Dynamic Games

- 2 Player zero sum turn taking games
- Markov Games
- Extensive Form Games

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

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

Minimax Tree

Minimax Trees

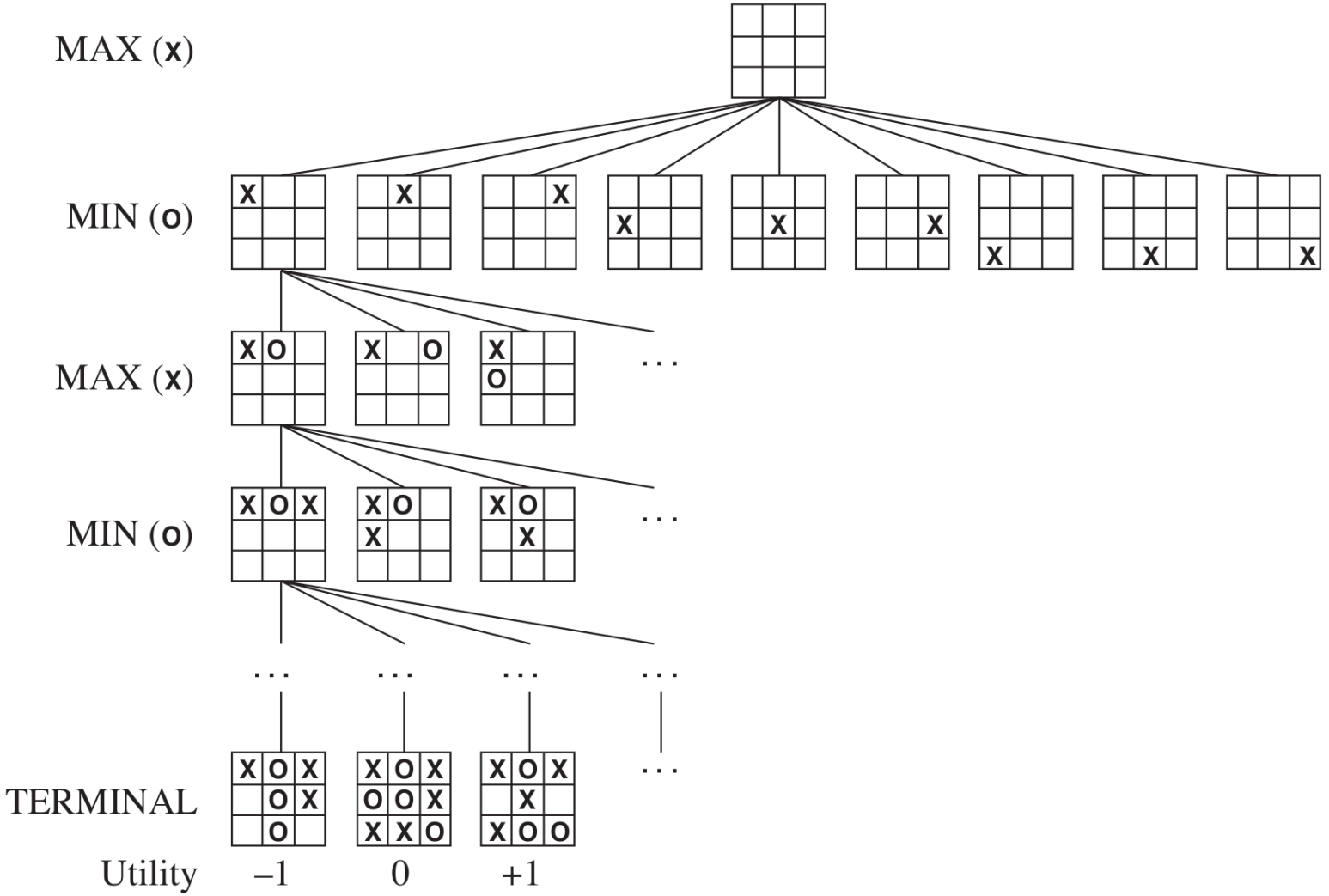
MDP Expectimax Tree

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

Minimax Tree

$$V(s) = \max_{a \in \mathcal{A}_1} \left(R(s,a) + \min_{a' \in A_2} (R(s',a') + V(s'')) \right)$$

