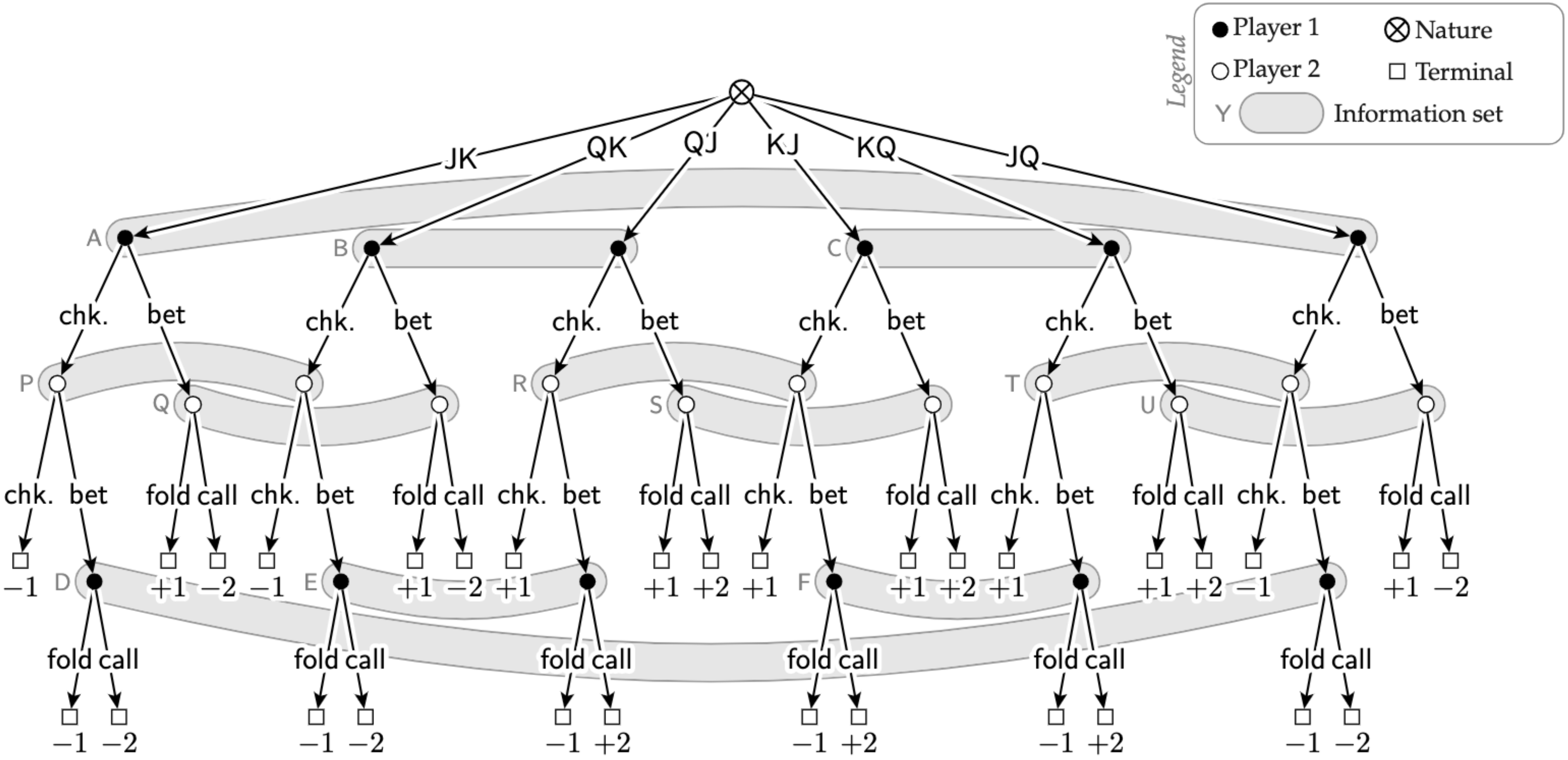
6.S890:
Topics in Multiagent Learning

Lecture 13 – Prof. Farina

Computation of Nash equilibria in two-player zero-sum extensive-form games

Fall 2023

Recall: Extensive-form games

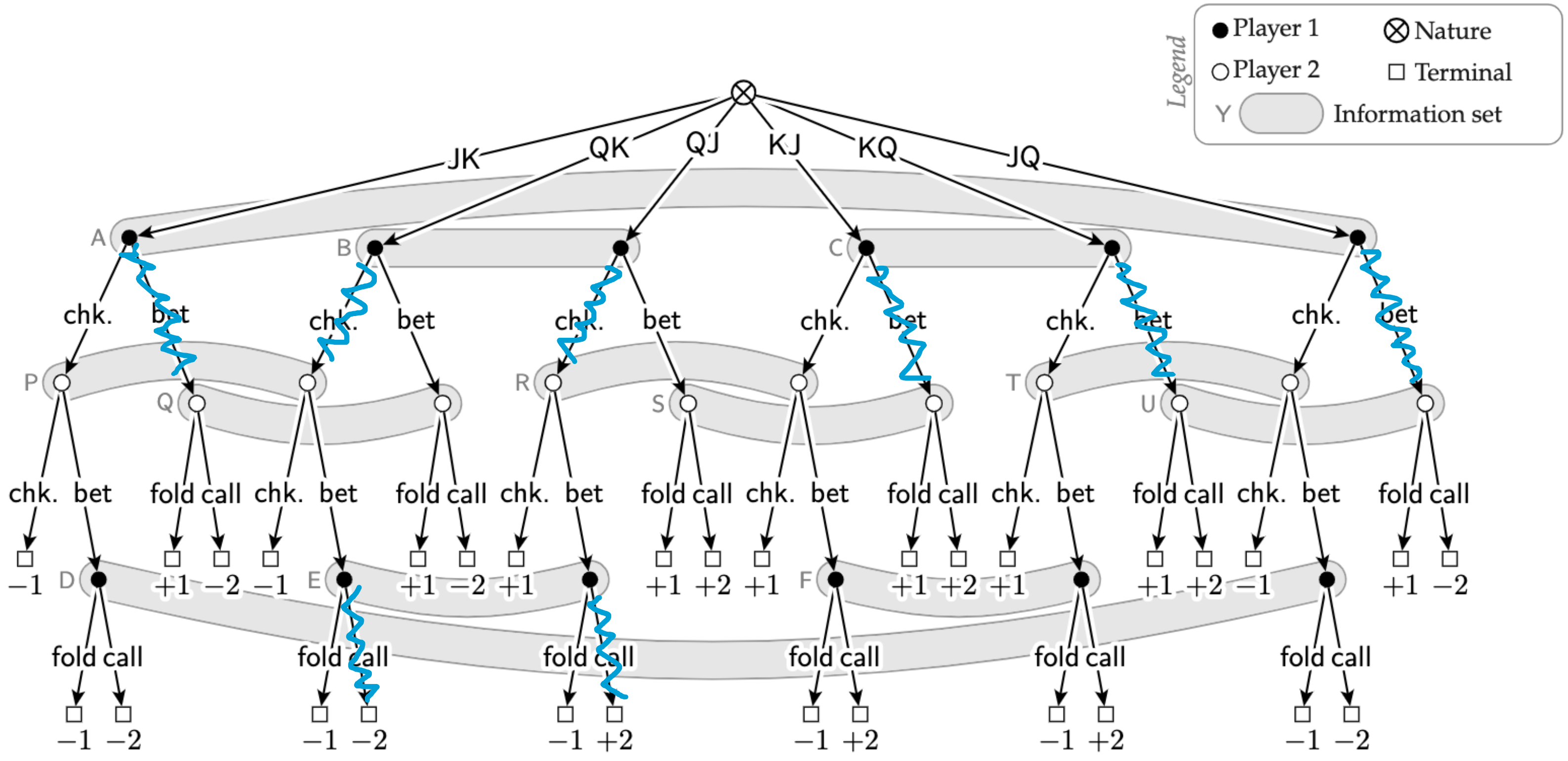


Recall: Strategies

	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 $b \in \times_j \Delta(A_j)$	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.**

