

# Dynamic Games

- Markov Games
- 2 Player zero sum turn taking games

# Markov Games

# Markov Game Definition

# Markov Game Definition

$$(I, S, A, T, R, \gamma)$$

- $I$  = set of players
- $S$  = state space (combined for all players)
- $A$  = *joint* action space
- $T$  = transition distributions:  $T(s' | s, a)$  is the distribution of  $s'$  given state  $s$  and *joint* action  $a$
- $R$  = joint reward function:  $R^i(s, a)$  is the reward for agent  $i$  in state  $s$  when *joint* action  $a$  is taken

# Goofspiel (or Game of Pure Strategy)

# Goofspiel (or Game of Pure Strategy)

- Aces Low
- One suit represents the prizes
- Each player gets the cards from one of the other suits
- At each step, one of the prize cards is presented
- Both players simultaneously present a card and the high card wins the prize
- Prizes discarded with ties

# Goofspiel(5) Optimal Strategy

**Table 1.** Optimal strategy at the first move,  $N = 5$ .

	upcard				
	1	2	3	4	5
1	0.0470	0.1855	0.1182	0.1226	0.1123
2	0.8327	0	0.1188	0.07347	0.0241
3	0.1203	0.7375	0	0.1915	0
4	0	0.0770	0.7630	0.2043	0
5	0	0	0	0.4081	0.8636

Calculated with Dynamic Programming  
[Rhoads and Bartholdi, 2012]

This is not a game payoff matrix!

# Nash equilibrium at every state

$$\underset{\pi, U}{\text{minimize}} \quad \sum_{i \in \mathcal{I}} \sum_s \left( U^i(s) - Q^i(s, \pi(s)) \right)$$

$$\text{subject to} \quad U^i(s) \geq Q^i(s, a^i, \pi^{-i}(s)) \text{ for all } i, s, a^i$$

$$\sum_{a^i} \pi^i(a^i \mid s) = 1 \text{ for all } i, s$$

$$\pi^i(a^i \mid s) \geq 0 \text{ for all } i, s, a^i$$

where

$$Q^i(s, \pi(s)) = R^i(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U^i(s')$$

# Reduction to Simple Game

P2

	$\sigma_1^2$	$\sigma_2^2$	...	$\sigma_n^2$
$\sigma_1^1$	$U(\sigma_1^1, \sigma_1^2)$			
$\sigma_2^1$				
$\sigma_3^1$				
:				
$\sigma_n^1$				

P1

$$V^i(\sigma^1, \sigma^2) = \sum_s R^i(s, \sigma^1(s), \sigma^2(s))$$

$\sigma_i^1(s)$ : deterministic strategy

# Best response in a Markov Game

# Fictitious Play in a Markov Game

# Markov Game Recap

1. Definition  $(I, S, A, T, R, \gamma)$
2. Reduction to simple game
3. Computing a best response means solving  
an MDP - know how to construct the MDP

# Turn-taking Games

# Turn-taking Games



Terminology

# Turn-taking Games



## Terminology

- Max and Min players

# Turn-taking Games



## Terminology

- Max and Min players
- deterministic

# Turn-taking Games



## Terminology

- Max and Min players
- deterministic
- two player

# Turn-taking Games



## Terminology

- Max and Min players
- deterministic
- two player
- zero-sum

# Turn-taking Games



## Terminology

- Max and Min players
- deterministic
- two player
- zero-sum
- perfect information

