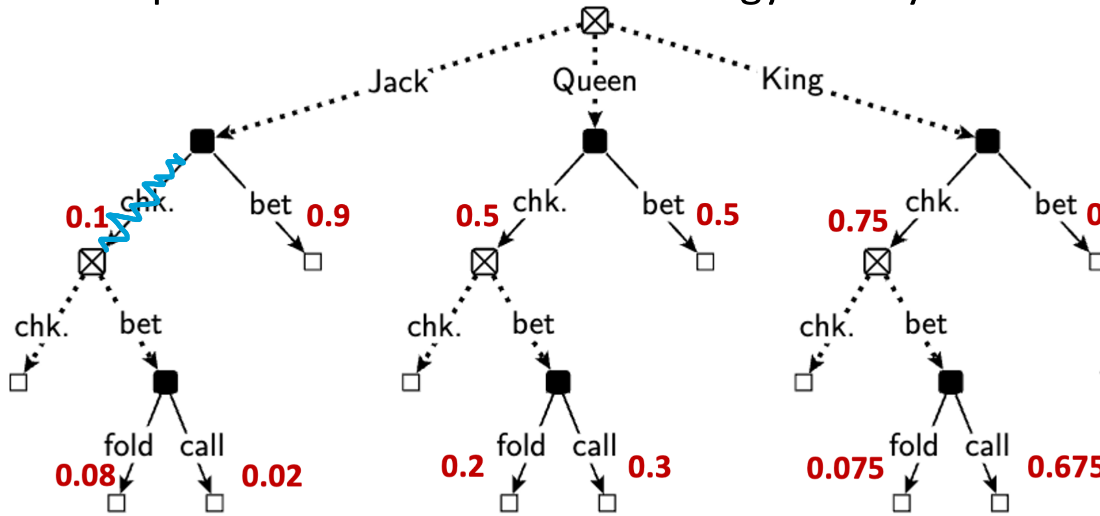
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1)

Expected Utility

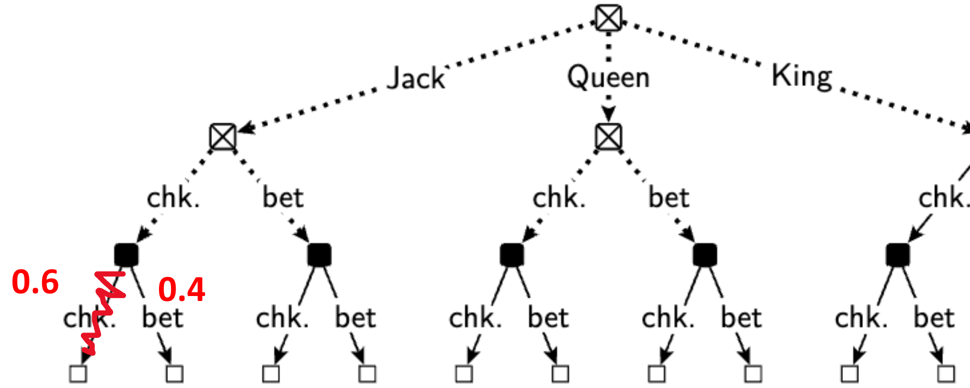
Game tree:



Decision problem and behavioral strategy of Player 1



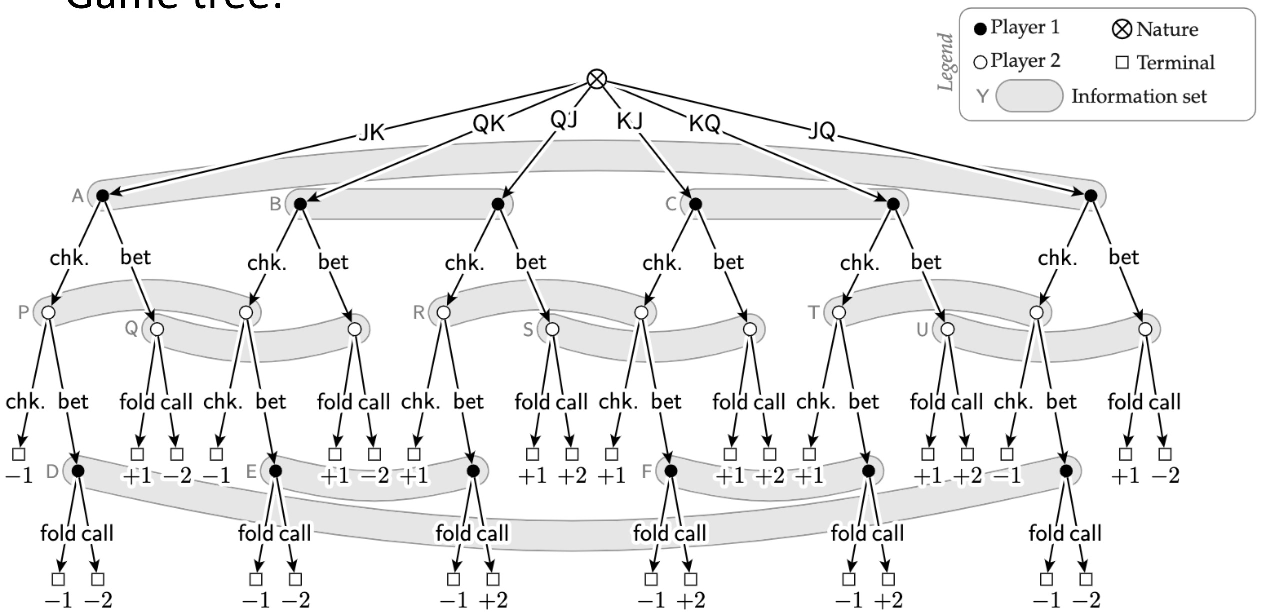
Decision problem and behavioral strategy of Player 2



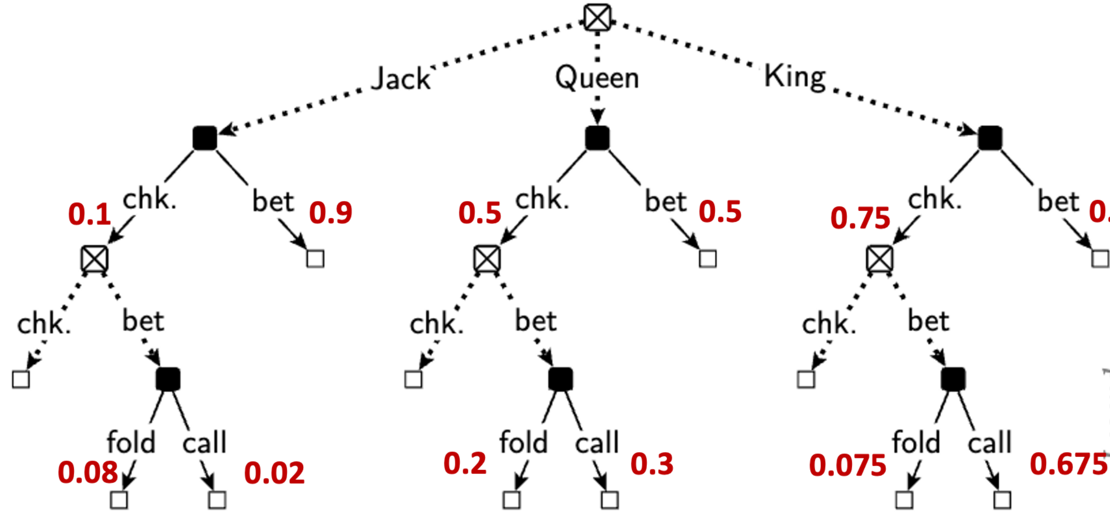
Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.6 (Pl2)

Expected Utility

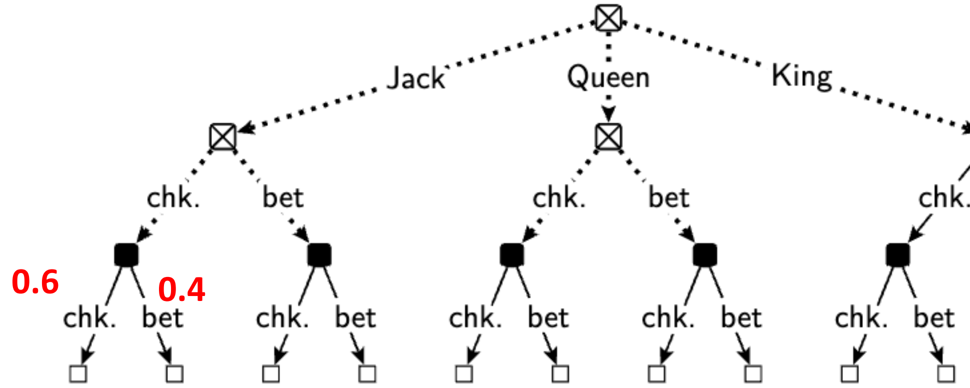
Game tree:



Decision problem and behavioral strategy of Player 1



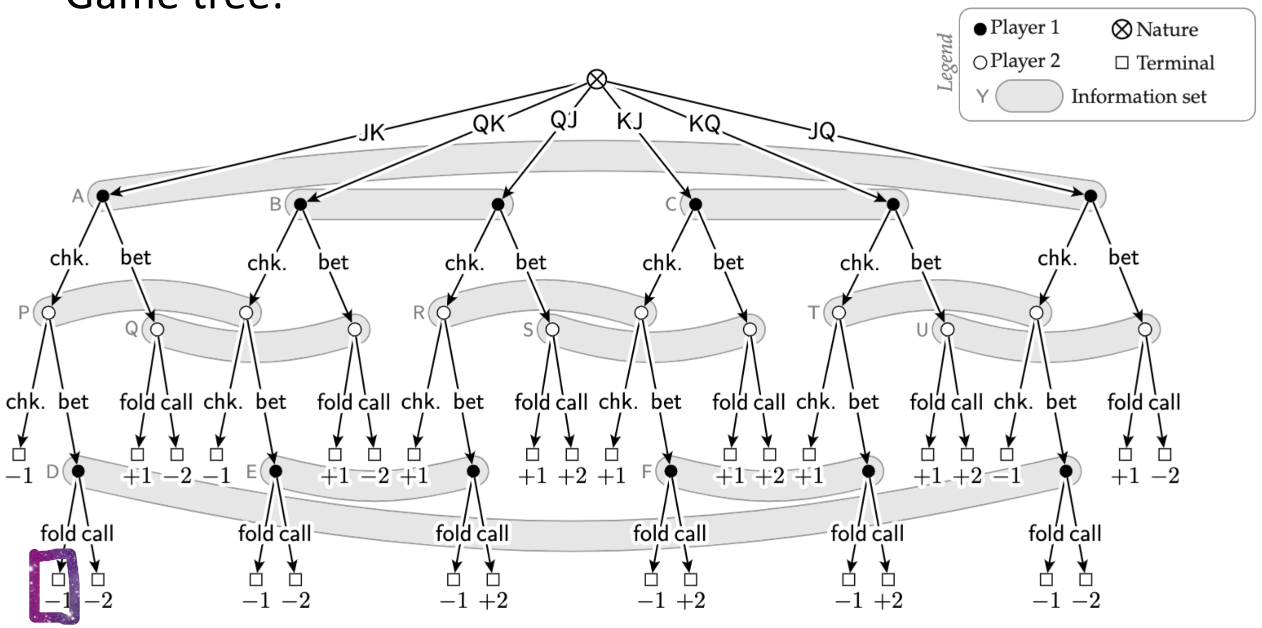
Decision problem and behavioral strategy of Player 2



Prob of reaching this terminal state:

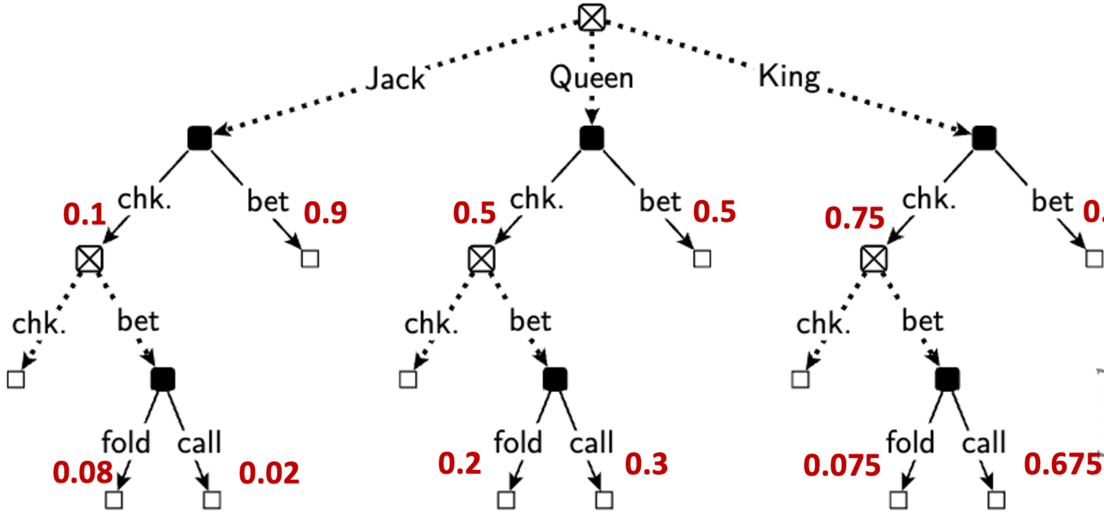
Expected Utility

Game tree:

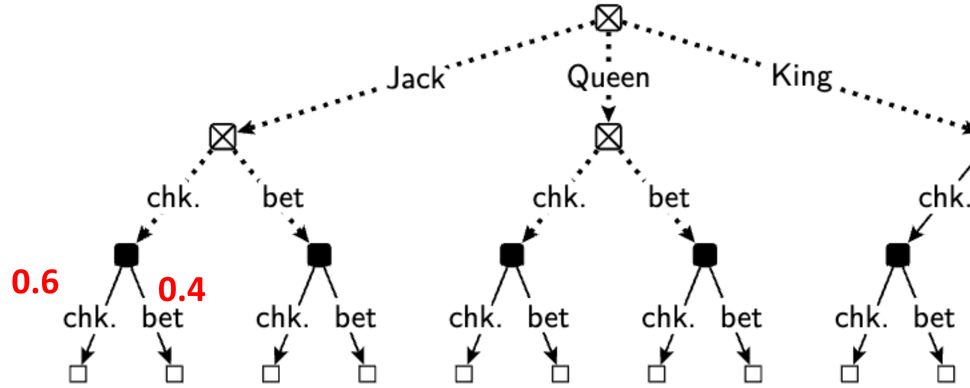


Prob of reaching this terminal state:

Decision problem and behavioral strategy of Player 1

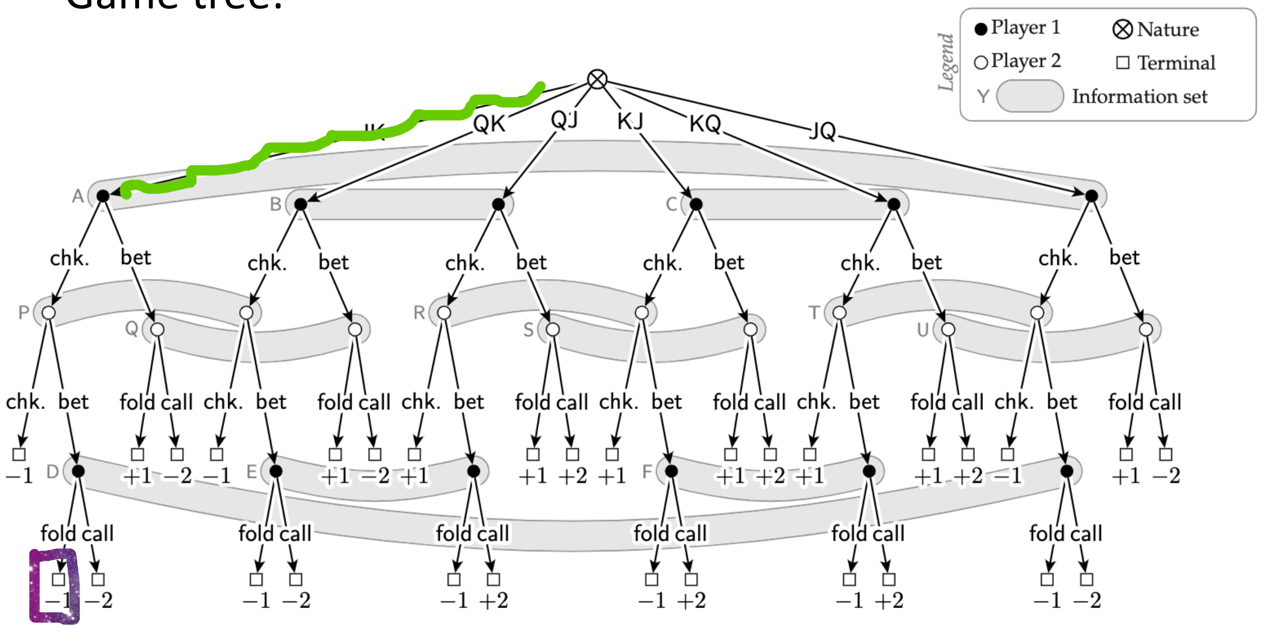


Decision problem and behavioral strategy of Player 2

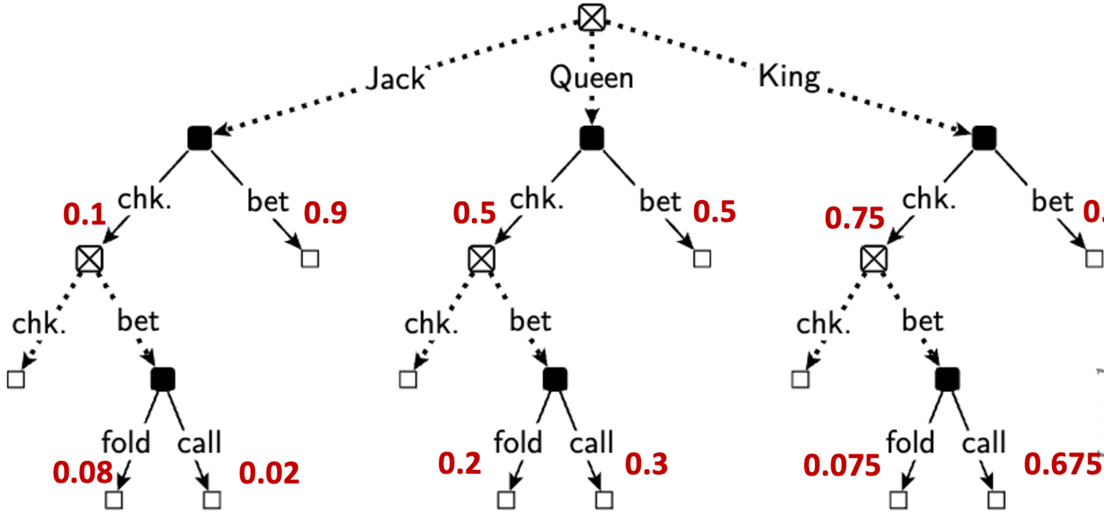


Expected Utility

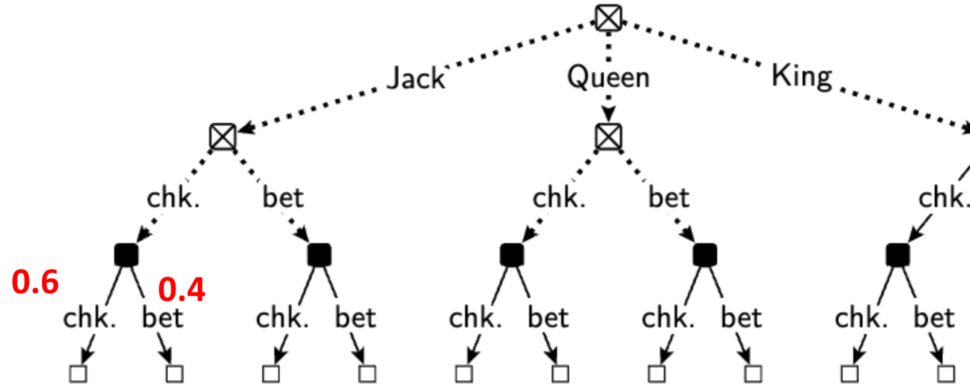
Game tree:



Decision problem and behavioral strategy of Player 1



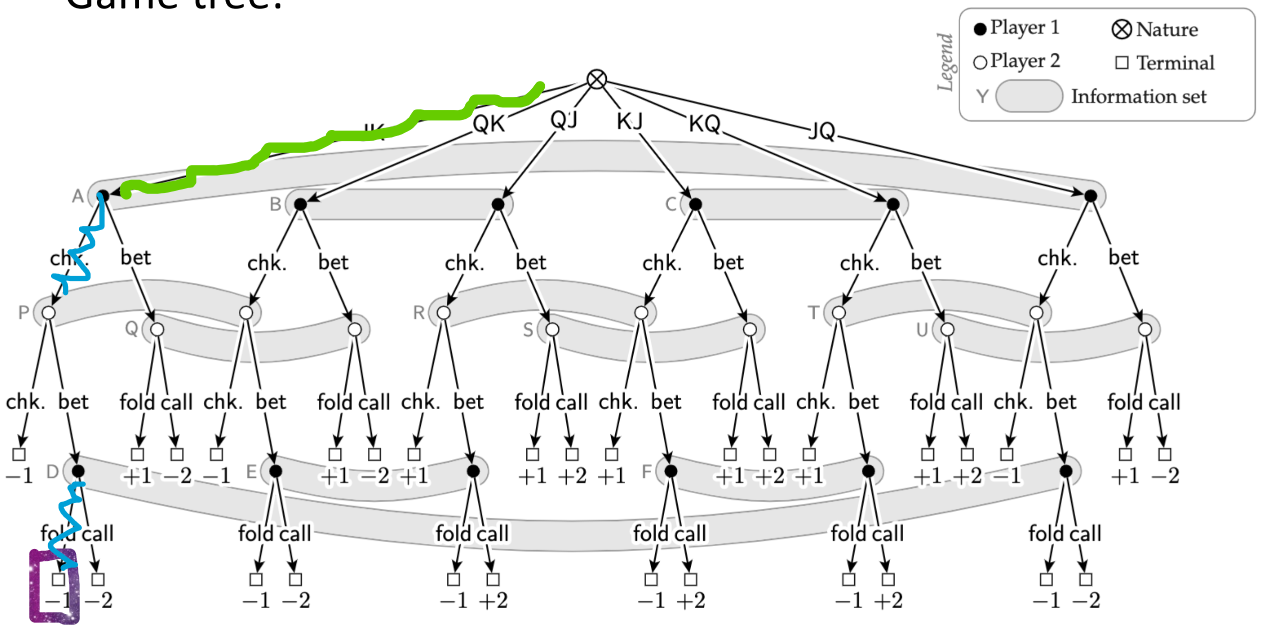
Decision problem and behavioral strategy of Player 2



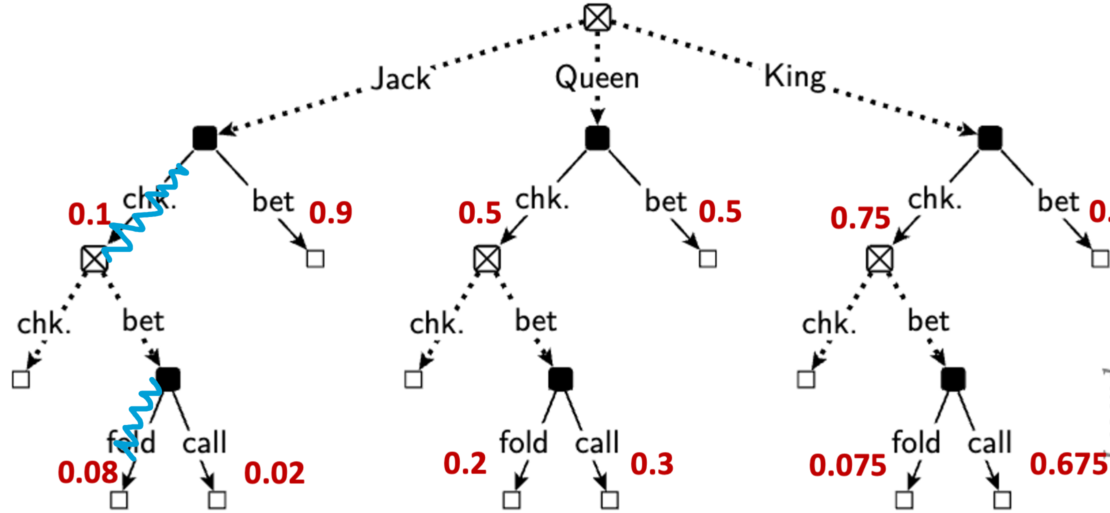
Prob of reaching this terminal state: 1/6 (Nature)

Expected Utility

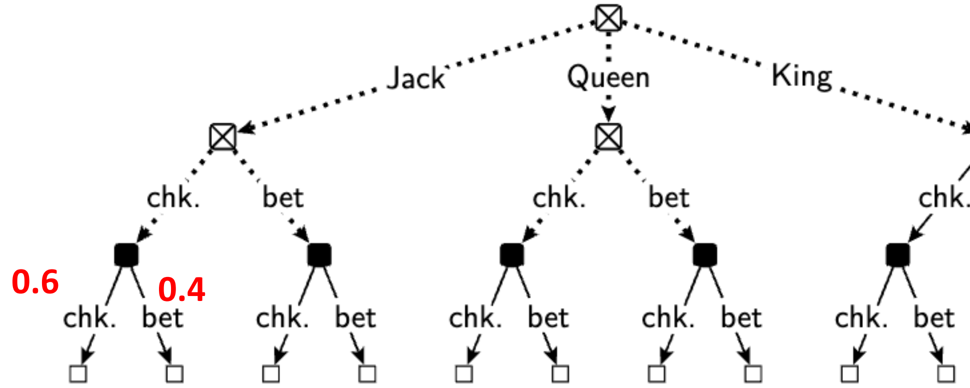
Game tree:



Decision problem and behavioral strategy of Player 1



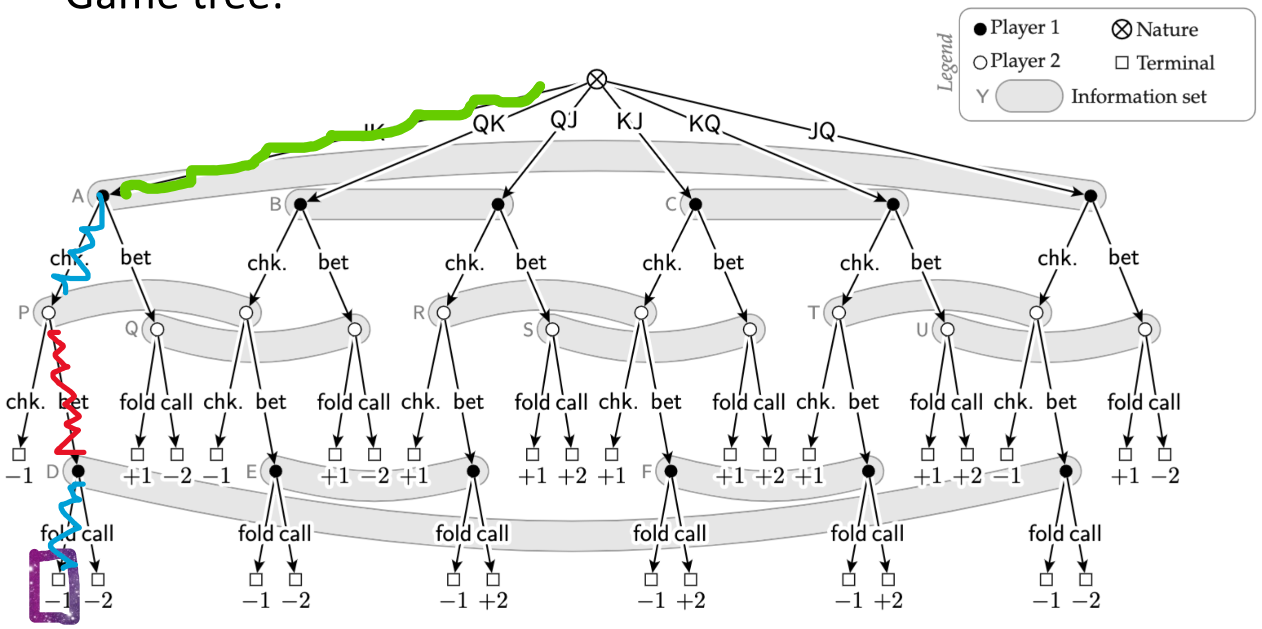
Decision problem and behavioral strategy of Player 2