Recall: Strategic Form

Idea: Strategy = randomize a deterministic contingency plan

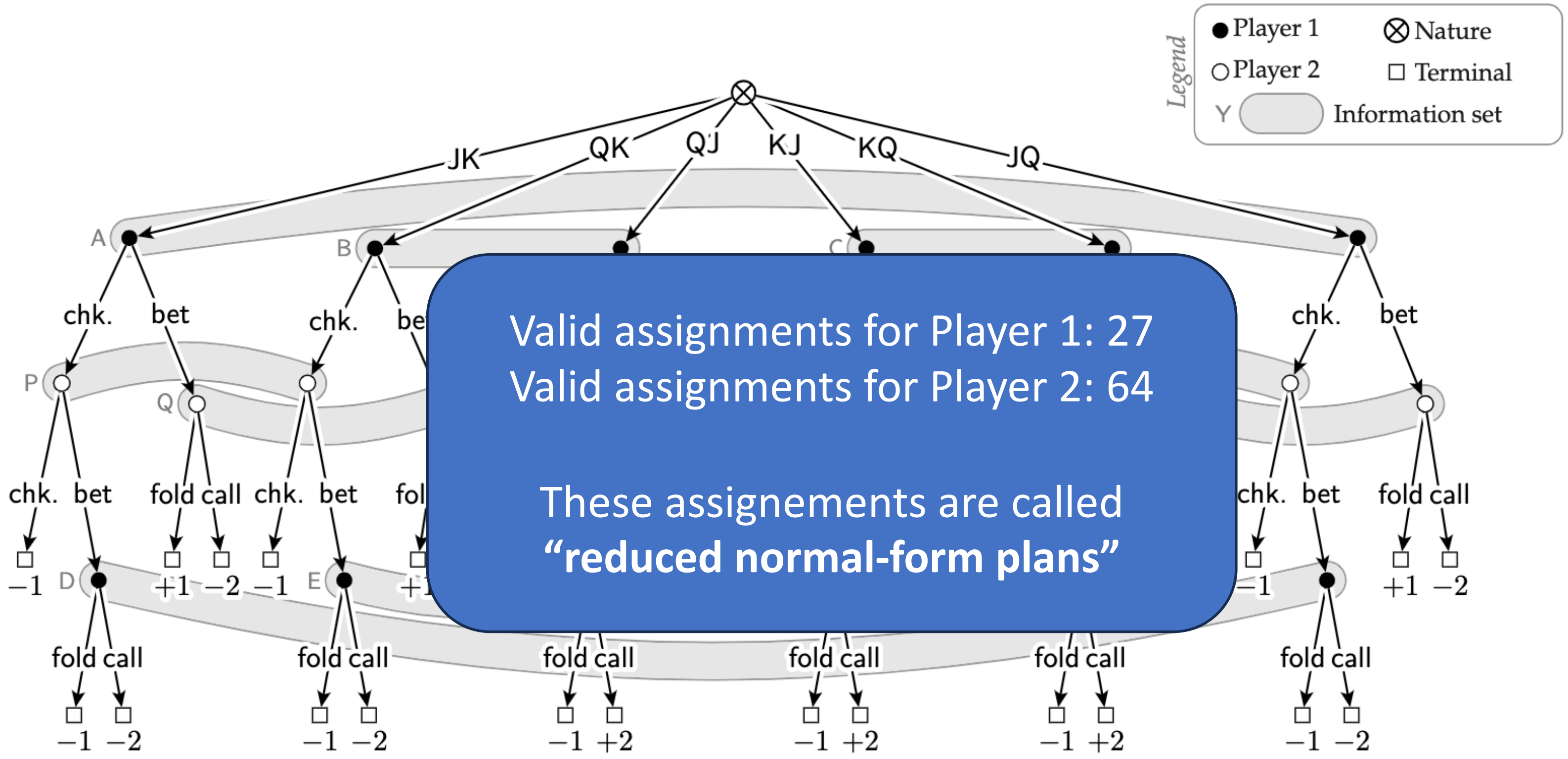


Each player constructs a list of all possible assignments of actions at each information set

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

Recall: Strategic Form

Idea: Strategy = randomize a deterministic contingency plan



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

These assignements 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)

Recall: 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, …

Recall: 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:

$x^T Ay$

Payoff matrix on the left

the 27 plans of Player 1

Distribution over the 64 plans of Player 2

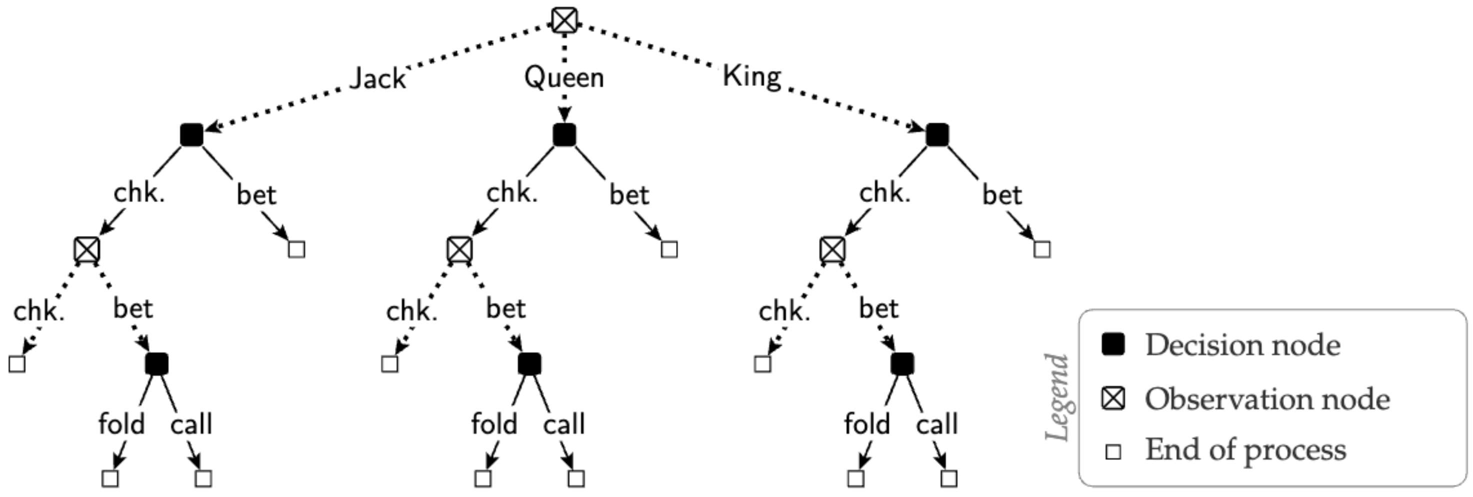
You can use any technique for normal-form games: learning, linear programming, …

Recall: Behavioral Strategies

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

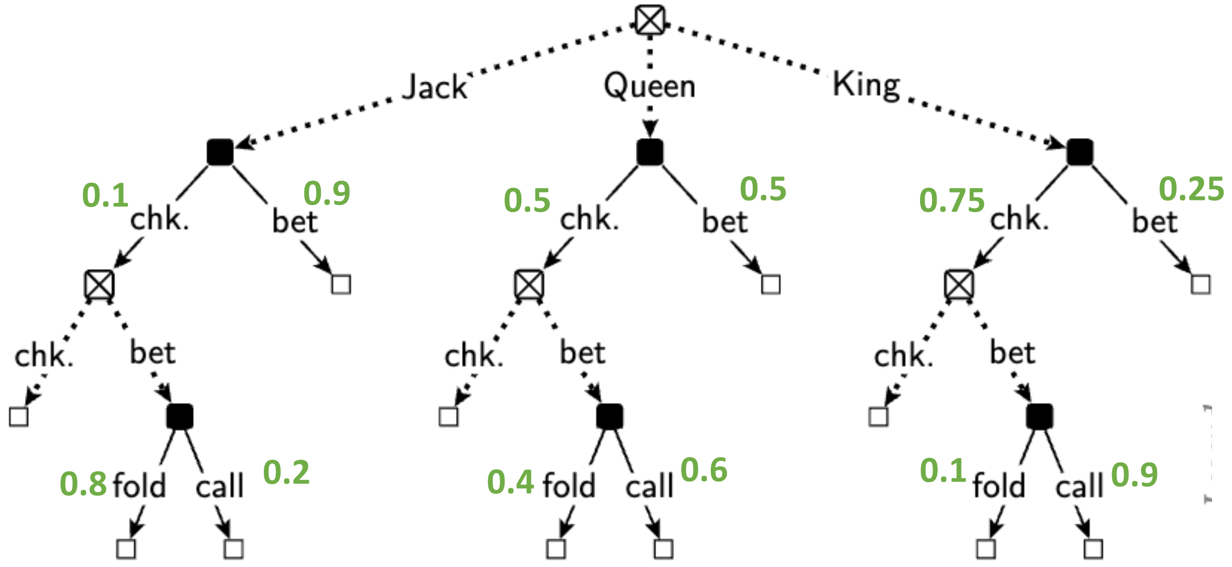
Information set

We found it convenient to take the point of view of a single player: face decisions and observations