# Minimax Trees

# Minimax Trees

MDP Expectimax Tree

# Minimax Trees

MDP Expectimax Tree

Minimax Tree

# Minimax Trees

MDP Expectimax Tree

Minimax Tree

$$V(s) = \max_{a \in \mathcal{A}} (R(s, a) + \mathbb{E}[V(s')])$$

# Minimax Trees

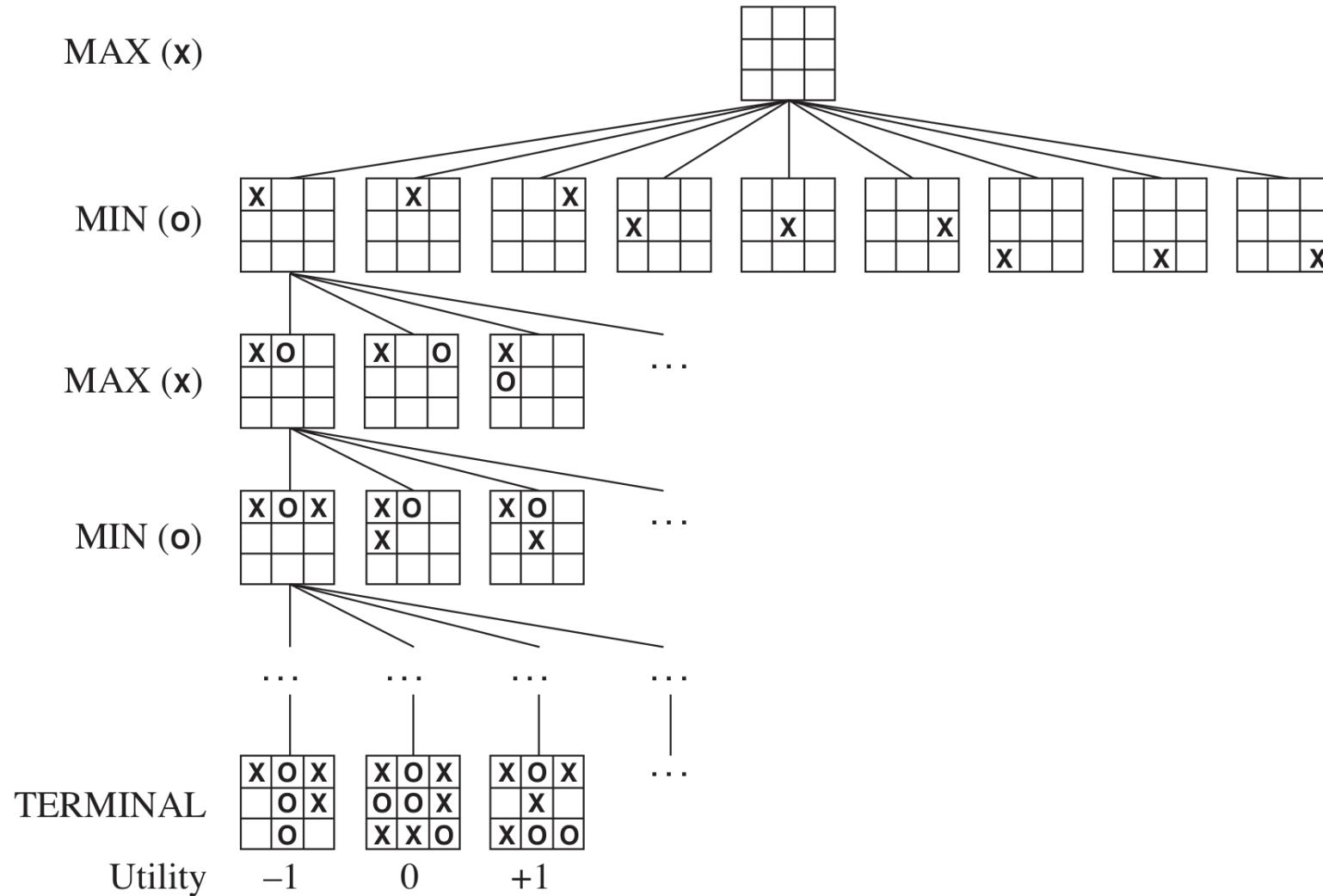
MDP Expectimax Tree

$$V(s) = \max_{a \in \mathcal{A}} (R(s, a) + \mathbb{E}[V(s')])$$

Minimax Tree

$$V(s) = \max_{a \in \mathcal{A}_1} (R(s, a) + \min_{a' \in \mathcal{A}_2} (R(s', a') + V(s'')))$$

# Tic-Tac-Toe Example



# Tree Backup Example

# Tree Backup Example

---

```
function MINIMAX-DECISION(state) returns an action
    return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 
```

---

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow -\infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

---

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow \infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

# Tree Backup Example

---

```
function MINIMAX-DECISION(state) returns an action
    return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 
```

---

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow -\infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

---

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow \infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

3 12 8 2 4 6 19 5 2

# Tree Backup Example

```
function MINIMAX-DECISION(state) returns an action
    return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 
```

---

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow -\infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