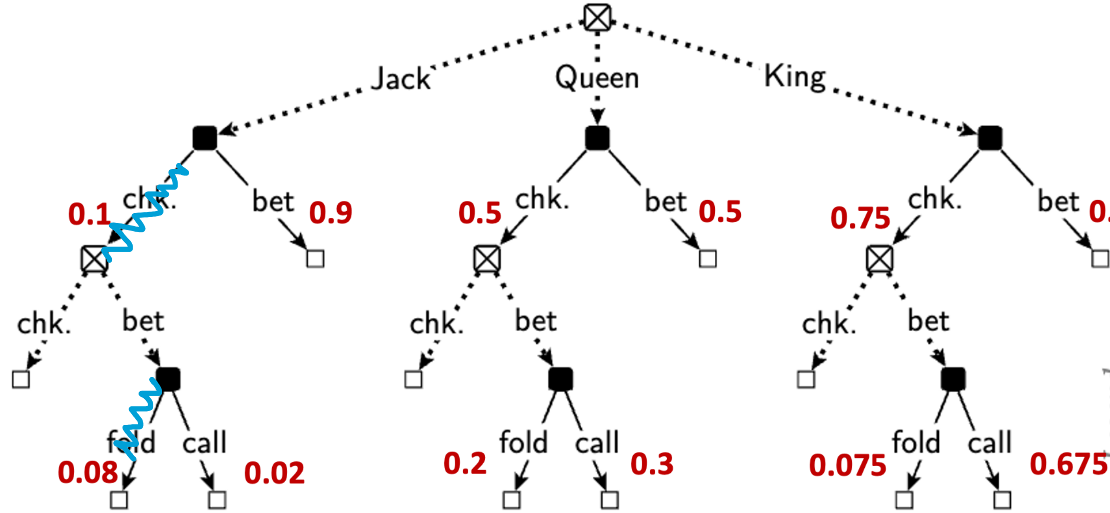
Prob of reaching this terminal state: 1/6 (Nature) x 0.08 (Pl1)

Expected Utility

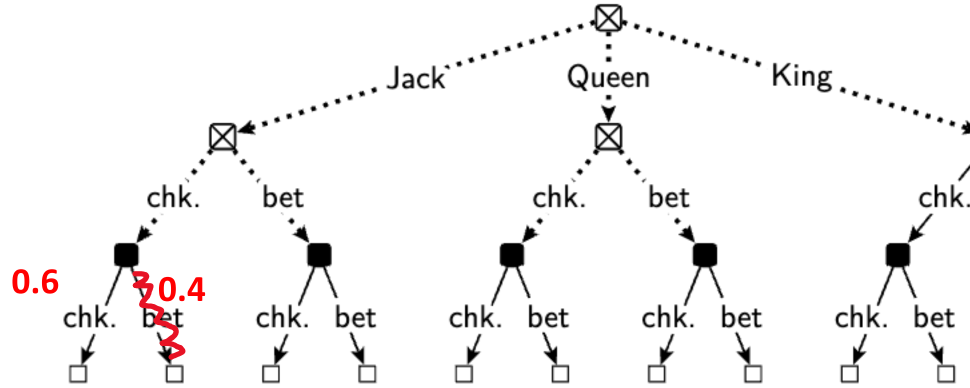
Game tree:



Decision problem and behavioral strategy of Player 1



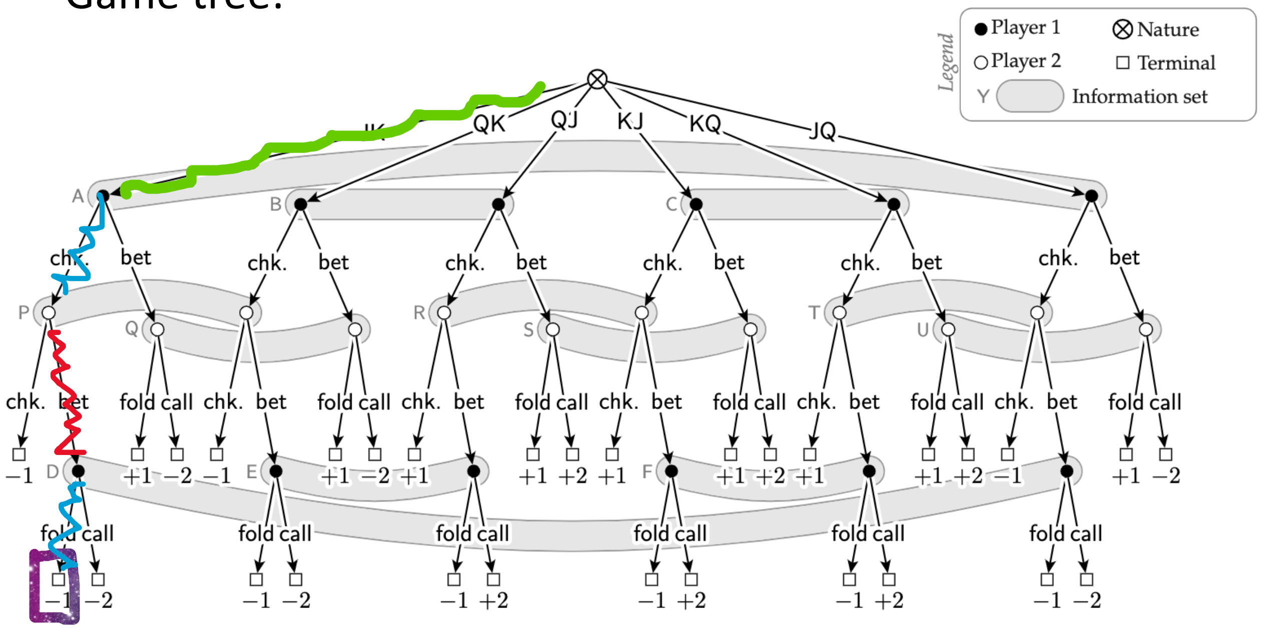
Decision problem and behavioral strategy of Player 2



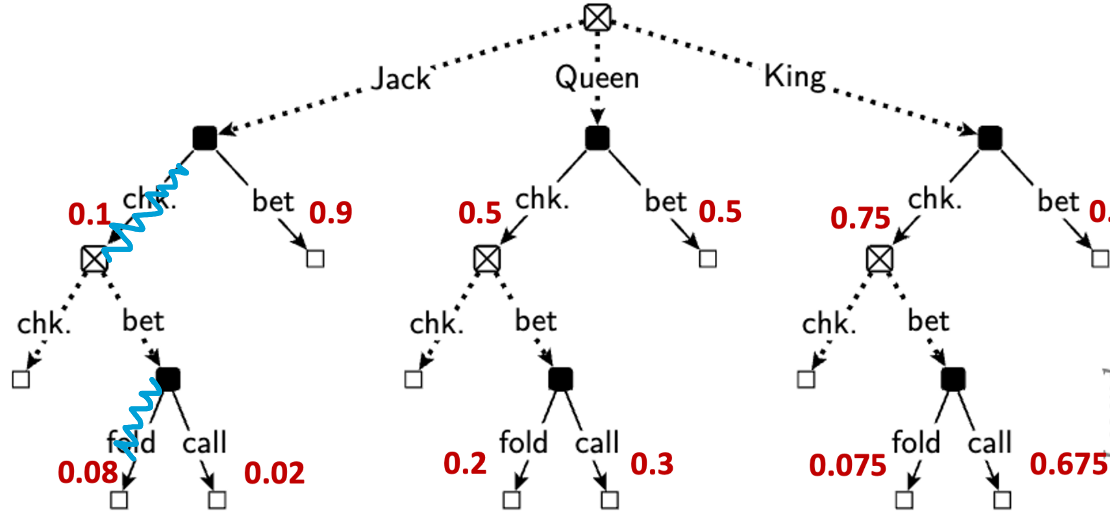
Prob of reaching this terminal state: 1/6 (Nature) x 0.08 (Pl1) x 0.4 (Pl2)

Expected Utility

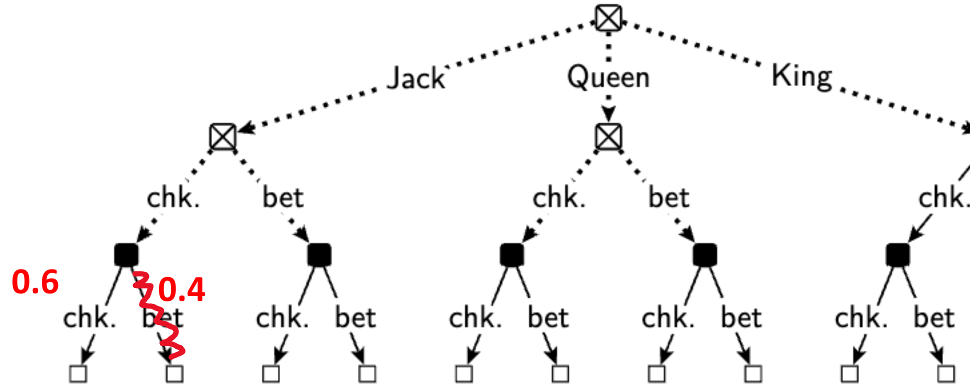
Game tree:



Decision problem and behavioral strategy of Player 1



Decision problem and behavioral strategy of Player 2



Prob of reaching this terminal state: 1/6 (Nature) x 0.08 (Pl1) x 0.4 (Pl2)

Nonlinearity is gone

Sequence-Form Representation

Expected utility is linear in every player's strategy (just like normal-form games)

Sequence-Form Representation

Expected utility is linear in every player's strategy (just like normal-form games)

Where did we pay a price? In normal-form games, strategy set is very simple (simplex). In extensive-form games, we have **sequence-form polytopes**

Sequence-Form Representation

Expected utility is linear in every player’s strategy (just like normal-form games)

Where did we pay a price? In normal-form games, strategy set is very simple (simplex). In extensive-form games, we have **sequence-form polytopes**

Everything still convex: We can use convex optimization tools

Equilibrium Computation (Extensive-Form)

BEFORE: Reduced−normal form

Nash equilibrium in Kuhn poker:

$$\max_{x} \min_{y} x^T A y$$

Distribution over the 27 plans of Player 1

Distribution over the 64 plans of Player 2

Payoff matrix on the left

You can use any technique for normal-form games: learning, linear programming, …

Equilibrium Computation (Extensive-Form)

BEFORE: Reduced–normal form

Nash equilibrium in Kuhn poker:

$$\max_x \min_y x^T A y$$

Distribution over the 27 plans of Player 1

Distribution over the 64 plans of Player 2

Payoff matrix on the left

You can use any technique for normal-form games: learning, linear programming, …

NOW: Sequence form

Nash equilibrium in Kuhn poker:

$$\max_{x'} \min_{y'} x'^T B y'$$

Sequence-form polytope of player 1 (dimension 12)

Sequence-form polytope of player 2 (dimension 12)

Sequence-form payoff matrix

You can **still** use learning, linear programming, …

Equilibrium Computation (Extensive-Form)

BEFORE: Reduced–normal form	NOW: Sequence form
Nash equilibrium in Kuhn poker:	Nash equilibrium in Kuhn poker:
$$\max_{x} \min_{y} x^T A y$$	$$\max_{x'} \min_{y'} x'^T B y'$$
Distribution over the 27 plans of Player 1	Sequence-form polytope of player 1 (dimension 12)
Distribution over the 64 plans of Player 2	Sequence-form polytope of player 2 (dimension 12)
Payoff matrix on the left	Sequence-form payoff matrix
You can use any tech learning, linear	You can **still** use learning, linear programming, …

Scale exponentially with tree size

Equilibrium Computation (Extensive-Form)

BEFORE: Reduced−normal form

Nash equilibrium in Kuhn poker:

$$\max_{x} \min_{y} x^T A y$$

Distribution over the 27 plans of Player 1

Distribution over the 64 plans of Player 2

Payoff matrix on the left

Scale exponentially with tree size

You can use any tech...
learning, linear progra...

NOW: Sequence form

Nash equilibrium in Kuhn poker:

$$\max_{x'} \min_{y'} x'^T B y'$$

Sequence-form polytope of player 1 (dimension 12)

Sequence-form polytope of player 2 (dimension 12)

Sequence-form payoff matrix

Scale **linearly** with tree size

You can **still** use learnin...

Recap

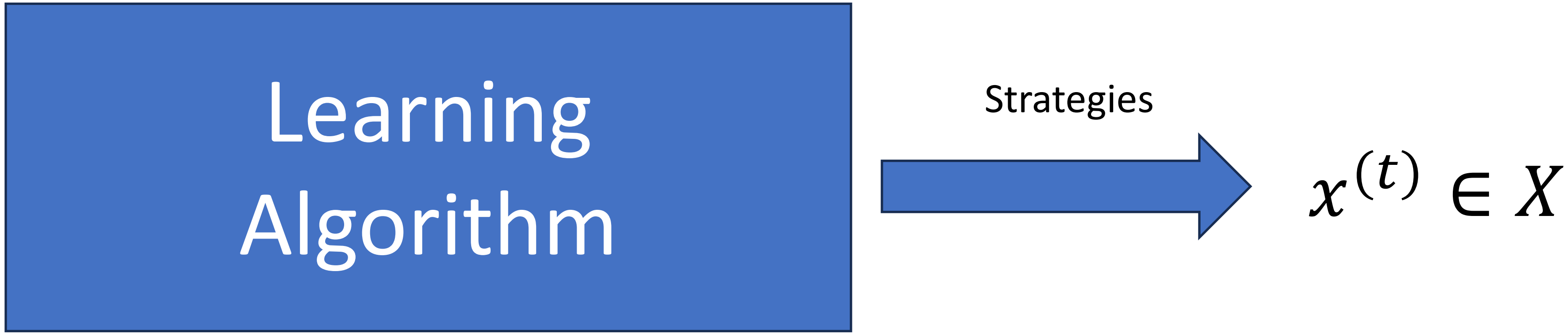
	Idea	Obvious downsides	Good news
(Reduced) Normal-form strategies	Distribution over deterministic strategies $\mu \in \Delta(\Pi)$	Exponentially-sized object	In rare cases, it's possible to operate implicitly on the exponential object via a kernel trick
Behavioral strategies	Local distribution over actions at each decision point $b \in \times_j \Delta(A_j)$	Expected utility is nonconvex in the the entries of vector b	Kuhn's theorem: same power as reduced normal-form strategies
Sequence-form strategies	**"Probability flows" on the tree-form decision process** $\boldsymbol{x} \in \boldsymbol{Q}$ **(convex polytope)**	**None**	**Everything is convex!** **Kuhn's theorem applies automatically.**