Recall: Behavioral strategies

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



✓ Set of strategies is convex

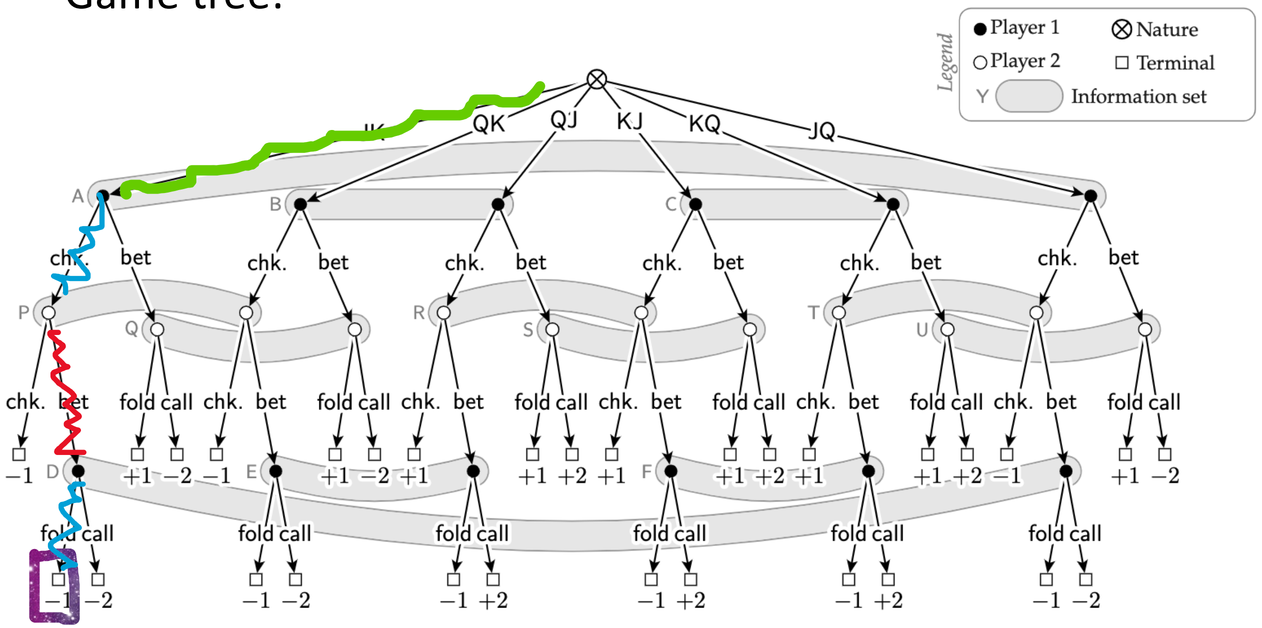
✗ Expected utility is **not** linear in this representation

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

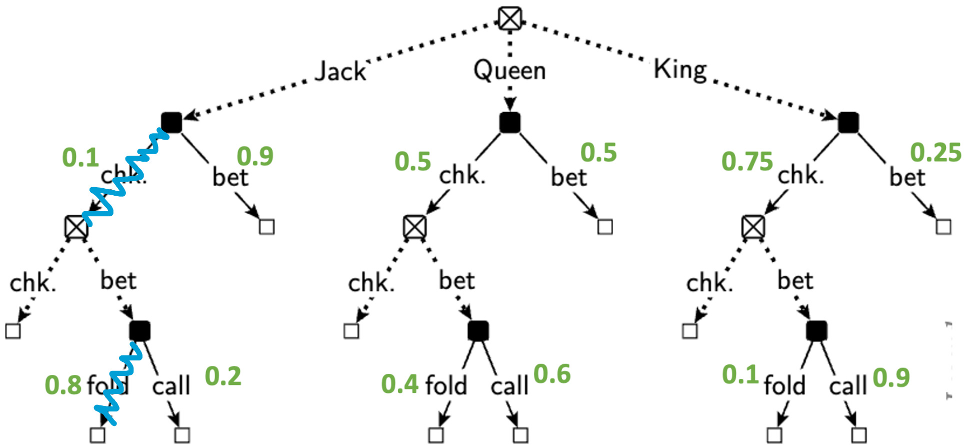
Products = non-convexity 😪

Recall: Expected Utility

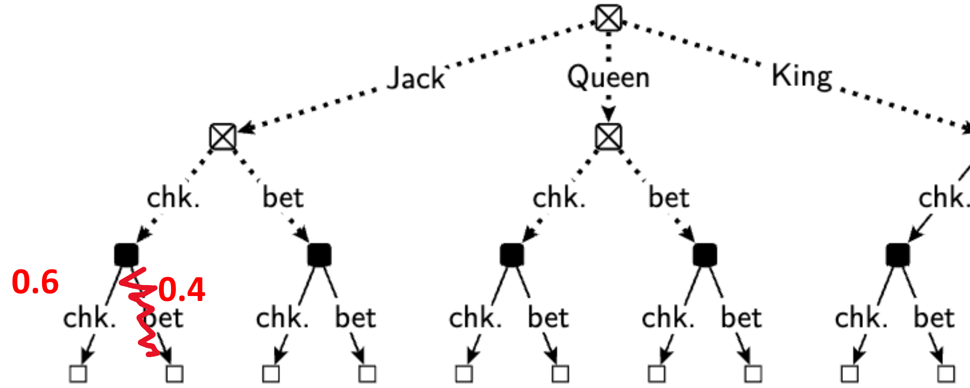
Game tree:



Decision problem and behavioral strategy of Player 1



Decision problem and behavioral strategy of Player 2

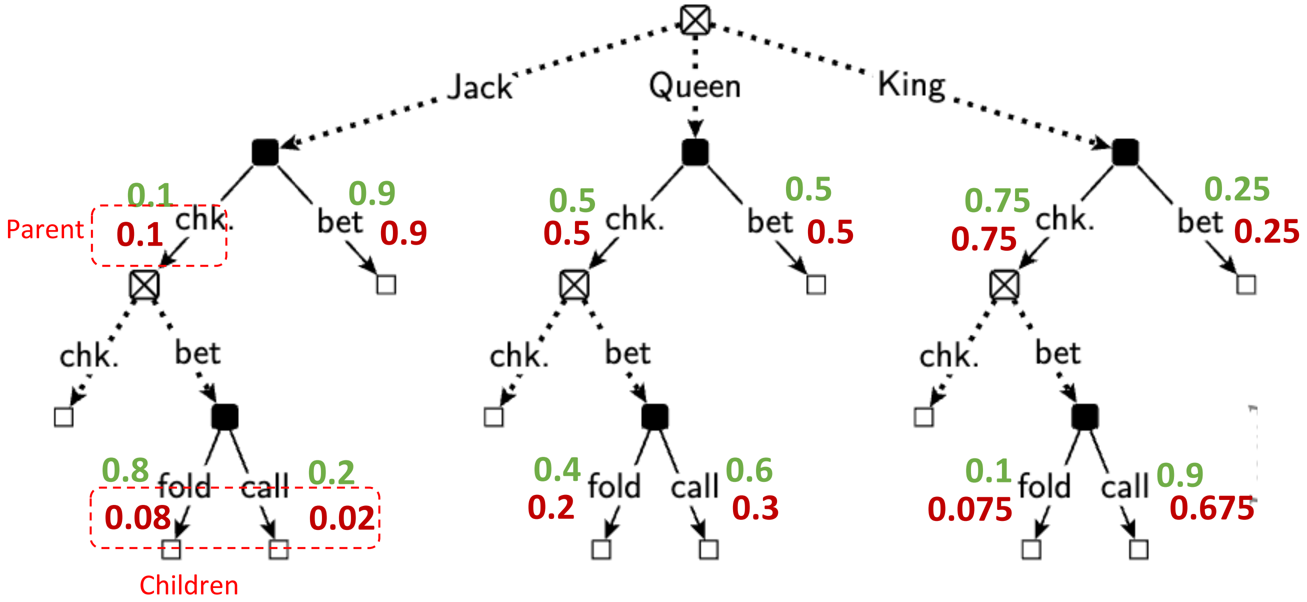


Prob of reaching this terminal state: 1/6 (Nature) x 0.1 (Pl1) x 0.4 (Pl2) x 0.8 (Pl1)