---

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
     $v \leftarrow \infty$ 
    for each a in ACTIONS(state) do
         $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
    return v
```

3 12 8 2 4 6 19 5 2

Why is this harder than an MDP? (think  
back to sparse sampling)

# Alpha-Beta Pruning

**function** ALPHA-BETA-SEARCH(*state*) **returns** an action

$v \leftarrow \text{MAX-VALUE}(\textit{state}, -\infty, +\infty)$

**return** the *action* in ACTIONS(*state*) with value  $v$

---

**function** MAX-VALUE(*state*,  $\alpha$ ,  $\beta$ ) **returns** a utility value

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for each** *a* in ACTIONS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$

**if**  $v \geq \beta$  **then return**  $v$

$\alpha \leftarrow \text{MAX}(\alpha, v)$

**return**  $v$

---

**function** MIN-VALUE(*state*,  $\alpha$ ,  $\beta$ ) **returns** a utility value

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow +\infty$

**for each** *a* in ACTIONS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$

**if**  $v \leq \alpha$  **then return**  $v$

$\beta \leftarrow \text{MIN}(\beta, v)$

**return**  $v$

# Alpha-Beta Pruning

---

```
function ALPHA-BETA-SEARCH(state) returns an action
  v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
  return the action in ACTIONS(state) with value v
```

---

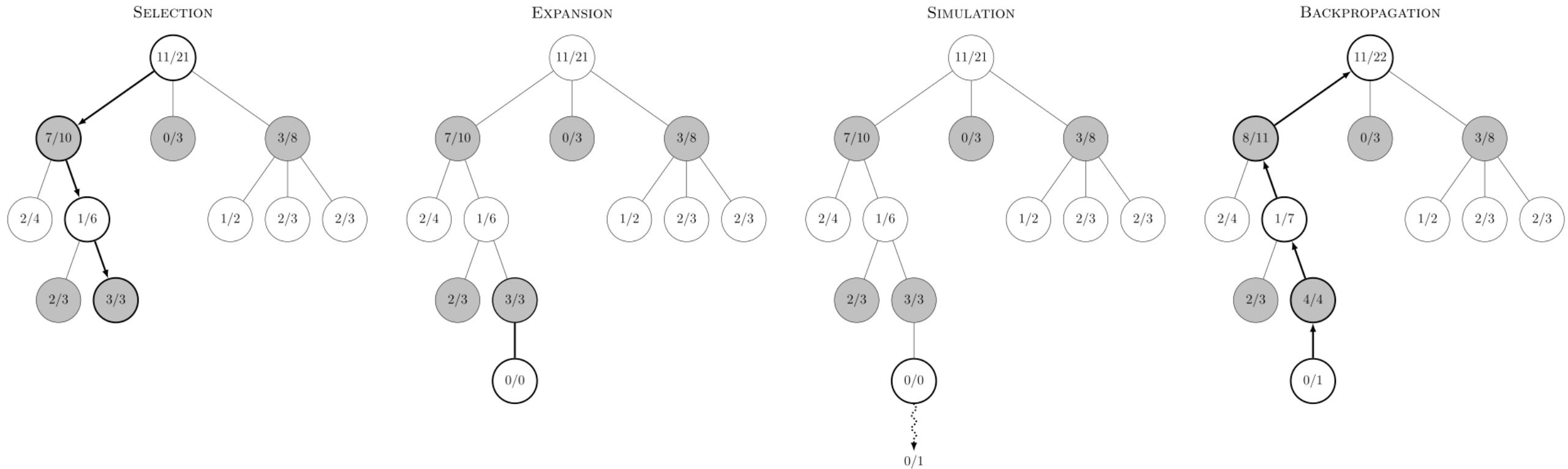
```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow -\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if v  $\geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v
```

---

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v  $\leftarrow +\infty$ 
  for each a in ACTIONS(state) do
    v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if v  $\leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v
```

3 12 8 2 4 6 19 5 2

# MCTS for Games



Note: the above example does not follow UCB exploration

# Imperfect Information



But you must have known I was not a great fool

# Partially Observable Markov Game

# Belief updates?

# Reduction to Simple Game

# Extensive Form Game

(Alternative to POMGs that is more common in the literature)

- Similar to a minimax tree for a turn-taking game
- Chance nodes
- Information sets