Tic-Tac-Toe Example



Tree Backup Example

Tree Backup Example

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

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Tree Backup Example

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

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

3 12 8 2 4 6 14 5 2

Tree Backup Example

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

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

3 12 8 2 4 6 14 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 ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Alpha-Beta Pruning

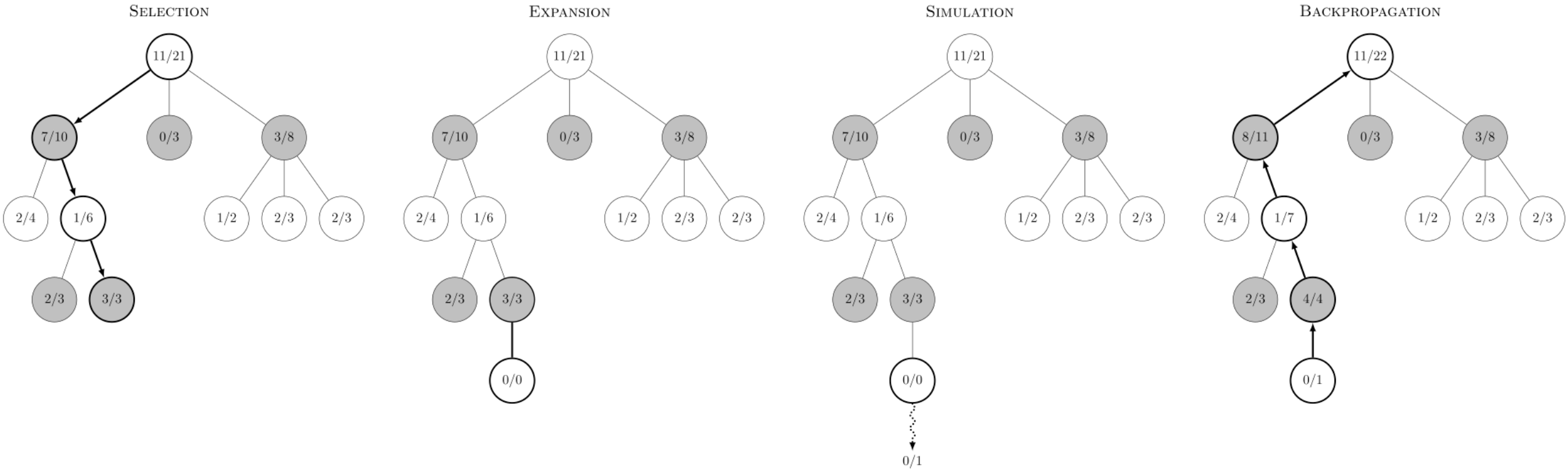
```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

3 12 8 2 4 6 14 5 2

MCTS for Games



Markov Games

Markov Game Definition