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

“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

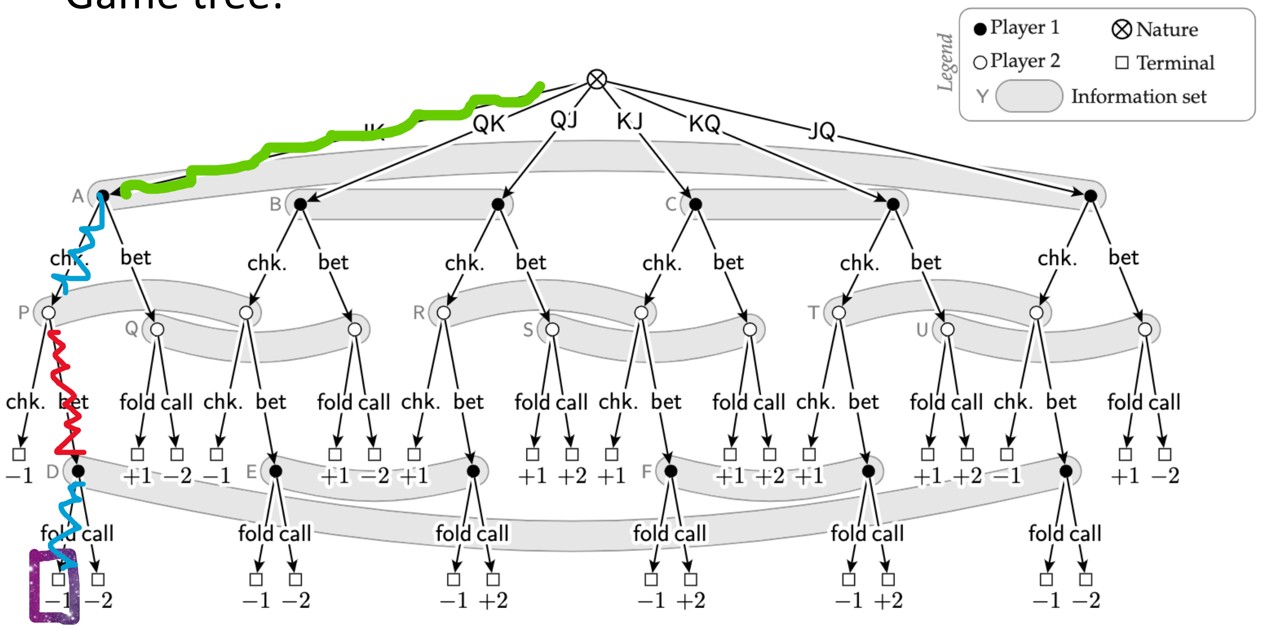
Consistency constraints

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

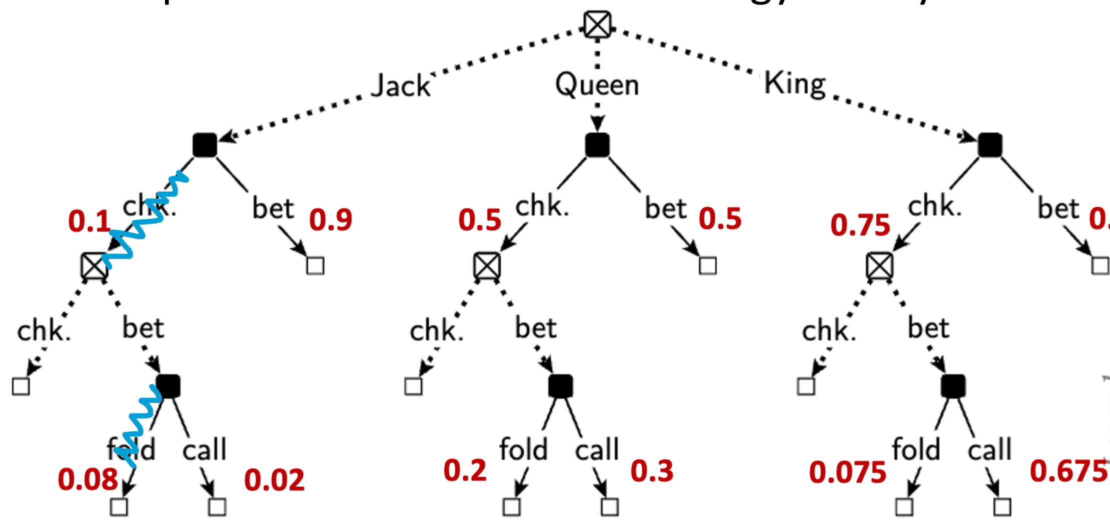
Since sequence-form strategies already automatically encode products of probabilities on paths, expected utility is linear in this strategy representation!

Recall: Expected Utility

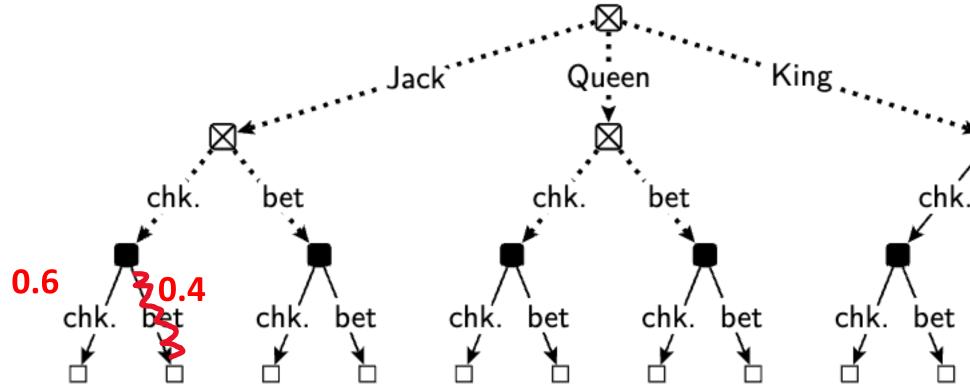
Game tree:



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)

Single variable from strategy vector! Nonlinearity is gone

Recall: Equilibrium Computation

BEFORE: Reduced–normal form

Nash equilibrium in Kuhn poker:

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

Distribution over the 27 plans of Player 1

Distribution over the 64 plans of Player 2

Payoff matrix in reduced normal form

Scale exponentially with tree size

You can use any tec
learning, linear

NOW: Sequence form

Nash equilibrium in Kuhn poker:

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

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 learning

Nash equilibrium
(two-player zero-sum):

$$\max_{x \in Q_1} \min_{y \in Q_2} x^T A y$$

Sequence-form polytope of player 1 (dimension 12)

Sequence-form polytope of player 2 (dimension 12)

Sequence-form payoff matrix for player 1

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

Let's code up a solver together!

Two Approaches to Solve The Max-Min Problem

Approach 1: Linear Programming

Approach 2: Learning

For sequence-form polytopes in particular: Counterfactual Regret Minimization (CFR)

Nash equilibrium
(two-player zero-sum):

$$\max_{x \in Q_1} \min_{y \in Q_2} x^T A y$$

Sequence-form polytope of player 1 (dimension 12)

Sequence-form polytope of player 2 (dimension 12)

Sequence-form payoff matrix for player 1

Why / How can this be converted into a linear program?

Linear Program Formulation

$$\max_{x \in Q_1} \min_{y \in Q_2} x^T A y \xrightarrow{1} \begin{cases} \max & \boxed{\begin{cases} \min & x^T A y \\ \text{s.t.} & y \in Q_2 \end{cases}} \\ \text{s.t.} & x \in Q_1 \end{cases} \xrightarrow{2} \begin{cases} \max & \boxed{\begin{cases} \min & x^T A y \\ \text{s.t.} & F_2 y = f_2 \\ & y \geq 0 \end{cases}} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

Nested optimization problem. The inner problem is linear

Remember: y is from the sequence-form polytope Q_2

- Root decision points have mass 1
- Probability mass is conserved
- $y \geq 0$

Compactly:

$$Q_2 = \begin{cases} F_2 y = f_2 \\ y \geq 0 \end{cases}$$

$$Q_1 = \begin{cases} F_1 x = f_1 \\ x \geq 0 \end{cases} \qquad Q_2 = \begin{cases} F_2 y = f_2 \\ y \geq 0 \end{cases}$$

Linear Program Formulation

$$\max_{x\in Q_1} \min_{y \in Q_2} x^T A y$$

1

$$\begin{cases} \max & \boxed{\begin{cases} \min & x^T A y \\ \text{s.t.} & y \in Q_2 \end{cases}} \\ \text{s.t.} & x \in Q_1 \end{cases}$$

2

$$\begin{cases} \max & \boxed{\begin{cases} \min & x^T A y \\ \text{s.t.} & F_2 y = f_2 \\ & y \geq 0 \end{cases}} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

3

Dualize!

$$\begin{cases} \max & \boxed{\begin{cases} \max & f_2 v \\ \text{s.t.} & F_2^T v \leq A^T x \\ & v \in \mathbb{R} \end{cases}} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

4

$$\begin{cases} \max & f_2 v \\ \text{s.t.} & F_1 x = f_1 \\ & F_2^T v \leq A^T x \\ & x \geq 0 \\ & v \in \mathbb{R} \end{cases}$$

Single linear program!

$$Q_1 = \begin{cases} F_1 x = f_1 \\ x \geq 0 \end{cases} \qquad Q_2 = \begin{cases} F_2 y = f_2 \\ y \geq 0 \end{cases}$$

...tion

What do we need to implement this?

1. From the game tree, extract F_1, F_2, f_1, f_2, and A
2. Code up the linear program
3. Profit!

2

$$\begin{cases} \max & \boxed{\begin{cases} \min & x^T A y \\ \text{s.t.} & F_2 y = f_2 \\ & y \geq 0 \end{cases}} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

3

Dualize!