# King-Ace Poker Example

# King-Ace Poker Example

- 4 Cards: 2 Aces, 2 Kings

# King-Ace Poker Example

- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card

# King-Ace Poker Example

- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* ( $r$ ) the payoff to 2 points or *check* ( $k$ ) the payoff at 1 point

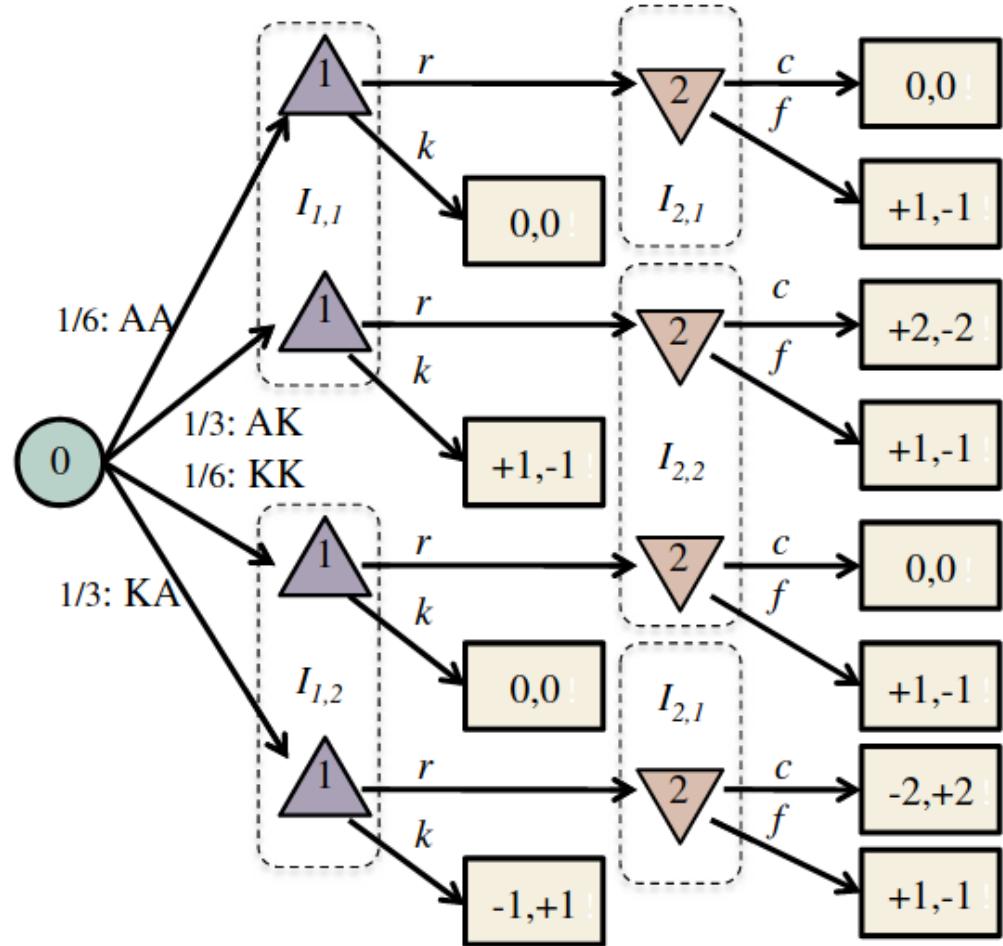
# King-Ace Poker Example

- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* ( $r$ ) the payoff to 2 points or *check* ( $k$ ) the payoff at 1 point
- If P1 raises, P2 can either *call* ( $c$ ) Player 1's bet, or *fold* ( $f$ ) the payoff back to 1 point