Nash equilibrium at every state

$$
\begin{aligned}
\underset{\boldsymbol{\pi}, U}{\text{minimize}} \quad & \sum_{i \in \mathcal{I}} \sum_{s} \left(U^i(s) - Q^i(s, \boldsymbol{\pi}(s)) \right) \\
\text{subject to} \quad & U^i(s) \geq Q^i(s, a^i, \boldsymbol{\pi}^{-i}(s)) \text{ for all } i, s, a^i \\
& \sum_{a^i} \pi^i(a^i \mid s) = 1 \text{ for all } i, s \\
& \pi^i(a^i \mid s) \geq 0 \text{ for all } i, s, a^i
\end{aligned}
$$

where

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

Reduction to Simple Game

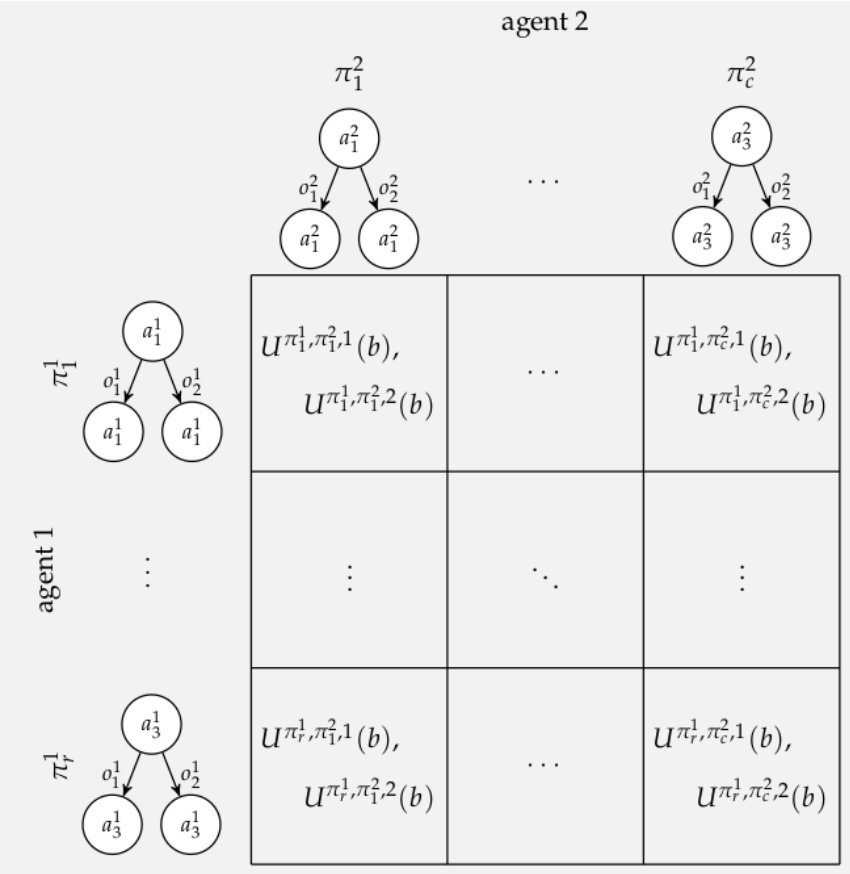
Best response in a Markov Game

Fictitious Play in a Markov Game

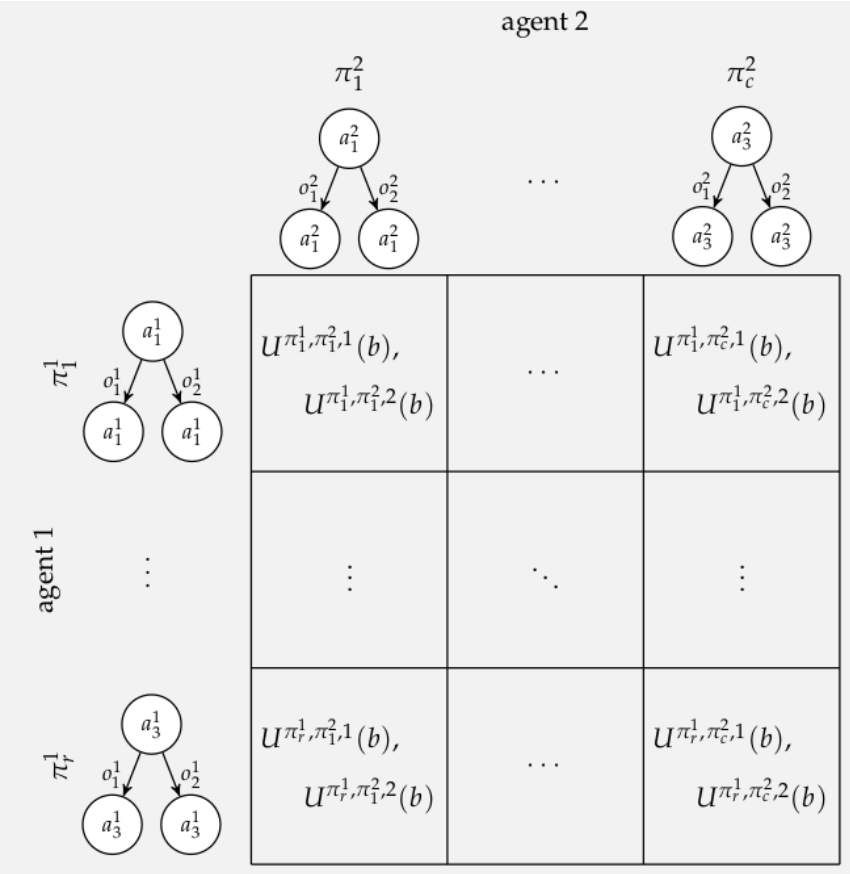
Incomplete Information

Partially Observable Markov Game

Partially Observable Markov Game