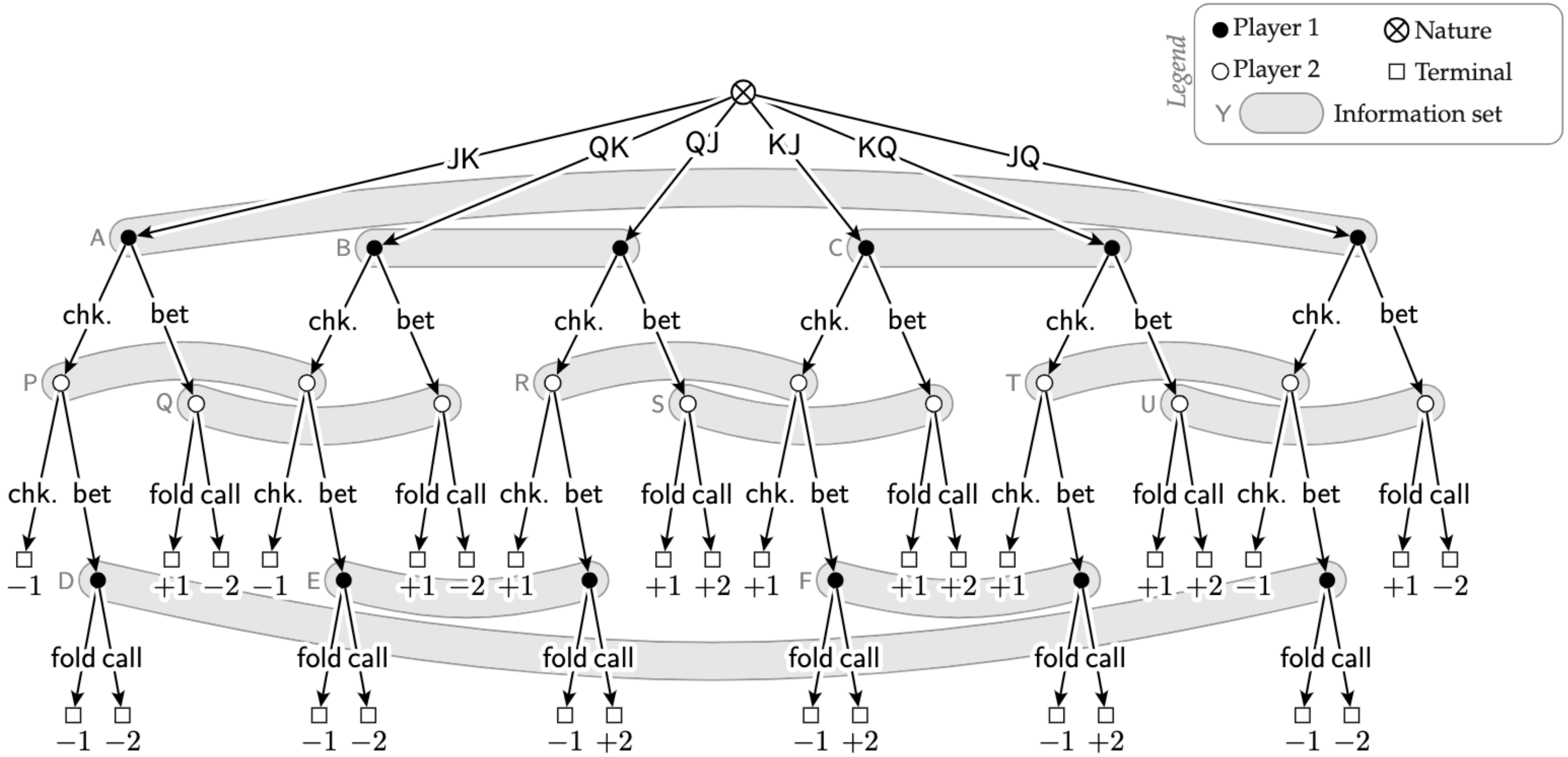
$$\begin{cases} \max & \boxed{\begin{cases} \max & f_2 v \\ \text{s.t.} & F_2^T v \leq A^T x \\ & v \in \mathbb{R} \end{cases}} \\ \text{s.t.} & F_1 x = f_1 \\ & x \geq 0 \end{cases}$$

4

$$\begin{cases} \max & f_2 v \\ \text{s.t.} & F_1 x = f_1 \\ & F_2^T v \leq A^T x \\ & x \geq 0 \\ & v \in \mathbb{R} \end{cases}$$

Single linear program!

How to construct F_1, f_1, F_2, f_2?



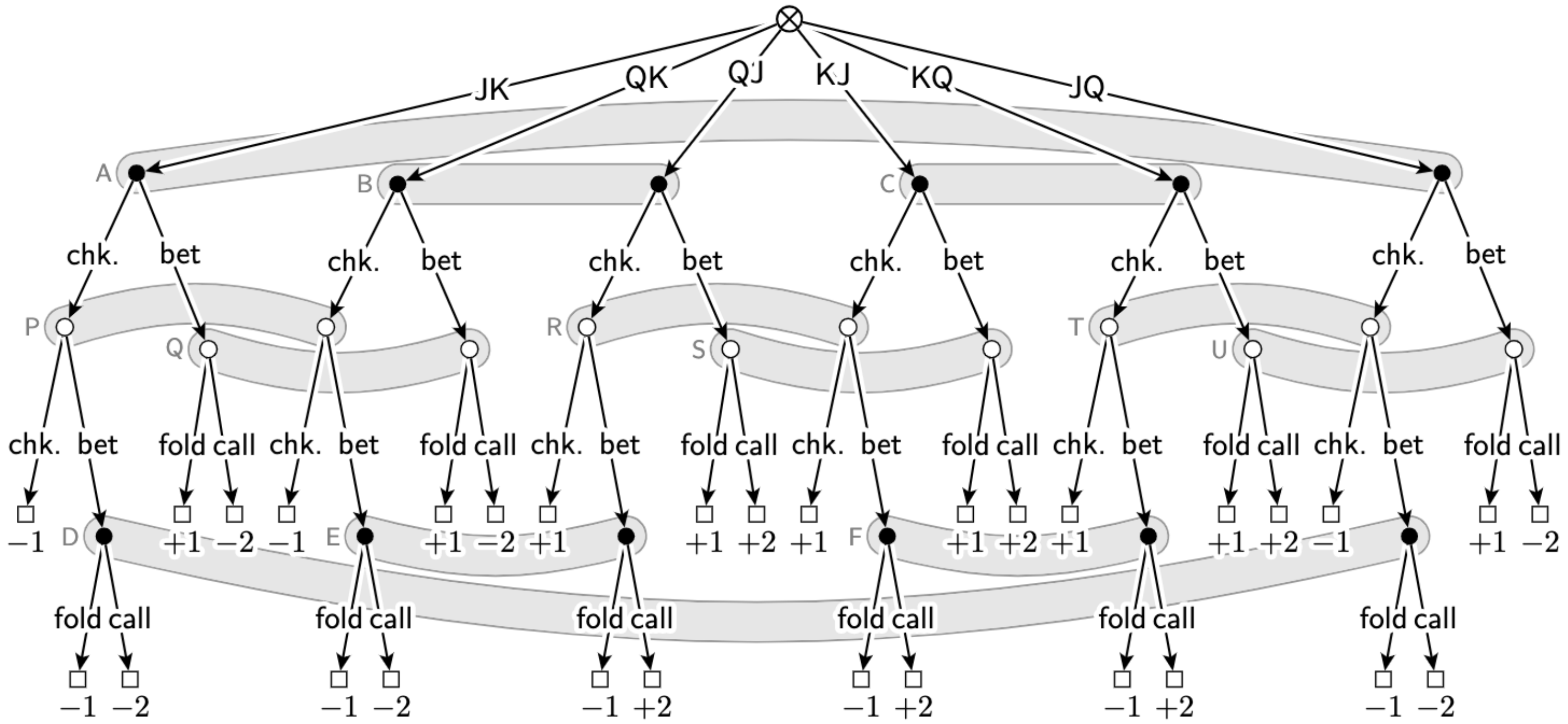
In sequence form, we have one variable per action at each decision point (information set)