Learning in extensive-form games

Recall (Part I): No-External-Regret

Learning
Algorithm

Recall (Part I): No-External-Regret



Recall (Part I): No-External-Regret

X = Simplex for normal-form games

X = sequence-form polytope for extensive-form games

Learning Algorithm

Strategies

$x^{(t)} \in X$

Recall (Part I): No-External-Regret

X = Simplex for normal-form games

X = sequence-form polytope for extensive-form games

$u^{(t)}$ → Utility vectors → **Learning Algorithm** → Strategies → $x^{(t)} \in X$

Recall (Part I): No-External-Regret

X = Simplex for normal-form games

X = sequence-form polytope for extensive-form games

$u^{(t)}$ → Utility vectors → **Learning Algorithm** → Strategies → $x^{(t)} \in X$

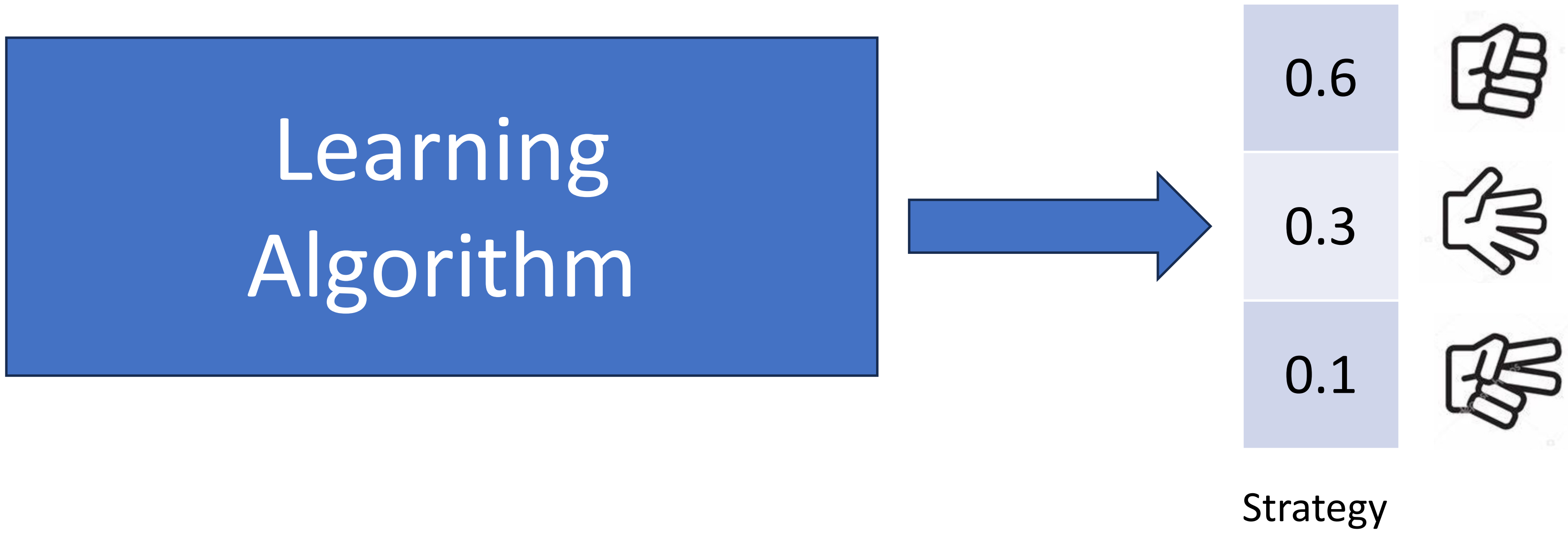
Objective: sublinear (external) regret

$$R^{(T)} := \max_{\hat{x} \in X} \sum_{t=1}^{T} \langle u^{(t)}, \hat{x} - x^{(t)} \rangle$$

Recall (Part I): Learning in Normal-Form Games

Learning
Algorithm

Recall (Part I): Learning in Normal-Form Games



Recall (Part I): Learning in Normal-Form Games

Recall (Part I): Learning Algorithms

Recall (Part I): Learning Algorithms

Regret matching (RM): Probability of each action proportional to ReLU of regret on the action

$$x^{(t)} \propto \left[r^{(t)}\right]^{+}$$

Recall (Part I): Learning Algorithms

Regret matching (RM): Probability of each action proportional to ReLU of regret on the action

$$x^{(t)} \propto \left[r^{(t)}\right]^+$$

Multiplicative Weights Update (MWU): Prob. of each action proportional to exp of regret on the action

$$x^{(t)} \propto \exp(\eta \cdot r^{(t)})$$

Recall (Part I): Learning Algorithms

Regret matching (RM): Probability of each action proportional to ReLU of regret on the action

$$x^{(t)} \propto \left[r^{(t)}\right]^{+}$$

Multiplicative Weights Update (MWU): Prob. of each action proportional to exp of regret on the action

$$x^{(t)} \propto \exp(\eta \cdot r^{(t)})$$

Follow-The-Regularized-Leader (FTRL):

$$x^{(t)} = \arg\max_{x \in \Delta} \langle r^{(t)}, x \rangle - \frac{1}{\eta}\psi(x)$$

Recall (Part I): Learning Algorithms

Regret matching (RM): Probability of each action proportional to ReLU of regret on the action

$$x^{(t)} \propto \left[r^{(t)}\right]^{+}$$

Multiplicative Weights Update (MWU): Prob. of each action proportional to exp of regret on the action

$$x^{(t)} \propto \exp(\eta \cdot r^{(t)})$$

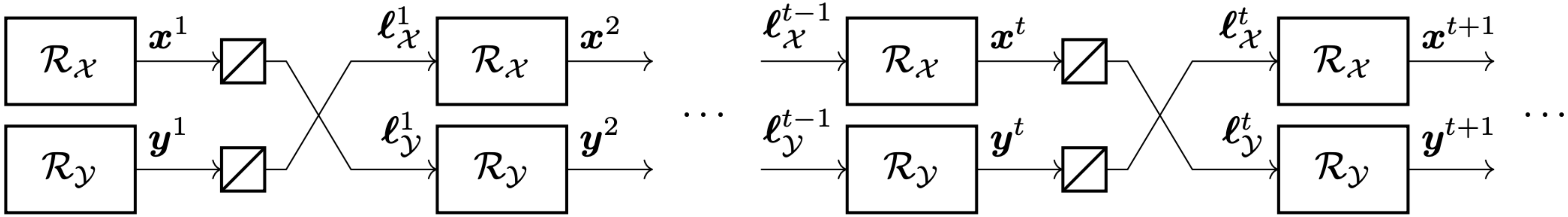
Recall: MWU is FTRL with negative entropy

Follow-The-Regularized-Leader (FTRL):

$$x^{(t)} = \arg\max_{x \in \Delta} \langle r^{(t)}, x \rangle - \frac{1}{\eta} \psi(x)$$

Recall (Part I): Connections with Equilibria

- Recall: when all players play external-regret-minimizing strategies, then:
  - In two-player zero-sum games, their average strategies converge to the set of Nash equilibrium (gives an alternative approach to previous lecture)
  - In general, the average product distribution of play converges to the set of coarse-correlated equilibria



$$\boldsymbol{\ell}_{\mathcal{X}}^t := \boldsymbol{A}\boldsymbol{y}^t, \qquad \boldsymbol{\ell}_{\mathcal{Y}}^t := -\boldsymbol{A}^\top \boldsymbol{x}^t$$

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Less specialized; general tool

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Less specialized; general tool

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Less specialized; general tool

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

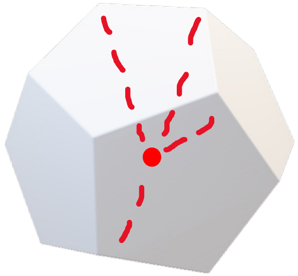
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Every point in the polytope is a convex combination of its finitely many vertices $\mathrm{V} \coloneqq \{v_1, \dots, v_m\}$. So, operate a change of **variable:** learn the convex combination, not the points $x^{(t)}$

$$R^{(T)} \coloneqq \max_{\hat{x} \in X} \sum_{t=1}^{T} \langle u^{(t)}, \hat{x} - x^{(t)} \rangle$$

Perf. of vertex

$$R^{(T)} \coloneqq \max_{\hat{\lambda} \in \Delta(V)} \sum_{t=1}^{T} \left\langle \begin{pmatrix} \vdots \\ \langle u^{(t)}, v \rangle \\ \vdots \end{pmatrix}, \hat{\lambda} - \lambda^{(t)} \right\rangle$$

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

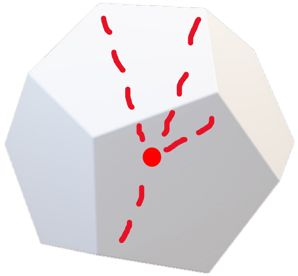
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Key question:

How to sidestep exponential size?

Every point in the polytope is a convex combination of its finitely many vertices $V := \{v_1, \dots, v_m\}$. So, operate a change of **variable:** learn the convex combination, not the points $x^{(t)}$

$$R^{(T)} := \max_{\hat{x} \in X} \sum_{t=1}^{T} \langle u^{(t)}, \hat{x} - x^{(t)} \rangle$$

Perf. of vertex

$$R^{(T)} := \max_{\hat{\lambda} \in \Delta(V)} \sum_{t=1}^{T} \left\langle \begin{pmatrix} \vdots \\ \langle u^{(t)}, v \rangle \\ \vdots \end{pmatrix}, \hat{\lambda} - \lambda^{(t)} \right\rangle$$

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

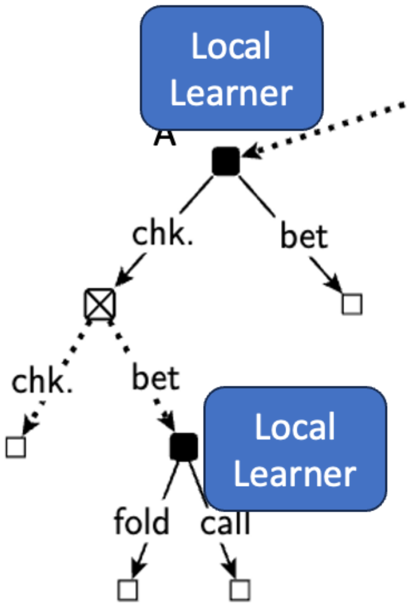
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Run a local no-regret algorithm at each decision point to update your strategy.

"Process" the utility vector $u^{(t)}$ (which is for the whole sequence-form strategy) and chop it up into local feedback for each decision point.

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

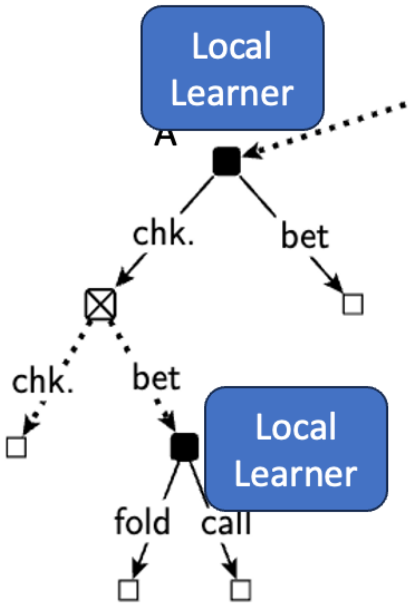
Less specialized; general tool

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Main idea:



Key question:

What is the local feedback?

Run a local no-regret algorithm at each decision point to update your strategy.

"Process" the utility vector $u^{(t)}$ (which is for the whole sequence-form strategy) and chop it up into local feedback for each decision point.

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:

The sequence-form polytope is a convex set. So, we can apply the FTRL algorithm in its general form, and that guarantees no-regret

$$x^{(t)} = \arg\max_{x \in Q} \langle U^{(t)}, x \rangle - \frac{1}{\eta}\psi(x)$$

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:

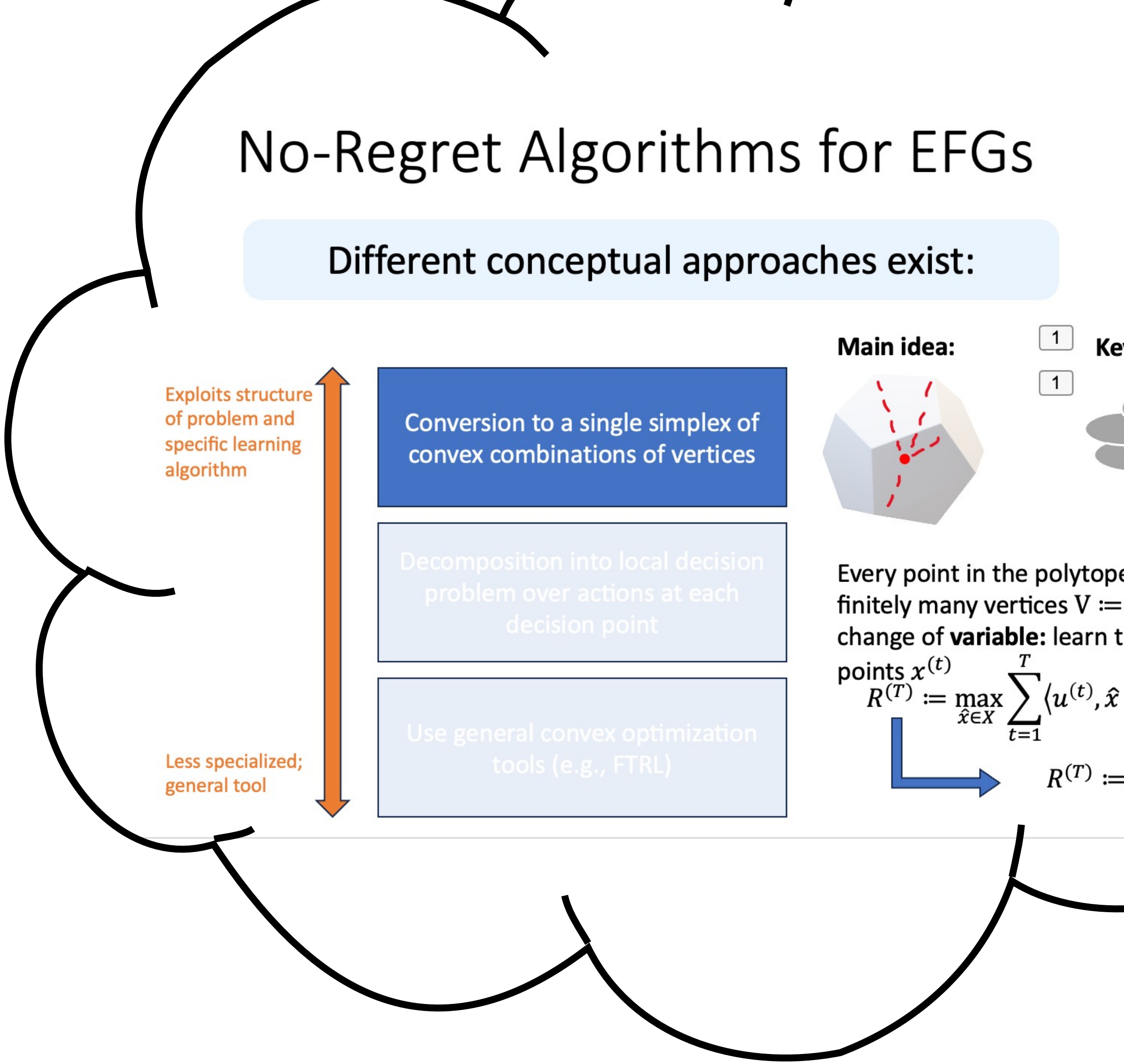
The sequence-form polytope is a convex set. So, we can apply the FTRL algorithm in its general form, and that guarantees no-regret

$$x^{(t)} = \arg\max_{x \in Q} \langle U^{(t)}, x \rangle - \frac{1}{\eta}\psi(x)$$

Key question:

What regularizers are easy to deal with?