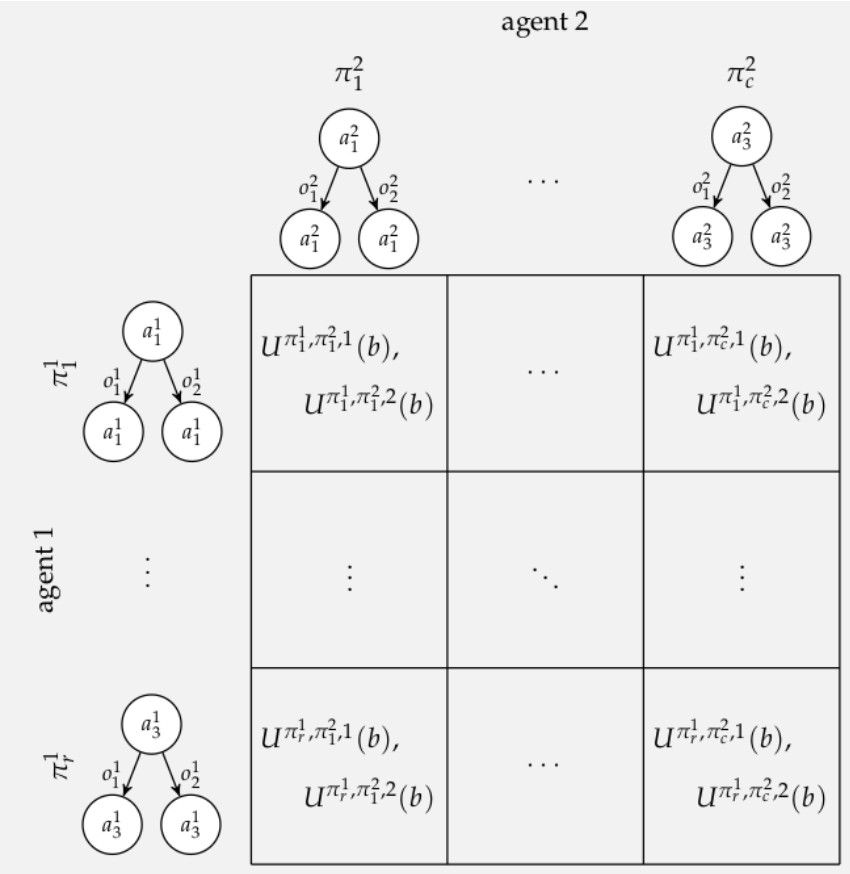


Partially Observable Markov Game



$$\sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}, i}(s) \geq \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i}, \boldsymbol{\pi}^{-i}, i}(s)$$

Partially Observable Markov Game



$$\sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}, i}(s) \geq \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) U^{\pi^{i}, \boldsymbol{\pi}^{-i}, i}(s)$$

$$\begin{aligned}
\underset{\delta, b}{\text{maximize}} \quad & \delta \\
\text{subject to} \quad & b(\boldsymbol{\pi}^{-i}, s) \geq 0 \text{ for all } \boldsymbol{\pi}^{-i}, s \\
& \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) = 1 \\