Matrices F_1, f_1, F_2, f_2 encode the probability flow conservation constraints

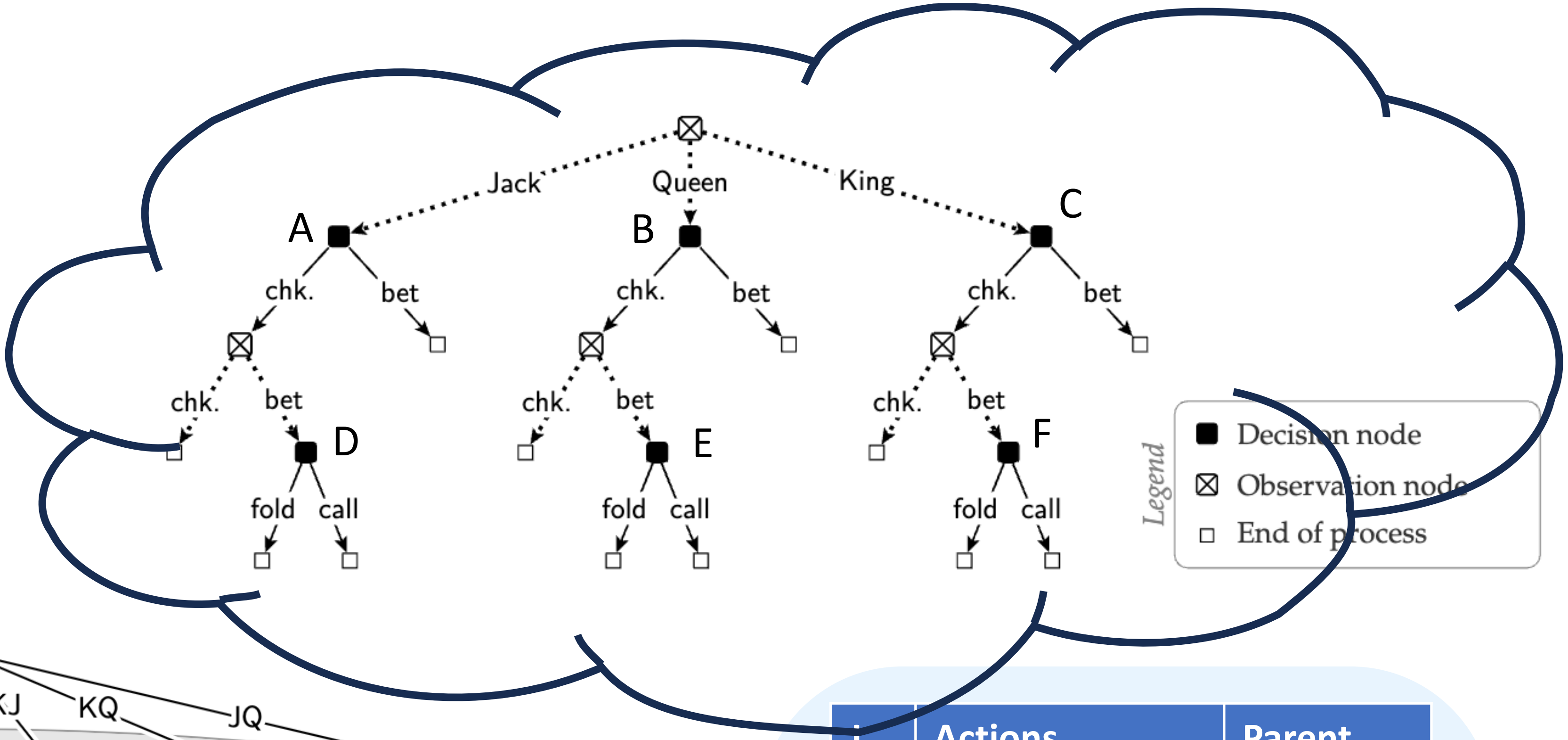
Step 1: Construct each player's tree-form decision process

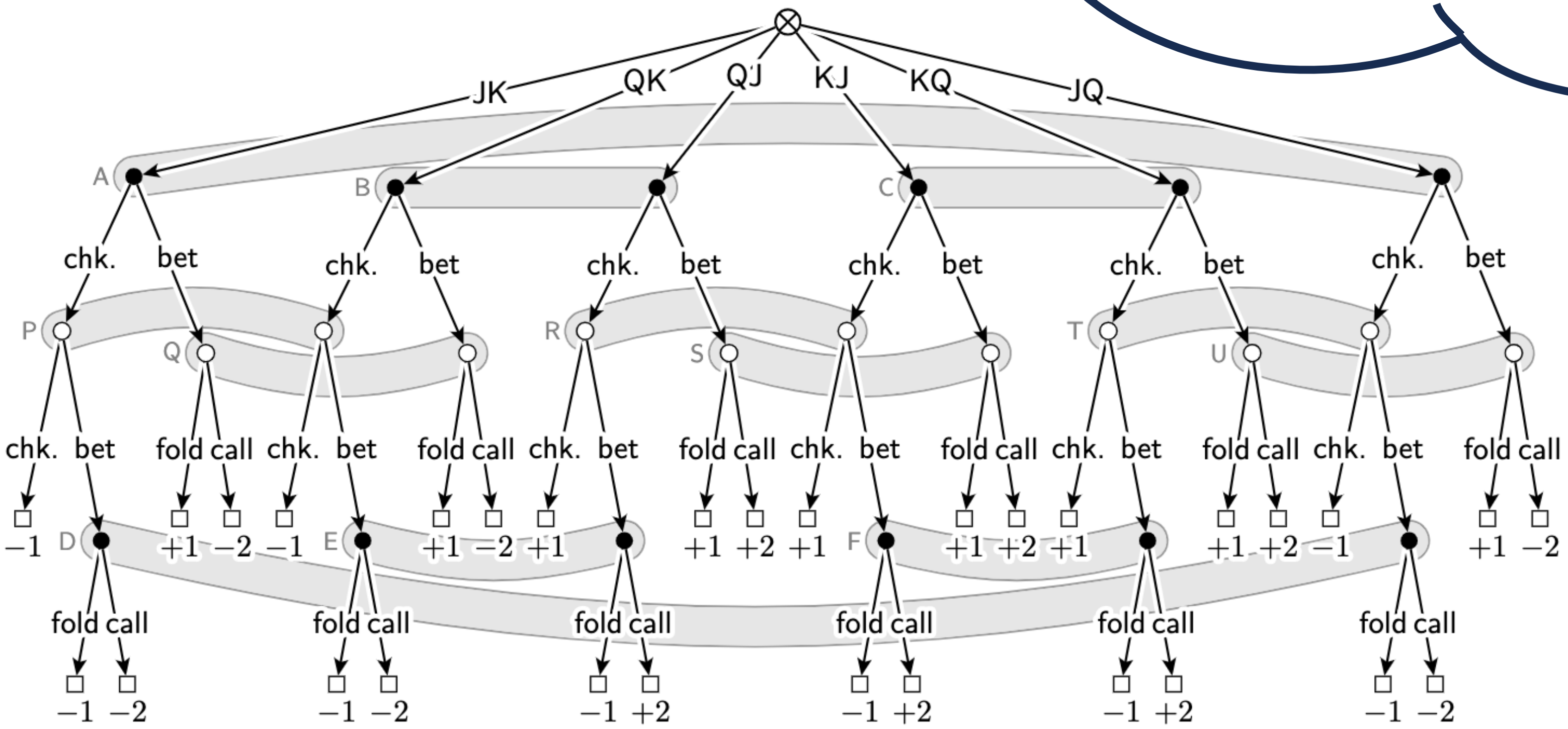
Effectively boils down to figuring out:

for each information set J of the player, what was the last (information set, action) pair <u>for the player</u> on the path from the root of the tree to J? **("parent" of J)**



J	Actions	Parent
A	[chk, bet]	
B	[chk, bet]	
C	[chk, bet]	
D	[fold, call]	
E	[fold, call]	
F	[fold, call]	





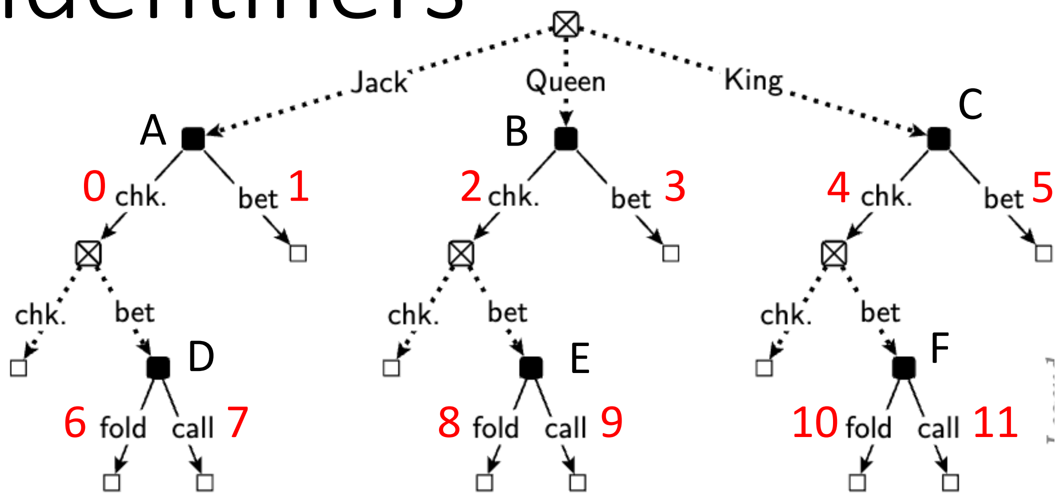
J	Actions	Parent
A	[chk, bet]	None
B	[chk, bet]	None
C	[chk, bet]	None
D	[fold, call]	(A, chk)
E	[fold, call]	(B, chk)
F	[fold, call]	(C, chk)

Step 2: Assign numerical identifiers

We will use numerical IDs to each action at each information set

J	Actions	Parent
A	[chk, bet]	None
B	[chk, bet]	None
C	[chk, bet]	None
D	[fold, call]	(A, chk)
E	[fold, call]	(B, chk)
F	[fold, call]	(C, chk)

(J, action)	ID
(A, chk)	0
(A, bet)	1
(B, chk)	2
(B, bet)	3
(C, chk)	4
(C, bet)	5
(D, fold)	6
(D, call)	7
(E, fold)	8
…	…
(F, call)	11