Kernelized MWU



General Setup:

$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$ polyhedral strategy set for Player i (e.g., sequence-form polytope for EFGs) with 0/1 vertices

V_i vertices of Ω_i

Vertex MWU algorithm

Setup
$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$
V_i vertices of Ω_i

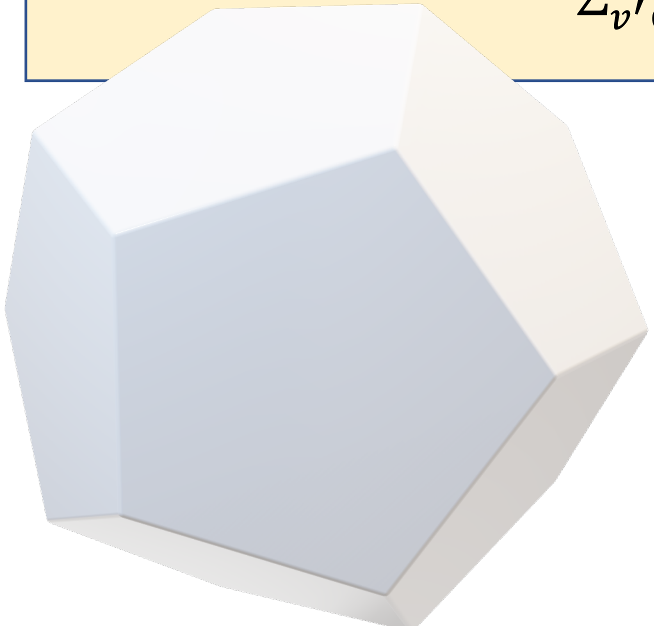
$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$



General Setup:

$\Omega_i \subseteq \mathbb{R}^d$ polyhedral strategy set for Player i (e.g., sequence-form polytope for EFGs) with 0/1 vertices

V_i vertices of Ω_i

Vertex MWU algorithm

Setup
$\Omega_i \subseteq \mathbb{R}^d$
V_i vertices of Ω_i

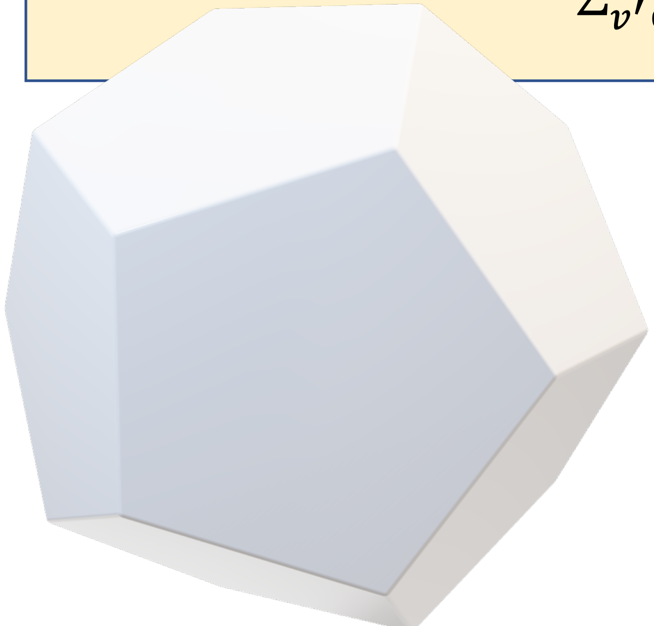
$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$



"Utility of vertex v"

…We weight vertices using MWU

Vertex MWU algorithm

Setup

$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$

V_i vertices of Ω_i

$\lambda^{(1)} \coloneqq \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Main theorem

When Ω_i has 0/1-coordinate vertices, Vertex MWU can be implemented using *d+1* evaluations of the 0/1-polyhedral kernel at each iteration

Vertex MWU algorithm

Setup

$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$

V_i vertices of Ω_i

$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Main theorem

When Ω_i has 0/1-coordinate vertices, Vertex MWU can be implemented using *d+1* evaluations of the 0/1-polyhedral kernel at each iteration

Vertex MWU algorithm

Setup
$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$
V_i vertices of Ω_i

$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Crucially independent on the number of vertices of Ω_i!

As long as the kernel function can be evaluated efficiently, then Vertex (O)MWU can be simulated in polynomial time

Setup

$\Omega \subseteq \mathbb{R}^d$

V vertices of Ω

$V \subseteq \{0,1\}^d$

Setup

$\Omega \subseteq \mathbb{R}^d$

V vertices of Ω

$V \subseteq \{0,1\}^d$

Definition (0/1-feature map of Ω)

$$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \qquad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$$

Setup

$\Omega \subseteq \mathbb{R}^d$

V vertices of Ω

$V \subseteq \{0,1\}^d$

Definition (0/1-feature map of Ω)

$$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \qquad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$$

Given any vector, for each vertex it computes the product of the coordinates that are hot for that vertex

Setup

$\Omega \subseteq \mathbb{R}^d$

V vertices of Ω

$V \subseteq \{0,1\}^d$

Definition (0/1-feature map of Ω)

$$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \qquad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$$

Given any vector, for each vertex it computes the product of the coordinates that are hot for that vertex

Definition (0/1-polyhedral kernel of Ω)

$$K_\Omega : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \quad K_\Omega(x,y) \coloneqq \langle \phi_\Omega(x), \phi_\Omega(y) \rangle = \sum_{v\in V} \prod_{k:v[k]=1} x[k] \cdot y[k]$$

Let’s see how the feature map and the kernel help simulate Vertex MWU

Idea #1

Recall (feature map):
$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \dots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Idea #1

$$\boldsymbol{\lambda}^{(t)} = \frac{\phi_\Omega(\boldsymbol{b}^{(t)})}{K_\Omega(\boldsymbol{b}^{(t)}, \mathbf{1})}$$

Recall (feature map):
$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

Lemma 1: At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

Proof: by induction

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Idea #1

$$\boldsymbol{\lambda}^{(t)} = \frac{\phi_\Omega(\boldsymbol{b}^{(t)})}{K_\Omega(\boldsymbol{b}^{(t)}, \mathbf{1})}$$

Recall (feature map):
$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

Lemma 1: At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

Proof: by induction

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Consequence: by keeping track of $b^{(t)}$ we are implicitly keeping track of $\lambda^{(t)}$ as well

…So, no need to actually perform the update on line 5 explicitly

Idea #1

Recall (feature map):
$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

Lemma 1: At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\left(\sum^{t-1} \right)$$

Pr

Remaining obstacle: how can we evaluate line 3 with only implicit access to $\lambda^{(t)}$ via $b^{(t)}$?

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Consequence: by keeping track of $b^{(t)}$ we are implicitly keeping track of $\lambda^{(t)}$ as well

…So, no need to actually perform the update on line 5 explicitly

Idea #2

Lemma 1: At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

Vertex MWU algorithm

Setup

$\Omega \subseteq \mathbb{R}^d$

V vertices of Ω

$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Idea #2

Lemma 1: At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} := \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} := \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Lemma 2: At all times t, $x^{(t)}$ can be reconstructed from $b^{(t)}$ as

$$x^{(t)} = \left(1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_1)}{K_\Omega(b^{(t)}, \mathbf{1})}, \ldots, 1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_d)}{K_\Omega(b^{(t)}, \mathbf{1})}\right)$$

(d+1 kernel evaluations)

Vertex MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Kernelized MWU algorithm

Setup
$\Omega \subseteq \mathbb{R}^d$
V vertices of Ω
$V \subseteq \{0,1\}^d$

$b^{(1)} \coloneqq \mathbf{1} \in \mathbb{R}^d$

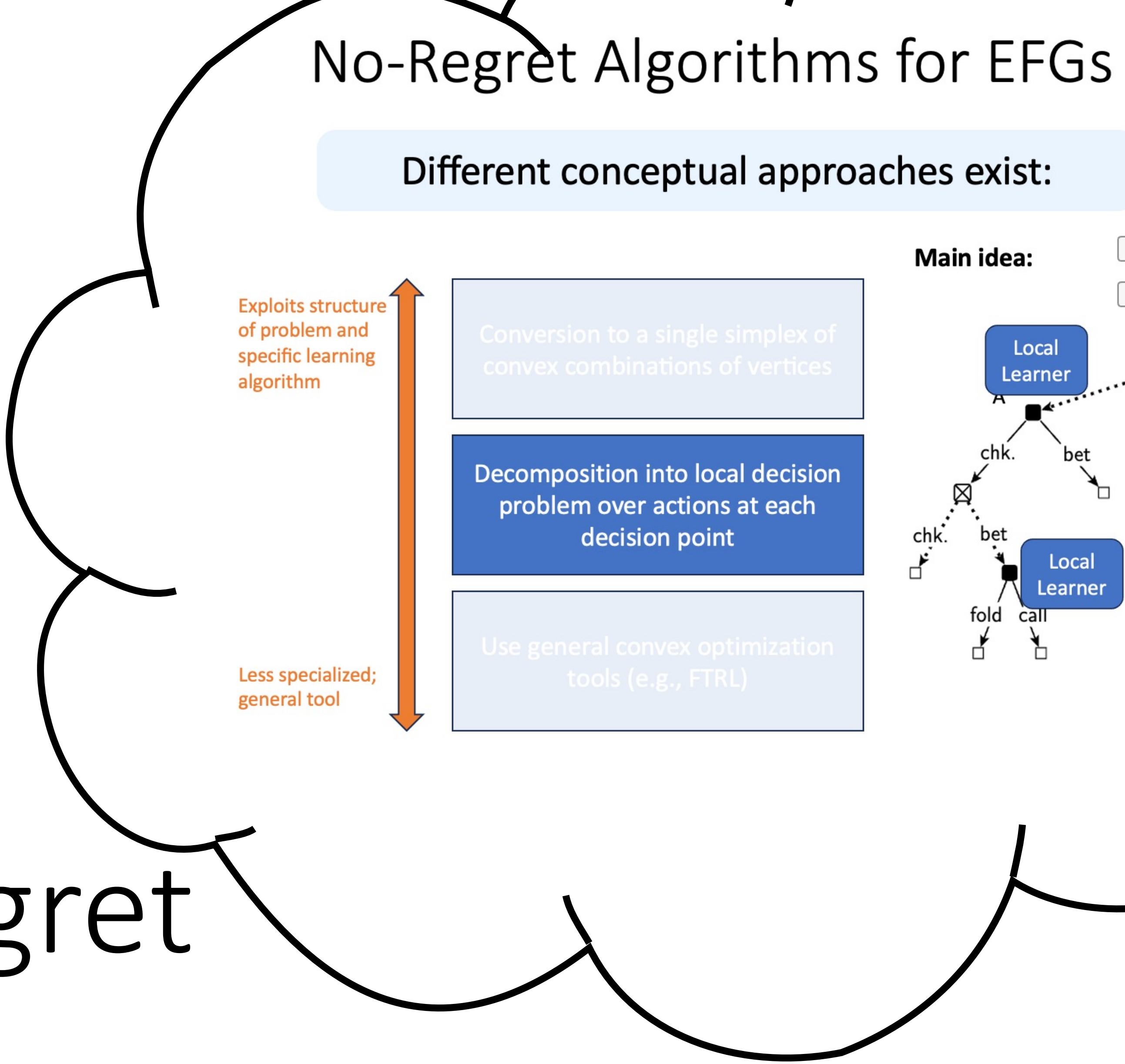
For $t = 1, 2, \ldots$

Play $x^{(t)} \coloneqq \left(1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_1)}{K_\Omega(b^{(t)}, \mathbf{1})}, \ldots, 1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_d)}{K_\Omega(b^{(t)}, \mathbf{1})}\right)$

Observe utility $u^{(t)} \in \mathbb{R}^d$

Set $b^{(t+1)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t} u^{(\tau)}\right\}$

Handout with proof at: https://www.mit.edu/~gfarina/2024/mit_theory_reading_group_komwu_2024/