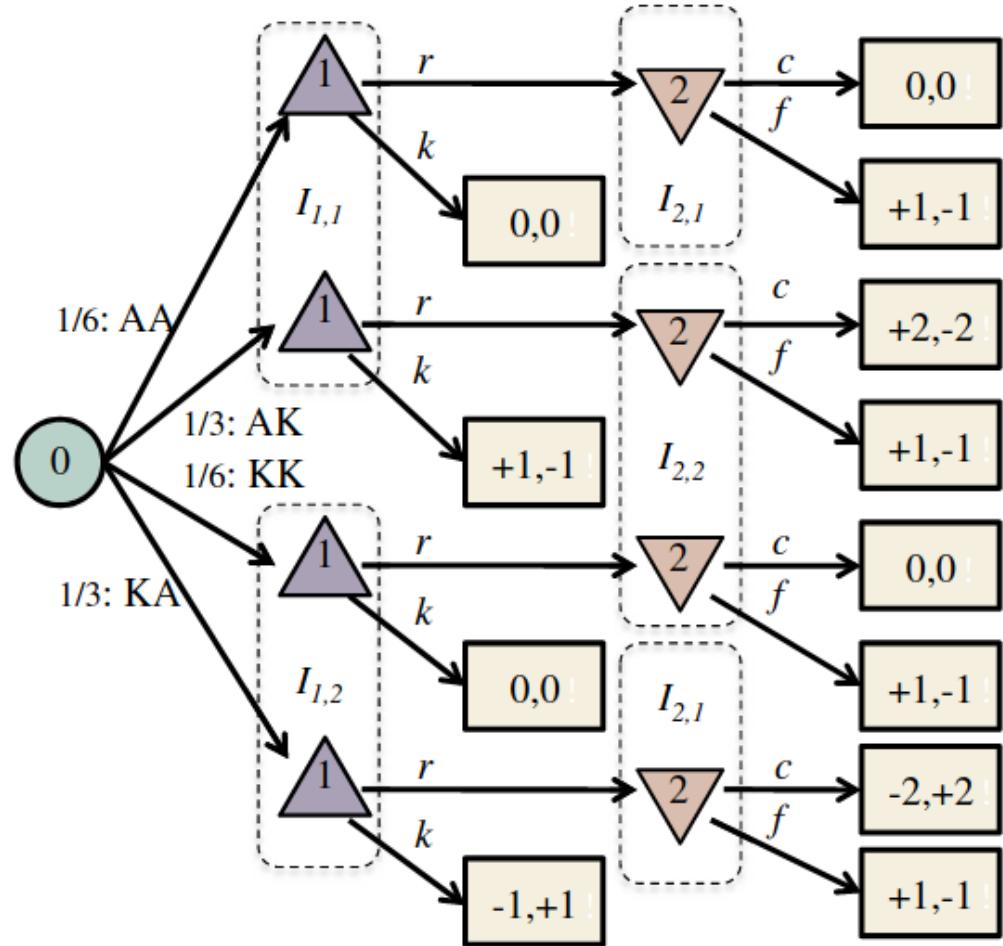
# King-Ace Poker Example

- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* ( $r$ ) the payoff to 2 points or *check* ( $k$ ) the payoff at 1 point
- If P1 raises, P2 can either *call* ( $c$ ) Player 1's bet, or *fold* ( $f$ ) the payoff back to 1 point
- The highest card wins