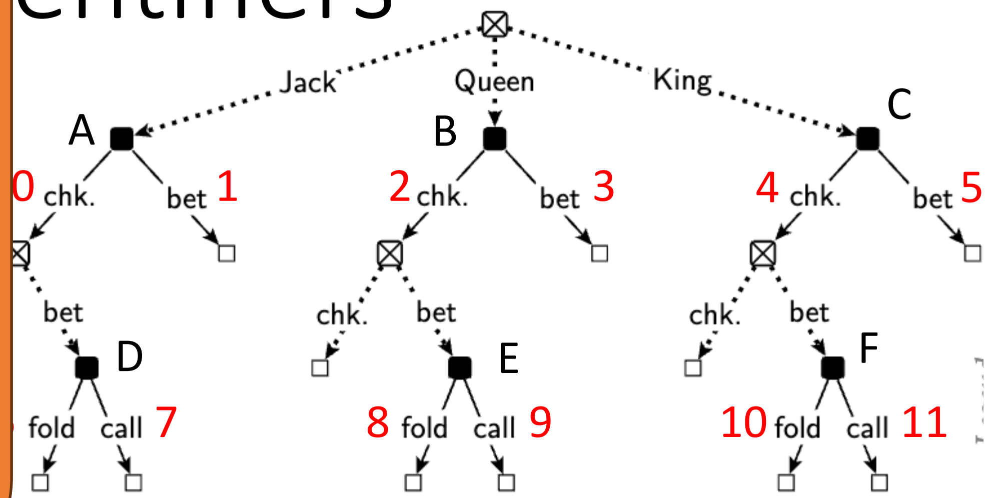
Sequence-form constraints:

$$\begin{cases} x_0 + x_1 = 1 \\ x_2 + x_3 = 1 \\ x_4 + x_5 = 1 \\ x_6 + x_7 = x_0 \\ x_8 + x_9 = x_2 \\ x_{10} + x_{11} = x_4 \\ x_0, \dots, x_{11} \geq 0 \end{cases}$$

In matrix-vector form,

$$\underbrace{\begin{bmatrix} +1 & +1 & & & & & & & & & & \\ & & +1 & +1 & & & & & & & & \\ & & & & +1 & +1 & & & & & & \\ -1 & & & & & & +1 & +1 & & & & \\ & & -1 & & & & & & +1 & +1 & & \\ & & & & -1 & & & & & & +1 & +1 \end{bmatrix}}_{F_1} x = \underbrace{\begin{bmatrix} +1 \\ +1 \\ +1 \\ \\ \\ \end{bmatrix}}_{f_1}$$

entifiers



A	[chk, bet]	None
B	[chk, bet]	None
C	[chk, bet]	None
D	[fold, call]	(A, chk)
E	[fold, call]	(B, chk)
F	[fold, call]	(C, chk)

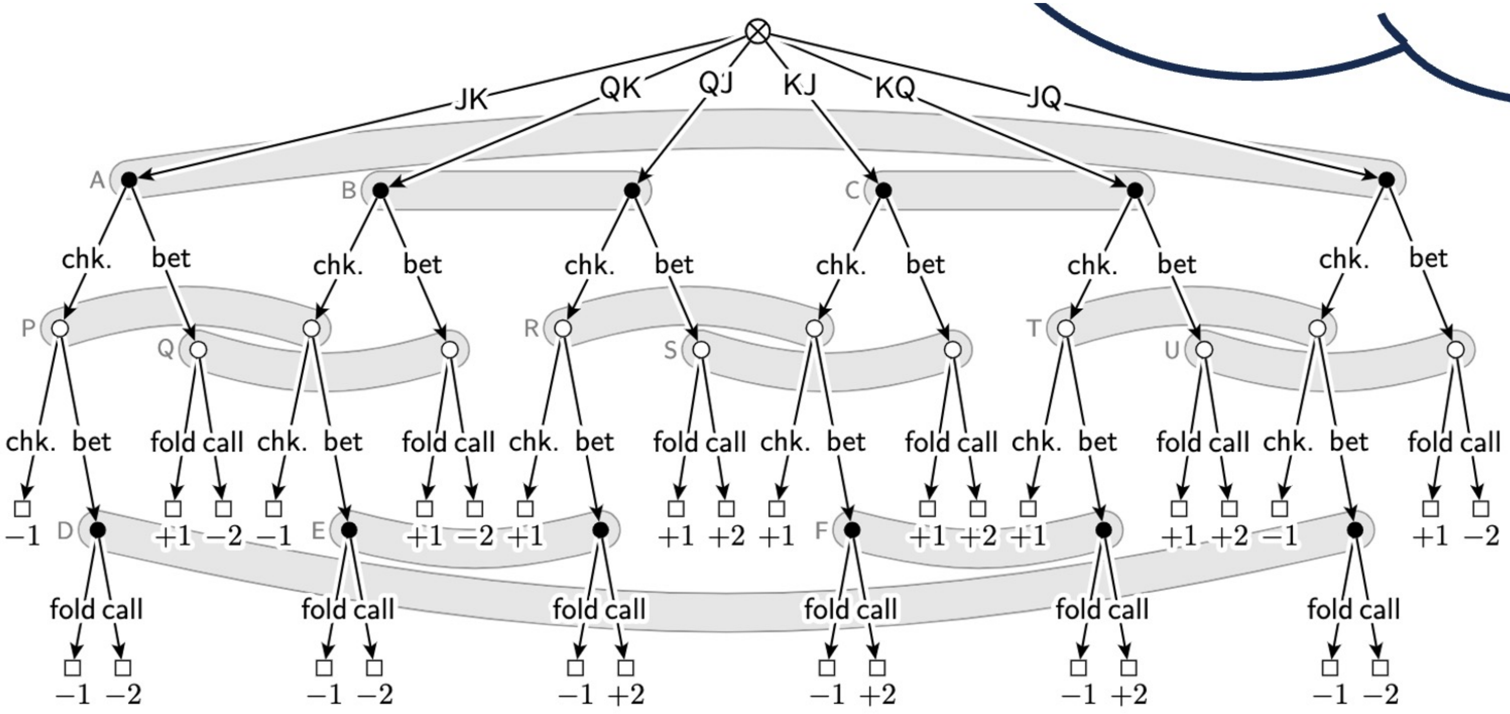
(C, bet)	5
(D, fold)	6
(D, call)	7
(E, fold)	8
…	…
(F, call)	11

Sequence-form constraints:

$$\begin{cases} x_0 + x_1 = 1 \\ x_2 + x_3 = 1 \\ x_4 + x_5 = 1 \\ x_6 + x_7 = x_0 \\ x_8 + x_9 = x_2 \\ x_{10} + x_{11} = x_4 \\ x_0, \dots, x_{11} \geq 0 \end{cases}$$

Plan of attack

- Step 1: for each player, figure out the parent relationships

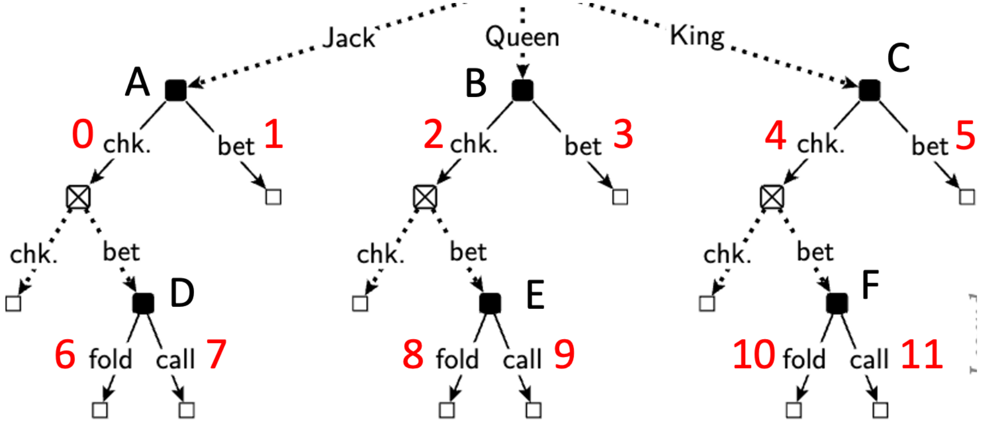


J	Actions	Parent
A	[chk, bet]	None
B	[chk, bet]	None
C	[chk, bet]	None
D	[fold, call]	(A, chk)
E	[fold, call]	(B, chk)
F	[fold, call]	(C, chk)

Plan of attack

- Step 1: for each player, figure out the parent relationships
- Step 2: then, assign numerical IDs and compile the matrices F and f

(J, action)	ID
(A, chk)	0
(A, bet)	1
(B, chk)	2
(B, bet)	3
(C, chk)	4
(C, bet)	5
(D, fold)	6
(D, call)	7
(E, fold)	8
…	…



Sequence-form constraints:

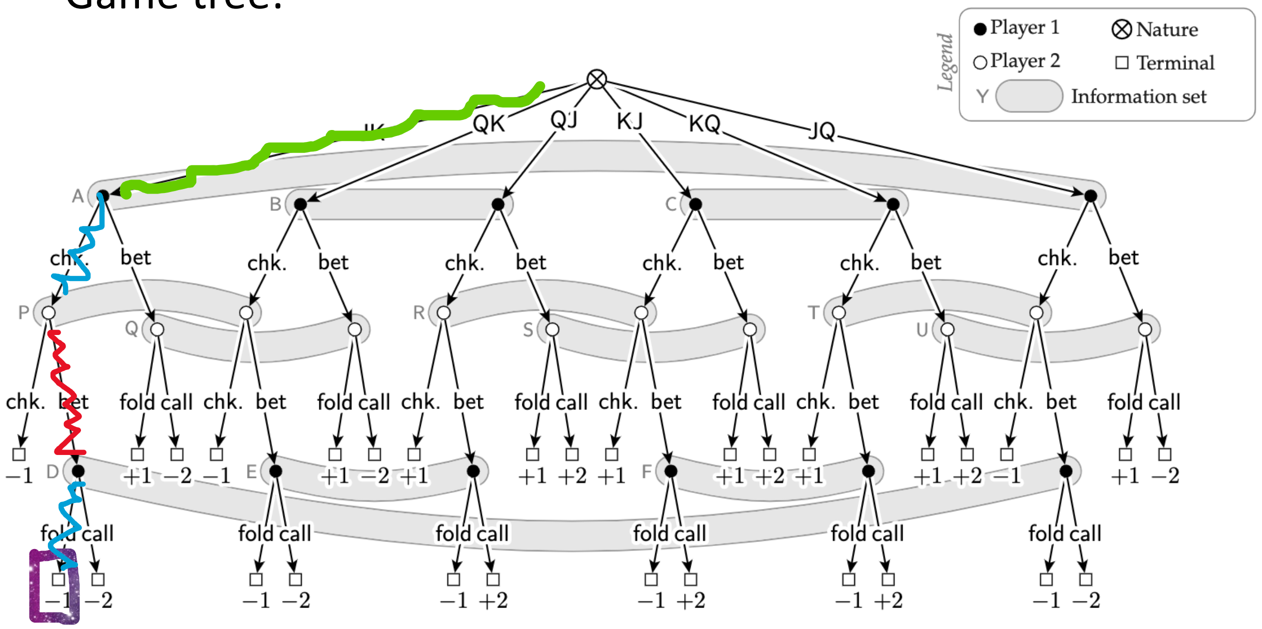
$$
\begin{cases}
x_0 + x_1 = 1 \\
x_2 + x_3 = 1 \\
x_4 + x_5 = 1 \\
x_6 + x_7 = x_0
\end{cases}
$$

In matrix-vector form,

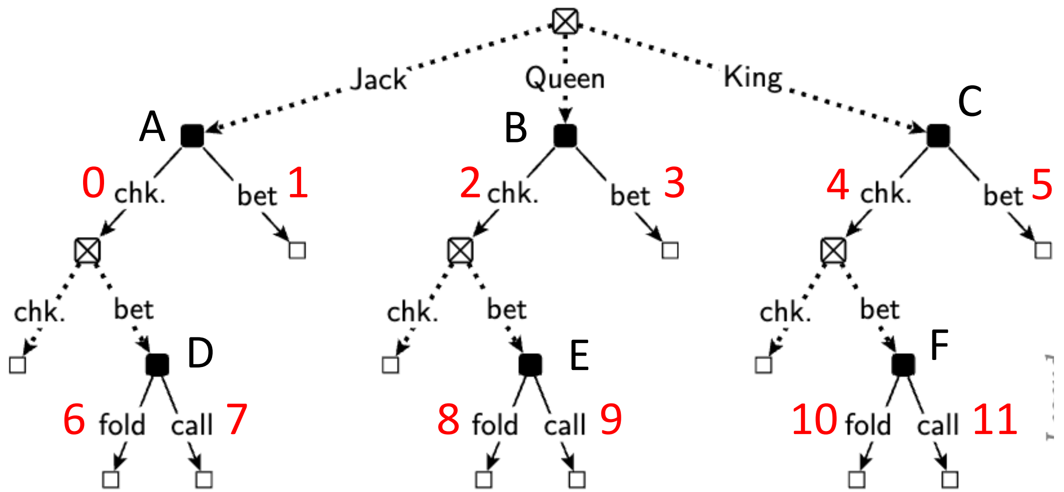
$$
\underbrace{\begin{bmatrix}
+1 & +1 & & & & & & & & & & \\
 & & +1 & +1 & & & & & & & & \\
 & & & & +1 & +1 & & & & & & \\
-1 & & & & & & +1 & +1 & & & & \\
 & & -1 & & & & & & +1 & +1 & & \\
 & & & & -1 & & & & & & +1 & +1
\end{bmatrix}}_{F_1} x = \underbrace{\begin{bmatrix} +1 \\ +1 \\ +1 \\ \\ \\ \\ \end{bmatrix}}_{f_1}
$$

The Payoff Matrix A

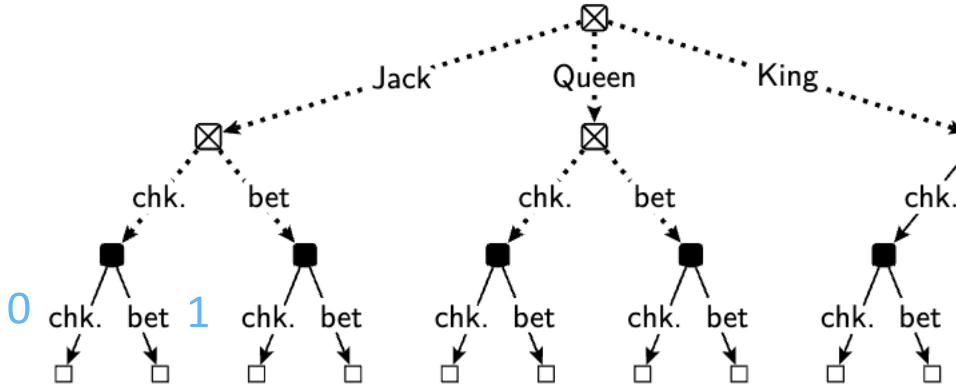
Game tree:



Decision problem and behavioral strategy of Player 1



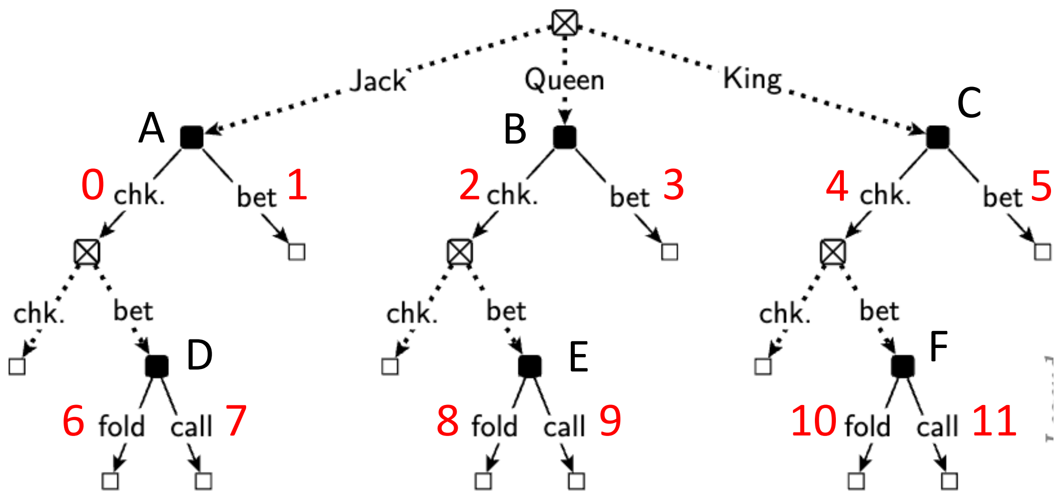
Decision problem and behavioral strategy of Player 2



Prob of reaching this terminal state: 1/6 (Nature) × x_6 (Pl1) × y_1 (Pl1)

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

Implementation



- class **Game**
  - `tpx_pl1`: **Treeplex**
  - `tpx_pl2`: **Treeplex**
  - `A`: payoff matrix (numpy array, player 1 on rows for A)
- class **Treeplex**
  - `infosets`: dict[str, **Infoset**]
  - `num_seqs`: int. Total number of actions across decision points (12 in figure)
- class **Infoset**:
  - `actions`: dictionary from action name (e.g., "fold") to unique ID (e.g., 6)
  - `parent:` unique ID of the parent infoset action. (may be None)

```python

game = Game('kuhn.txt')

def make_Ff(tpx):
    F = np.zeros((len(tpx.infosets), tpx.num_seqs))
    f = np.zeros((len(tpx.infosets)))

    for i, infoset in enumerate(tpx.infosets.values()):
        for a in infoset.actions.values():
            F[i, a] = 1
        if infoset.parent is None:
            f[i] = 1
        else:
            F[i, infoset.parent] = -1

    return F, f

F_1, f_1 = make_Ff(game.tpx_pl1)
print(F_1, f_1.T)

F_2, f_2 = make_Ff(game.tpx_pl2)
print(F_2, f_2.T)

m = gp.Model()
x = m.addMVar(game.tpx_pl1.num_seqs)
v = m.addMVar(len(game.tpx_pl2.infosets), lb=float("-inf"))

m.addConstr(F_1 @ x == f_1)
m.addConstr(F_2.T @ v <= game.A.T @ x)
m.setObjective(f_2 @ v, sense=GRB.MAXIMIZE)
m.optimize()
```