Counterfactual Regret Minimization

Counterfactual Regret Minimization

Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point

Counterfactual Regret Minimization

Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point

🚨
Papercut Alert™

Counterfactual Regret Minimization

Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point

🚨
Papercut Alert™

CFR updates strategies in *behavioral* form…

Counterfactual Regret Minimization

Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point

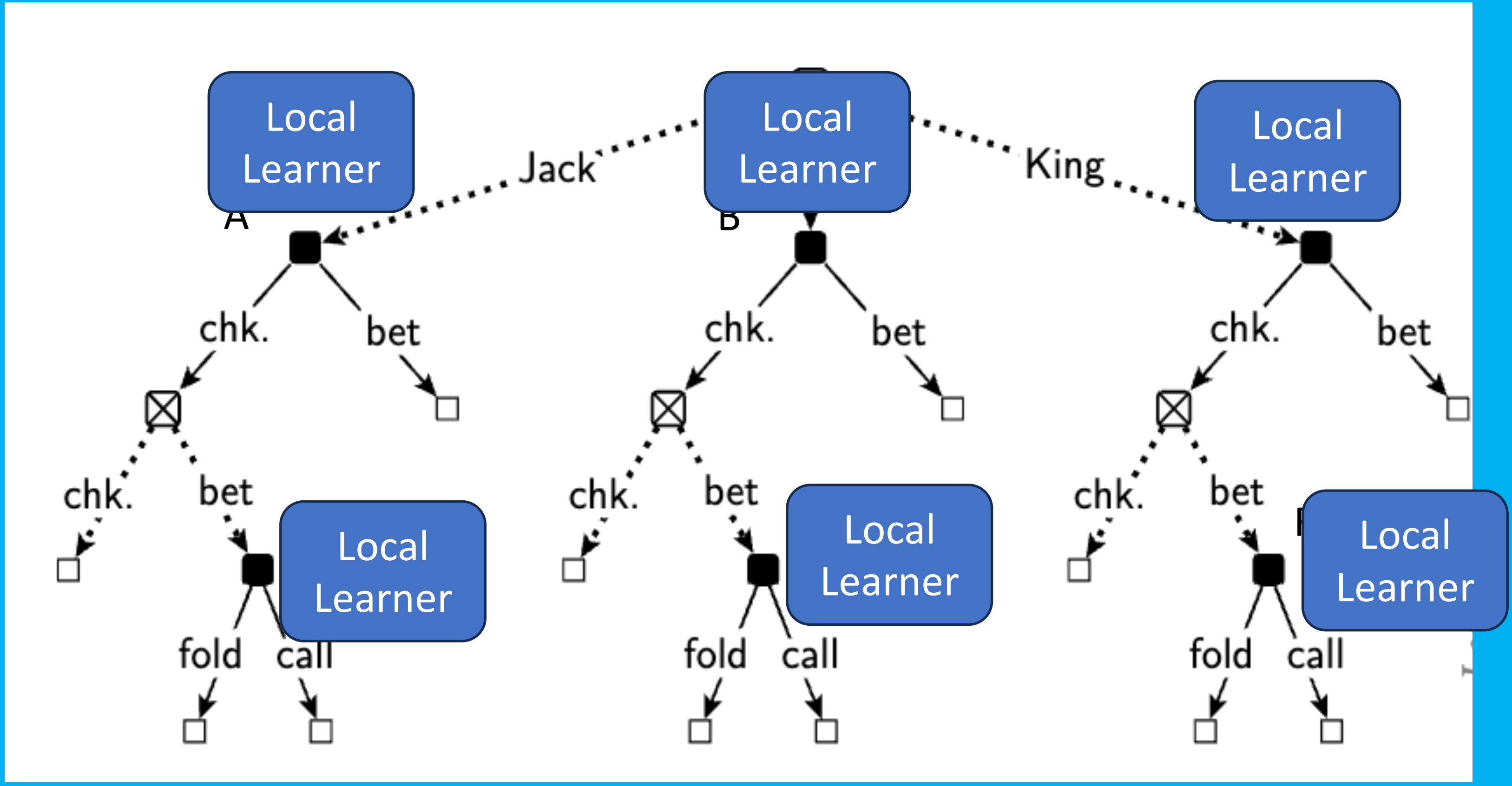
🚨

Papercut Alert™

CFR updates strategies in *behavioral* form…

…but is a no-external-regret algorithm for
sequence-form strategies

Big Picture Idea:



Each local learner is responsible for refining the **behavior** at their decision point

Can locally use regret matching, multiplicative weights update, …

Local Training Feedback

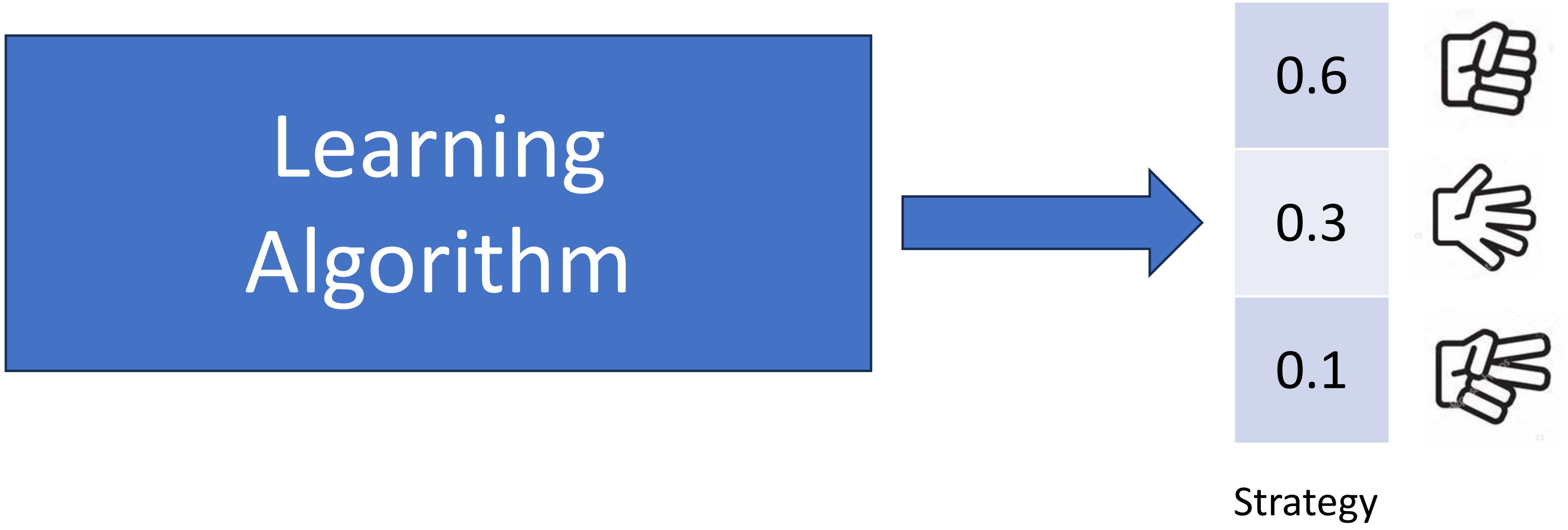
Each local learner receives as feedback what is known as a **counterfactual utility vector**

This is constructed starting from the $u^{(t)}$

Recall: Learning in Normal-Form Games

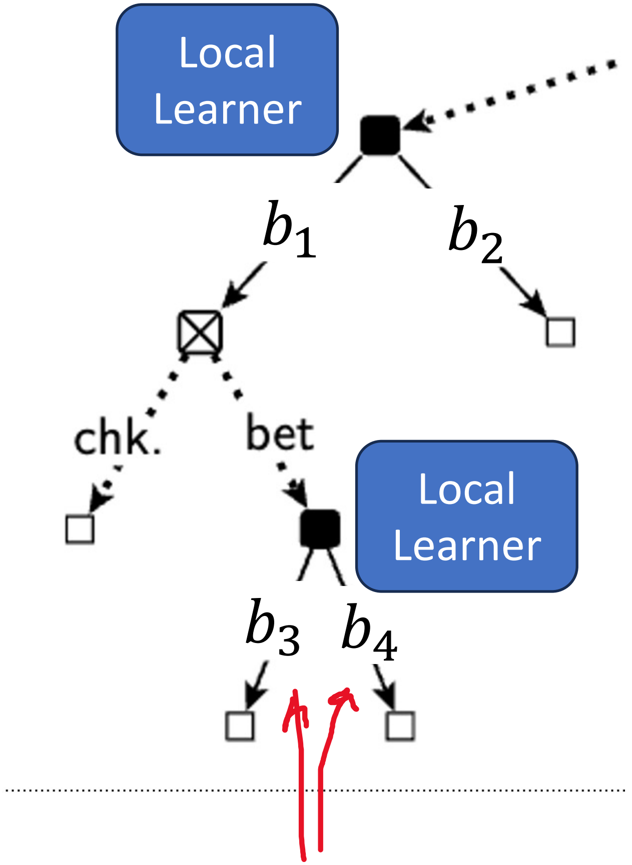
Learning
Algorithm

Recall: Learning in Normal-Form Games



Recall: Learning in Normal-Form Games

Recall: Learning in Normal-Form Games

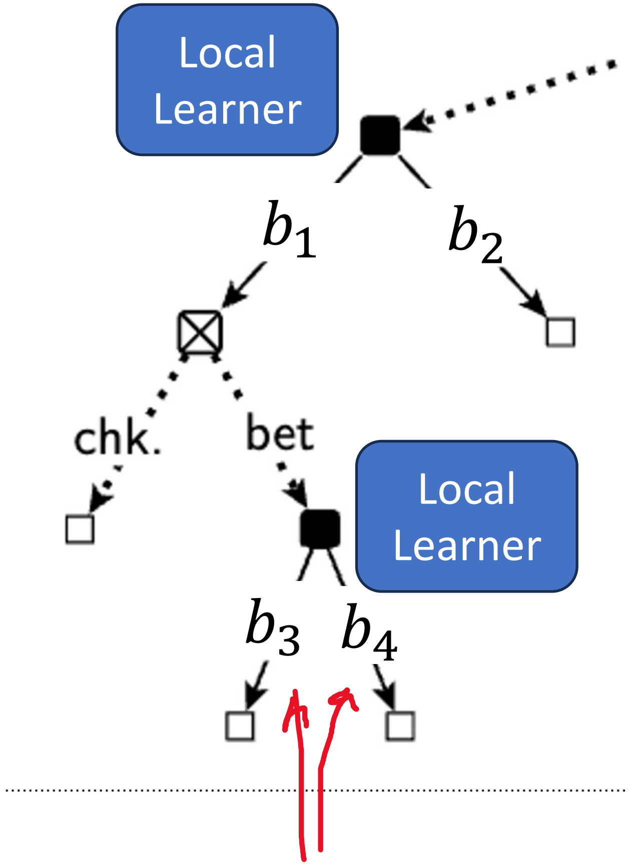


Probabilities of actions chosen by local learners

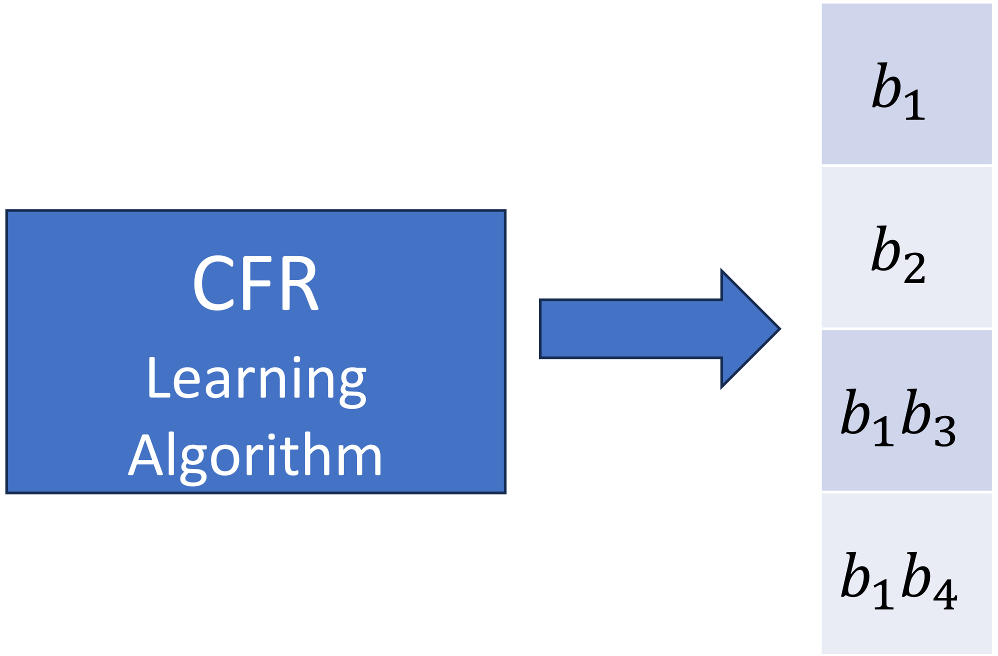
CFR

Learning Algorithm

Recall: Learning in Normal-Form Games

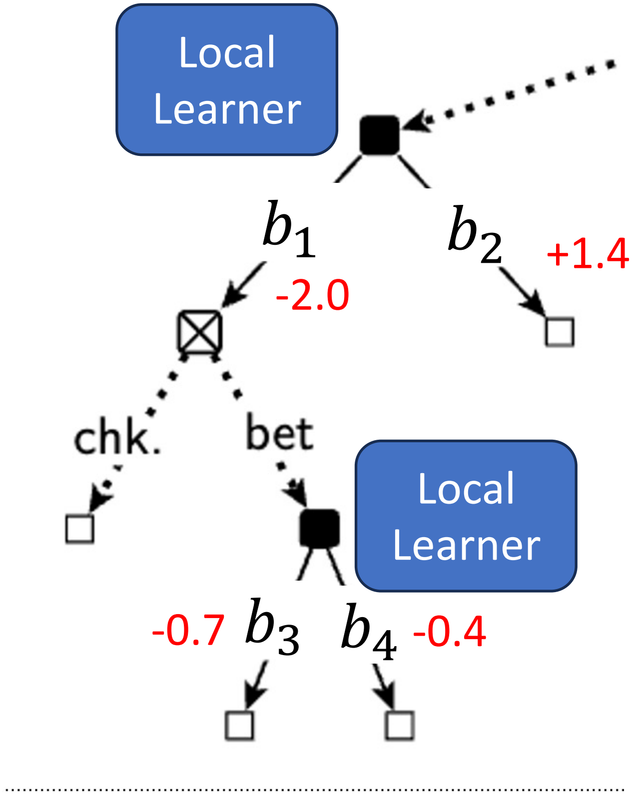


Probabilities of actions chosen by local learners



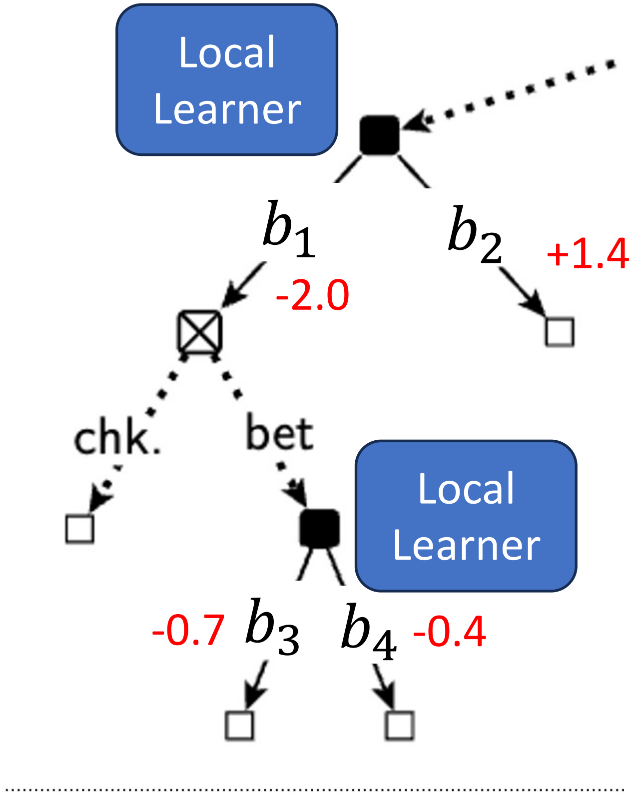
Strategy
(in sequence form)