# Extensive to Matrix Form

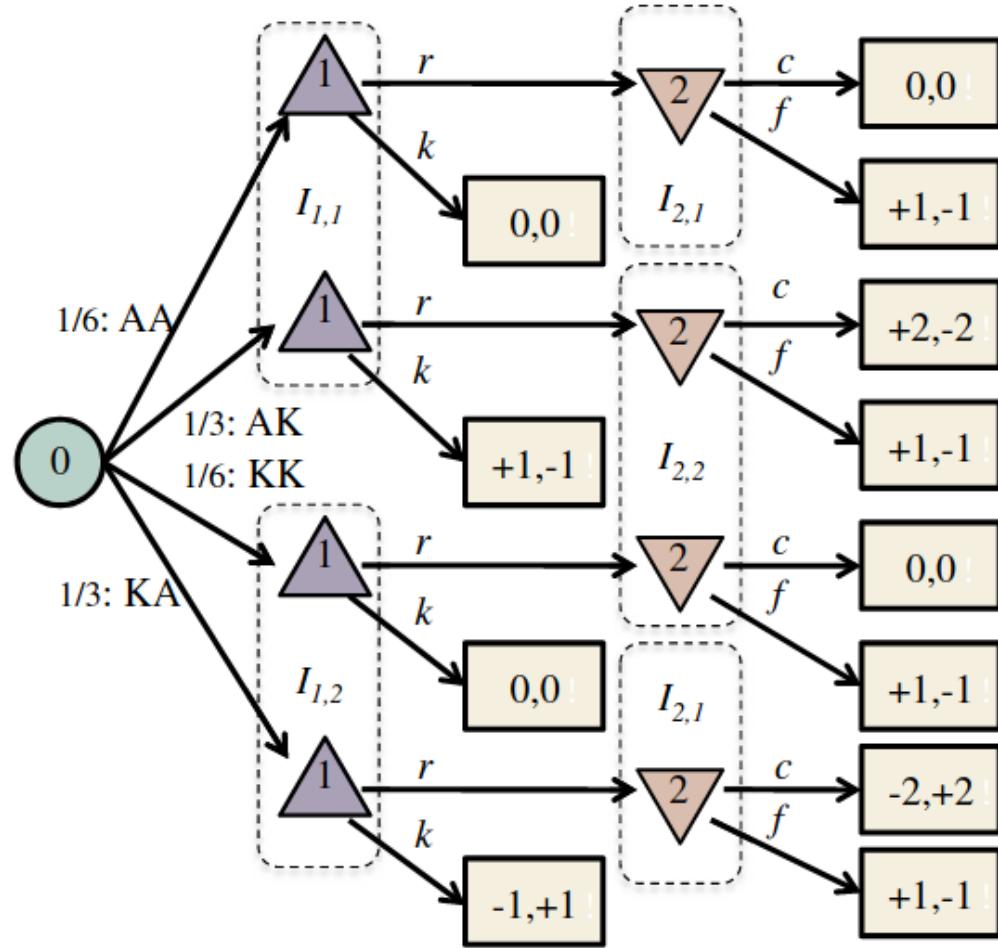


# Extensive to Matrix Form



	2:cc	2:cf	2:ff	2:fc
1:rr	0	-1/6	1	7/6
1:kr	-1/3	-1/6	5/6	2/3
1:rk	1/3	0	1/6	1/2
1:kk	0	0	0	0

# Extensive to Matrix Form



	$2:cc$	$2:cf$	$2:ff$	$2:fc$
$1:rr$	0	$-1/6$	1	$7/6$
$1:kr$	$-1/3$	$-1/6$	$5/6$	$2/3$
$1:rk$	$1/3$	0	$1/6$	$1/2$
$1:kk$	0	0	0	0

Exponential in number of info states!

# Fictitious Play in Extensive Form Games

**Algorithm 2** General Fictitious Self-Play

---

```

function FICTITIOUSSELFPLAY( $\Gamma, n, m$ )
    Initialize completely mixed  $\pi_1$ 
     $\beta_2 \leftarrow \pi_1$ 
     $j \leftarrow 2$ 
    while within computational budget do
         $\eta_j \leftarrow \text{MIXINGPARAMETER}(j)$ 
         $\mathcal{D} \leftarrow \text{GENERATEDATA}(\pi_{j-1}, \beta_j, n, m, \eta_j)$ 
        for each player  $i \in \mathcal{N}$  do
             $\mathcal{M}_{RL}^i \leftarrow \text{UPDATERLMEMORY}(\mathcal{M}_{RL}^i, \mathcal{D}^i)$ 
             $\mathcal{M}_{SL}^i \leftarrow \text{UPDATESLMEMORY}(\mathcal{M}_{SL}^i, \mathcal{D}^i)$ 
             $\beta_{j+1}^i \leftarrow \text{REINFORCEMENTLEARNING}(\mathcal{M}_{RL}^i)$ 
             $\pi_j^i \leftarrow \text{SUPERVISEDLEARNING}(\mathcal{M}_{SL}^i)$ 
        end for
         $j \leftarrow j + 1$ 
    end while
    return  $\pi_{j-1}$ 
end function

function GENERATEDATA( $\pi, \beta, n, m, \eta$ )
     $\sigma \leftarrow (1 - \eta)\pi + \eta\beta$ 
     $\mathcal{D} \leftarrow n$  episodes  $\{t_k\}_{1 \leq k \leq n}$ , sampled from strategy
    profile  $\sigma$ 
    for each player  $i \in \mathcal{N}$  do
         $\mathcal{D}^i \leftarrow m$  episodes  $\{t_k^i\}_{1 \leq k \leq m}$ , sampled from strat-
        egy profile  $(\beta^i, \sigma^{-i})$ 
         $\mathcal{D}^i \leftarrow \mathcal{D}^i \cup \mathcal{D}$ 
    end for
    return  $\{\mathcal{D}^k\}_{1 \leq k \leq N}$ 
end function

```

---