& \sum_{\boldsymbol{\pi}^{-i}} \sum_{s} b(\boldsymbol{\pi}^{-i}, s) \left(U^{\pi^{i\prime}, \boldsymbol{\pi}^{-i}, i}(s) - U^{\pi^{i}, \boldsymbol{\pi}^{-i}, i}(s) \right) \geq \delta \text{ for all } \pi^{i\prime}
\end{aligned}$$

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

- Finite set of n players, plus the "chance" player

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

- Finite set of n players, plus the "chance" player
- $P(h)$ (player at each history)

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

- Finite set of n players, plus the "chance" player
- $P(h)$ (player at each history)
- $A(h)$ (set of actions at each history)

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

- Finite set of n players, plus the "chance" player
- $P(h)$ (player at each history)
- $A(h)$ (set of actions at each history)
- $I(h)$ (information set that each history maps to)

Extensive Form Game

Extensive-form game definition (h is a sequence of actions called a "history"):

- Finite set of n players, plus the "chance" player
- $P(h)$ (player at each history)
- $A(h)$ (set of actions at each history)
- $I(h)$ (information set that each history maps to)
- $U(h)$ (payoff for each leaf node in the game tree)

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

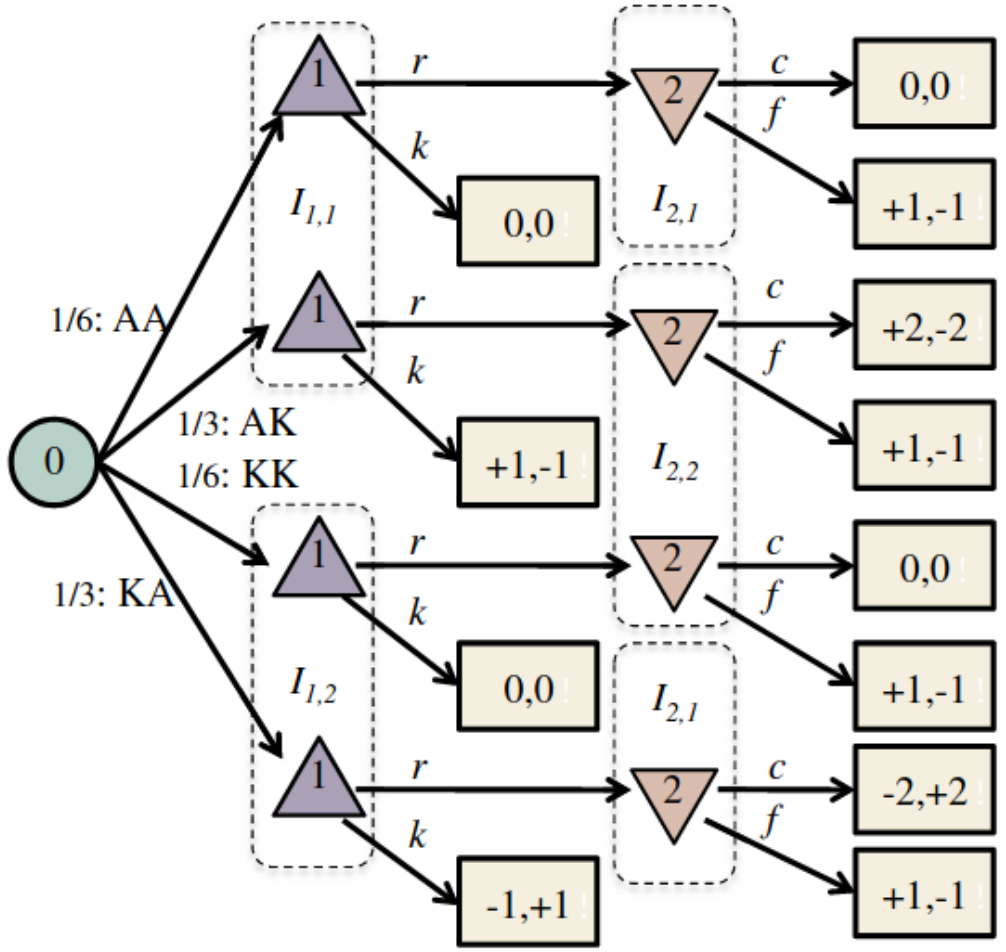
- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* (r) the payoff to 2 points or *check* (k) the payoff at 1 point
- If P1 raises, P2 can either *call* (c) Player 1's bet, or *fold* (f) the payoff back to 1 point

King-Ace Poker Example