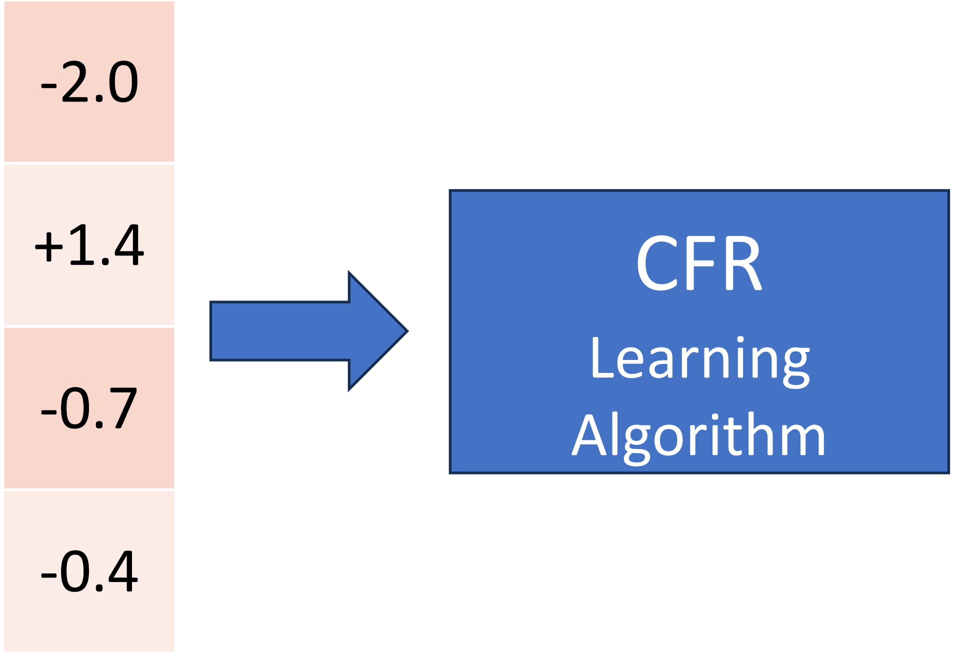
Recall: Learning in Normal-Form Games



CFR
Learning Algorithm

Recall: Learning in Normal-Form Games

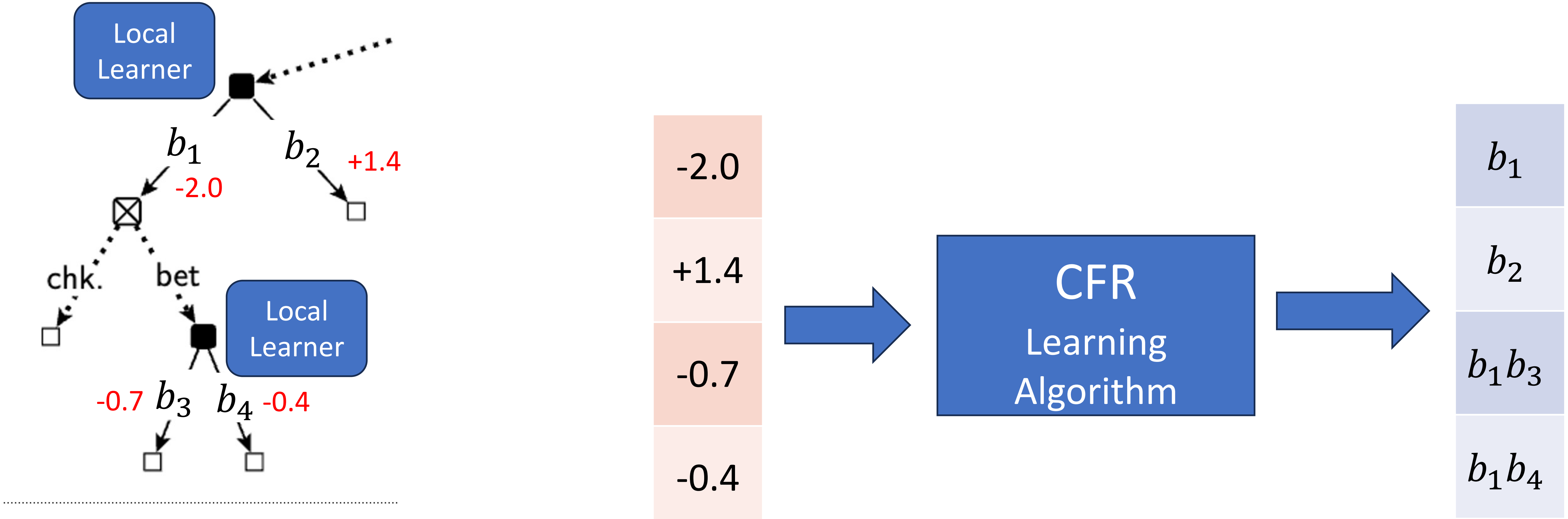




Utility vector
(for sequence-form
strategy)

Strategy

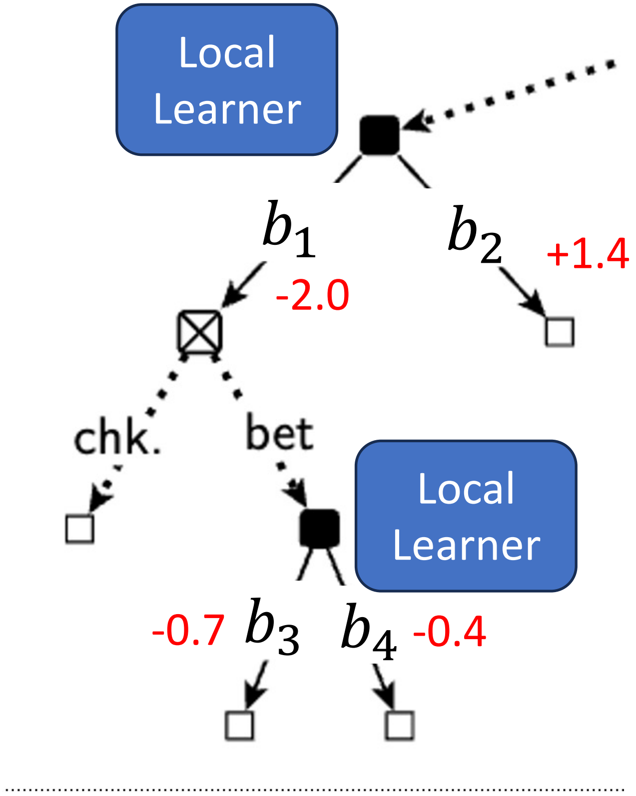
Recall: Learning in Normal-Form Games

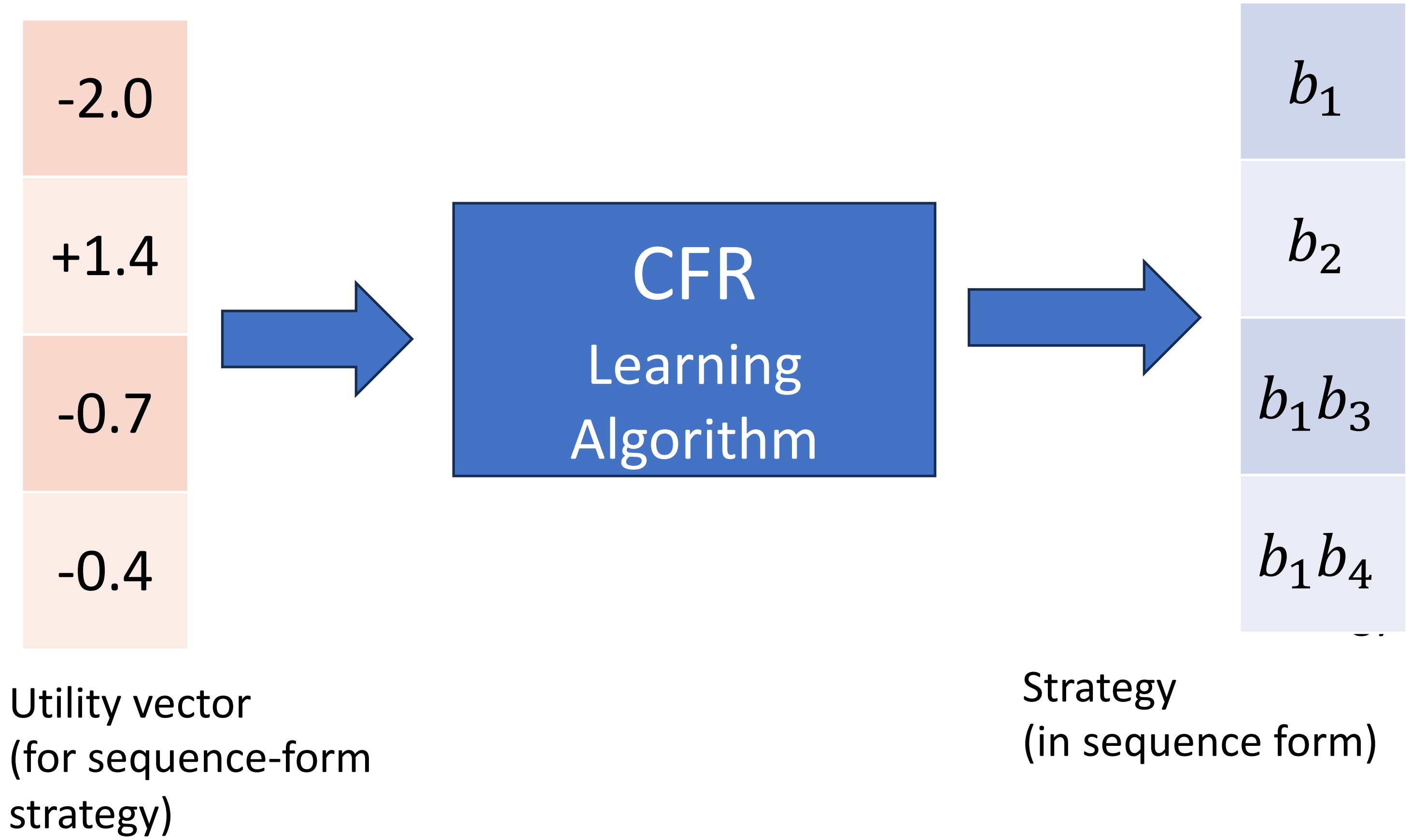


Utility vector
(for sequence-form
strategy)

Strategy
(in sequence form)

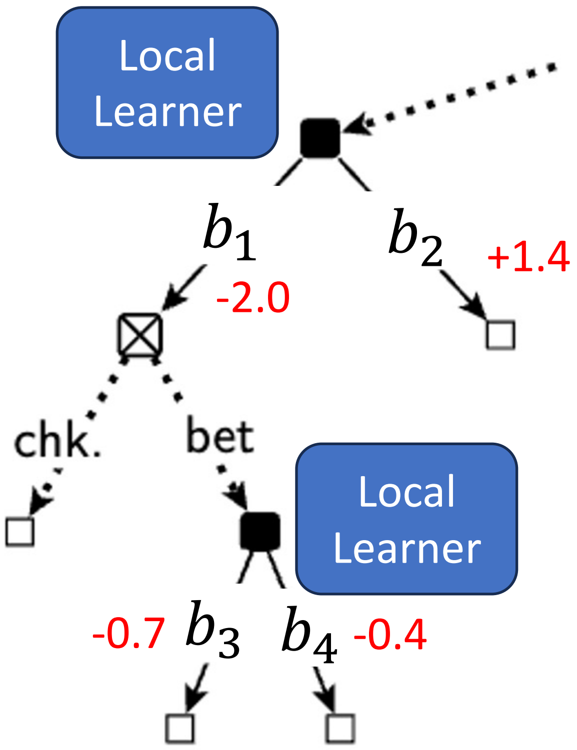
Recall: Learning in Normal-Form Games



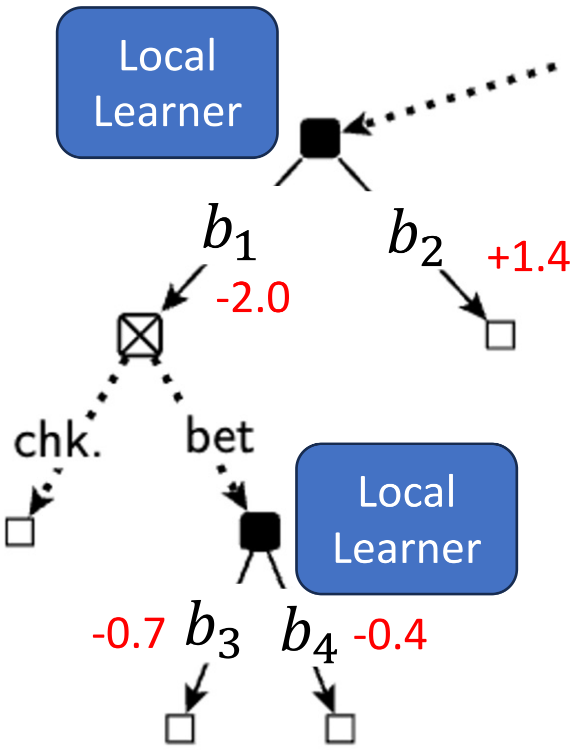


Main question: what utility to pass to the local learners?

Counterfactual Utilities

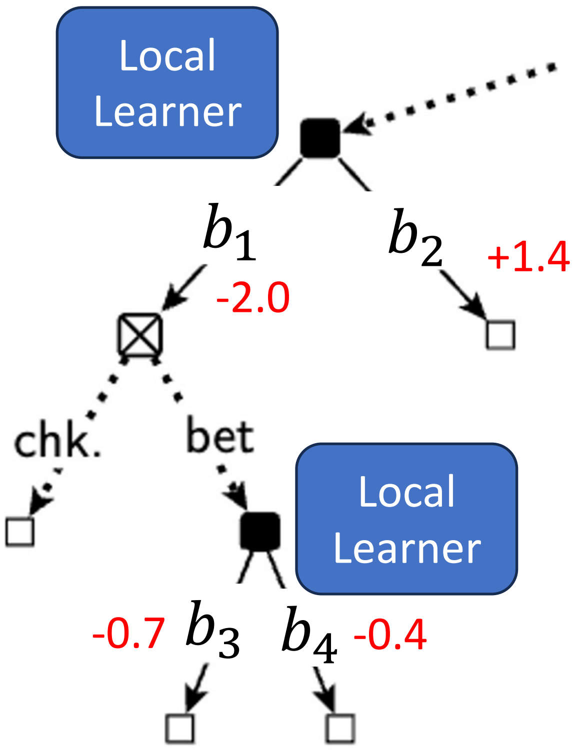


Counterfactual Utilities



Give to each local learner the **expected utility in the subtree** rooted at each action:

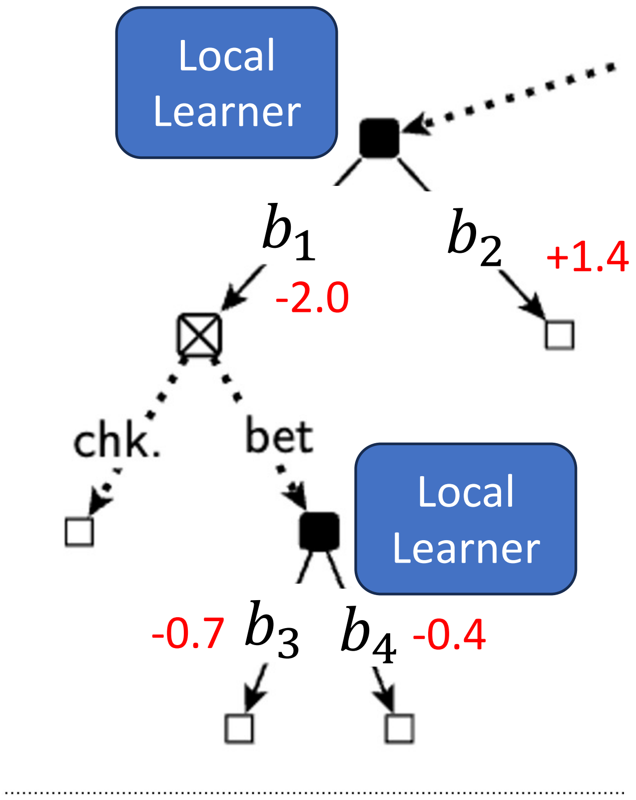
Counterfactual Utilities



Give to each local learner the **expected utility in the subtree** rooted at each action:

$$\widehat{u_3} = -0.7$$
$$\widehat{u_4} = -0.4$$

Counterfactual Utilities



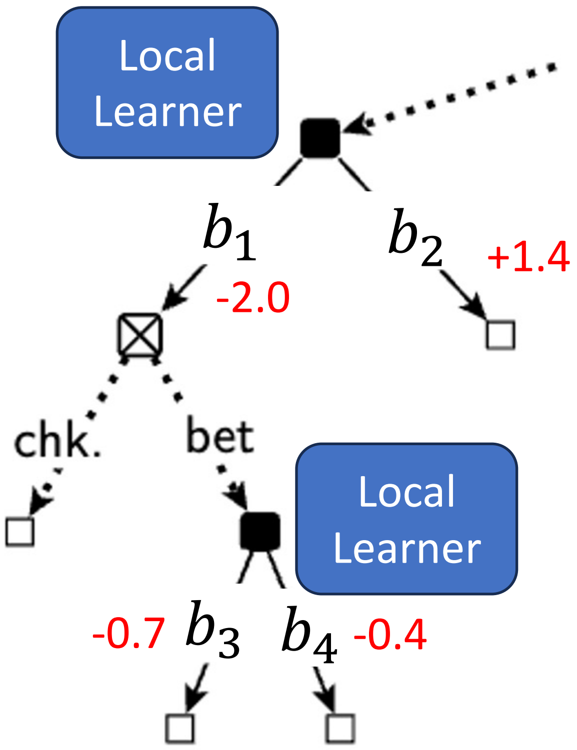
Give to each local learner the **expected utility in the subtree** rooted at each action:

$$\widehat{u_3} = -0.7$$

$$\widehat{u_4} = -0.4$$

$$\widehat{u_2} = +1.4$$

Counterfactual Utilities



Give to each local learner the **expected utility in the subtree** rooted at each action:

$$\widehat{u_3} = -0.7$$

$$\widehat{u_4} = -0.4$$

$$\widehat{u_2} = +1.4$$

$$\widehat{u_1} = -2.0 + b_3 \cdot (-0.7) + b_4 \cdot (-0.4)$$

Regret bound

- Theorem: the regret cumulated by CFR can be bounded as

$$R_{CFR}^{(T)} \leq \sum_{j} \max\left\{0, R_j^{(T)}\right\}$$

Decision points

Local regret cumulated by learner at j

Regret bound

- Theorem: the regret cumulated by CFR can be bounded as

$$R_{CFR}^{(T)} \leq \sum_{j} \max\left\{0, R_j^{(T)}\right\}$$

Decision points

Local regret cumulated by learner at j

- **Therefore**: if the local regret minimizers all have regret $O(\sqrt{T})$, then CFR has regret $O(\sqrt{T})$ (where the O hides game-dependent constants)

Regret bound

- Theorem: the regret cumulated by CFR can be bounded as

$$R_{CFR}^{(T)} \le \sum_{j} \max\left\{0, R_j^{(T)}\right\}$$

Decision points

Local regret cumulated by learner at j

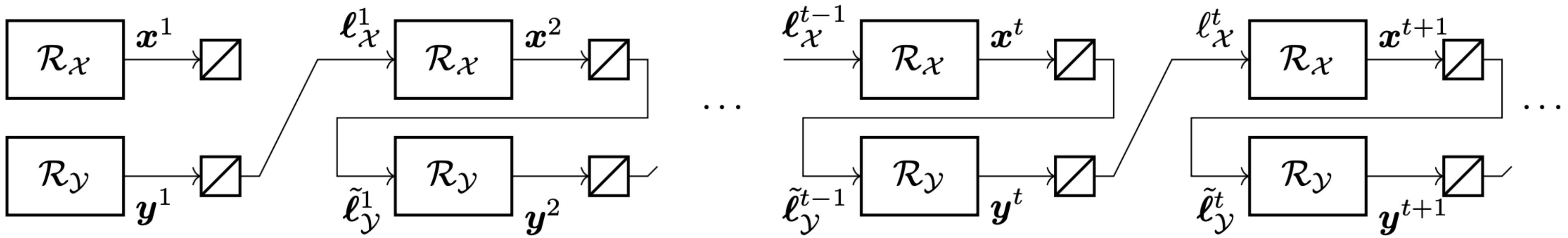
- **Therefore**: if the local regret minimizers all have regret $O(\sqrt{T})$, then CFR has regret $O(\sqrt{T})$ (where the O hides game-dependent constants)

Therefore: if both players in a zero-sum extensive-form game play according to CFR, the average strategy converges to Nash equilibrium at rate $O(1/\sqrt{T})$

Further pushing performance

CFR+: CFR with the following settings:

- Regret Matching+ at each decision point (see Lecture 5)
- Use alternation



- When computing average strategy, weigh strategy at time t by t:

$$\bar{x}^{(T)} \propto \sum^{T} t \cdot x^{(t)}$$

Advantages of CFR

Compared to linear programming, CFR is significantly more scalable

...On the other hand, it converges to equilibrium at a 1/sqrt(T) rate, rather than e^(-T)

CFR uses an approach local to each decision point (easier to parallelize, warm-start, etc.)

- [Brown & Sandholm, Reduced Space and Faster Convergence in Imperfect-Information Games via Pruning. ICML-17]
- [Brown & Sandholm, Strategy-based warm starting for regret minimization in games, AAAI 2016]
- ...

CFR Lends itself to further extensions

- Using utility estimators
  - Similar idea as stochastic gradient descent vs gradient descent
  - Instead of exactly computing the green numbers (gradients of the utility function), we use cheap unbiased estimators
  - Popular estimator: sample a trajectory in the game tree and use importance sampling
  - "**Monte Carlo CFR**" [Monte Carlo Sampling for Regret Minimization in Extensive Games; Lanctot, Waugh, Zinkevich, Bowling NIPS 2009]
  - Even better algorithm, **ESCHER**, does not use importance sampling [McAleer, Farina, Lanctot & Sandholm *ICLR*-23]

FTRL in Extensive-Form Games

Follow-the-Regularized-Leader

$$x^{(t)} = \arg\max_{x \in Q} \langle U^{(t)}, x \rangle - \frac{1}{\eta}\psi(x)$$

Depending on the choice of strongly convex regularizer ψ, solving the step above might be impractical

Follow-the-Regularized-Leader

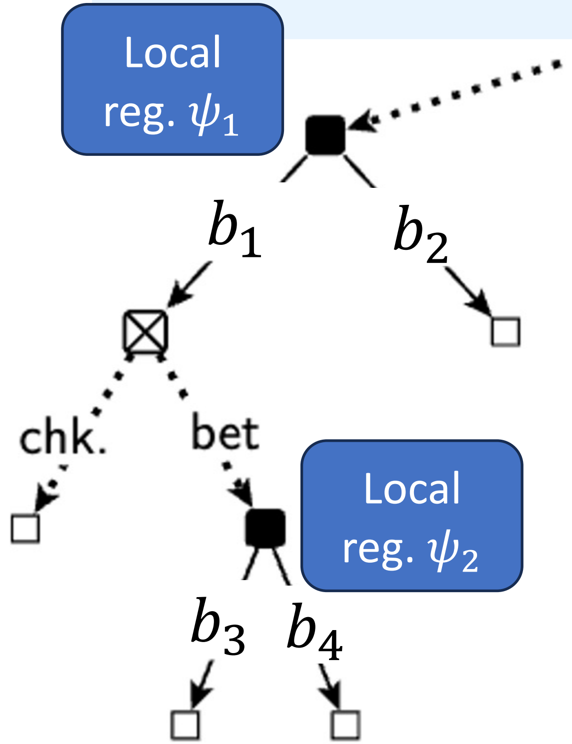
$$x^{(t)} = \arg\max_{x \in Q} \langle U^{(t)}, x \rangle - \frac{1}{\eta} \psi(x)$$

Depending on the choice of strongly convex regularizer ψ, solving the step above might be impractical

Example: if ψ is the squared Euclidean distance, then the solution can be found in polynomial time but it is complicated and expensive in practice!

Efficient Regularizers

Idea: construct regularizers that mimic the structure of the tree-form decision problem

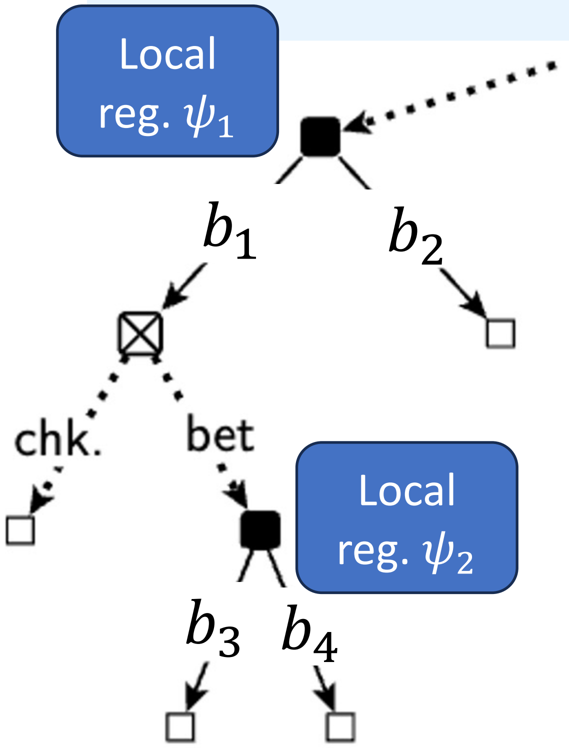


Strategy
(in sequence form)

Then is strongly convex, and the solution to the FTRL problem can be computed in a bottom-up fashion

Efficient Regularizers

Idea: construct regularizers that mimic the structure of the tree-form decision problem



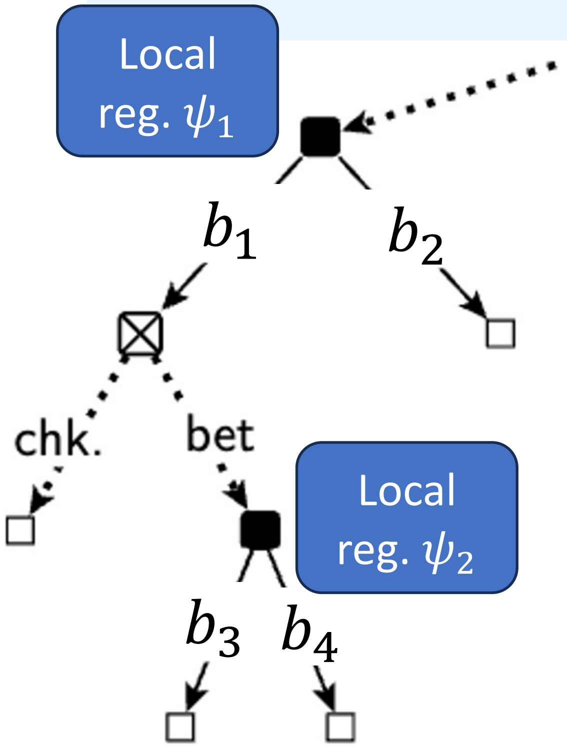
$$x = \begin{bmatrix} b_1 \\ b_2 \\ b_1 b_3 \\ b_1 b_4 \end{bmatrix}$$

Strategy
(in sequence form)

Then is strongly convex, and the solution to the FTRL problem can be computed in a bottom-up fashion

Efficient Regularizers

Idea: construct regularizers that mimic the structure of the tree-form decision problem



$$x = \begin{bmatrix} b_1 \\ b_2 \\ b_1 b_3 \\ b_1 b_4 \end{bmatrix}$$

Strategy
(in sequence form)

Dilated regularizers

$$\psi(x) \coloneqq \psi_1(b_1, b_2) + b_1 \cdot \psi_2(b_3, b_4)$$

Where f_1 and f_2 are local strongly convex regularizers (e.g., negative entropy)

Then is strongly convex, and the solution to the FTRL problem can be computed in a bottom-up fashion

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Overall: kernelization gives better **theoretical bounds** on the regret

CFR gives better empirical performance

For large games, learning-based methods (+ function approximation) are today the scalable state of the art