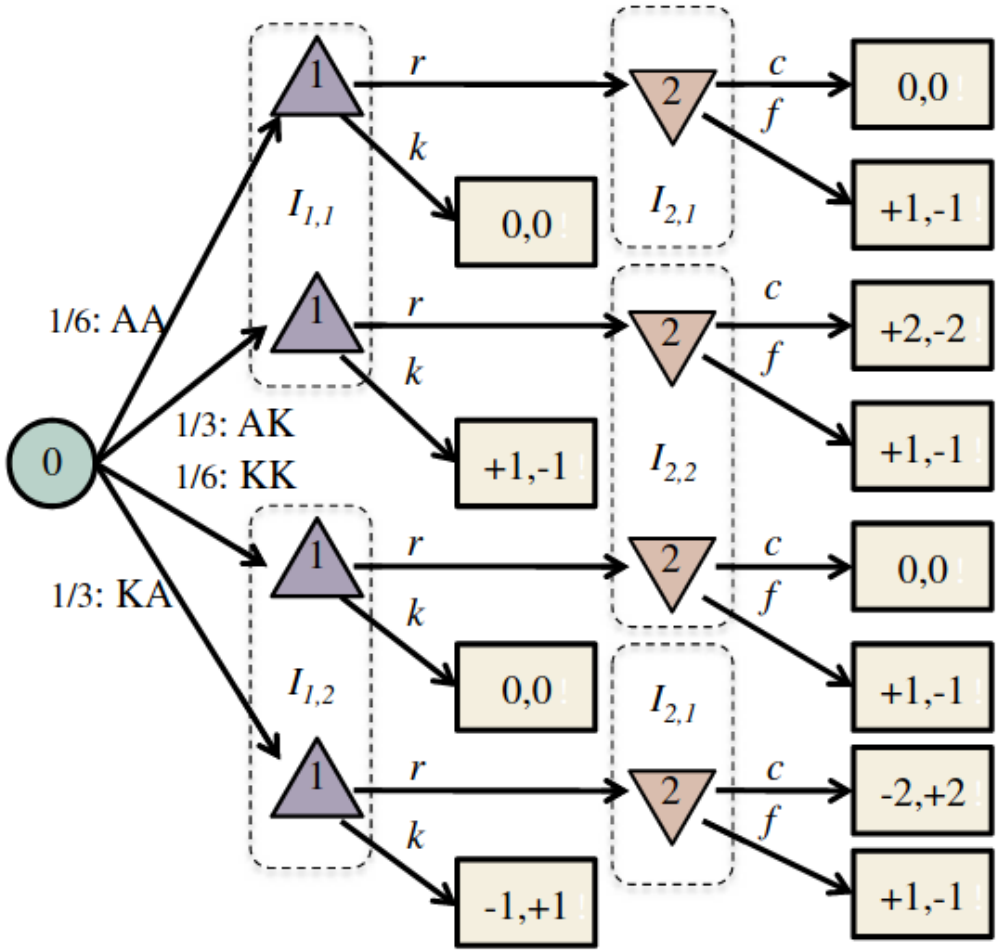
- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* (r) the payoff to 2 points or *check* (k) the payoff at 1 point
- If P1 raises, P2 can either *call* (c) Player 1's bet, or *fold* (f) the payoff back to 1 point
- The highest card wins

King-Ace Poker Example

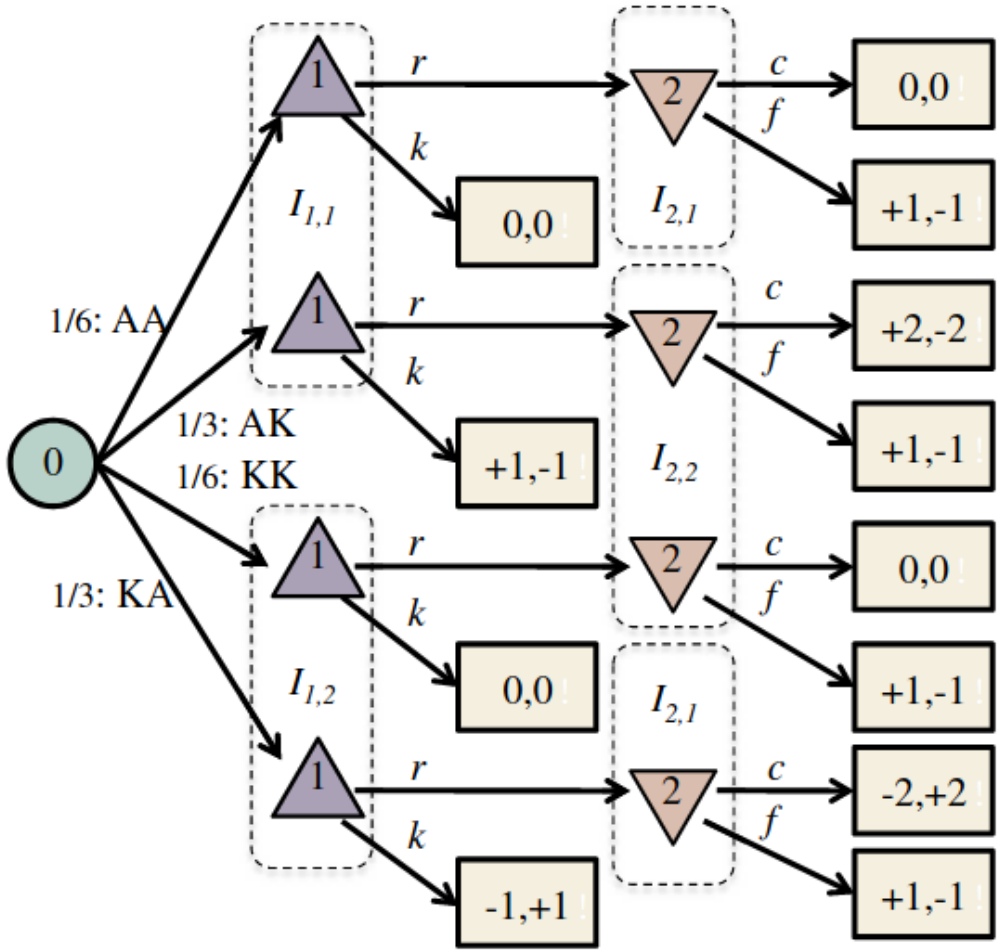
- 4 Cards: 2 Aces, 2 Kings
- Each player is dealt a card
- P1 can either *raise* (r) the payoff to 2 points or *check* (k) the payoff at 1 point
- If P1 raises, P2 can either *call* (c) Player 1's bet, or *fold* (f) the payoff back to 1 point
- The highest card wins



Extensive to Matrix Form

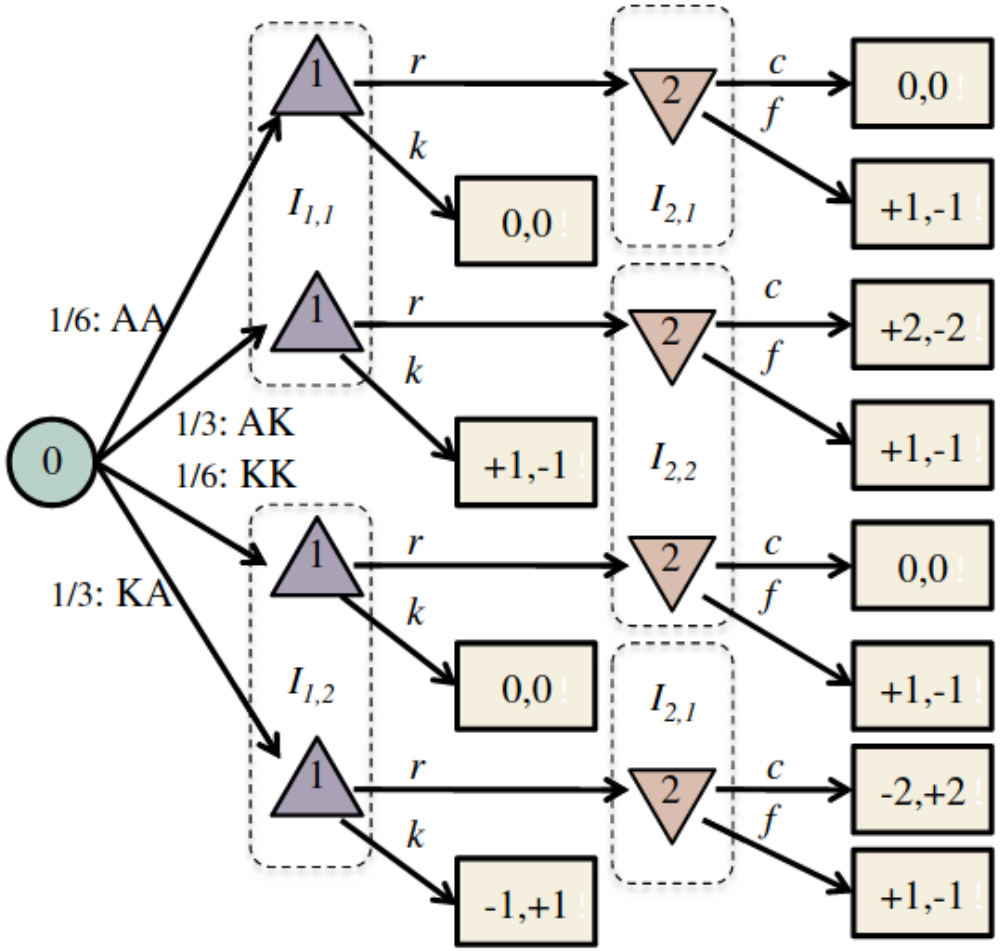


Extensive to Matrix Form



	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

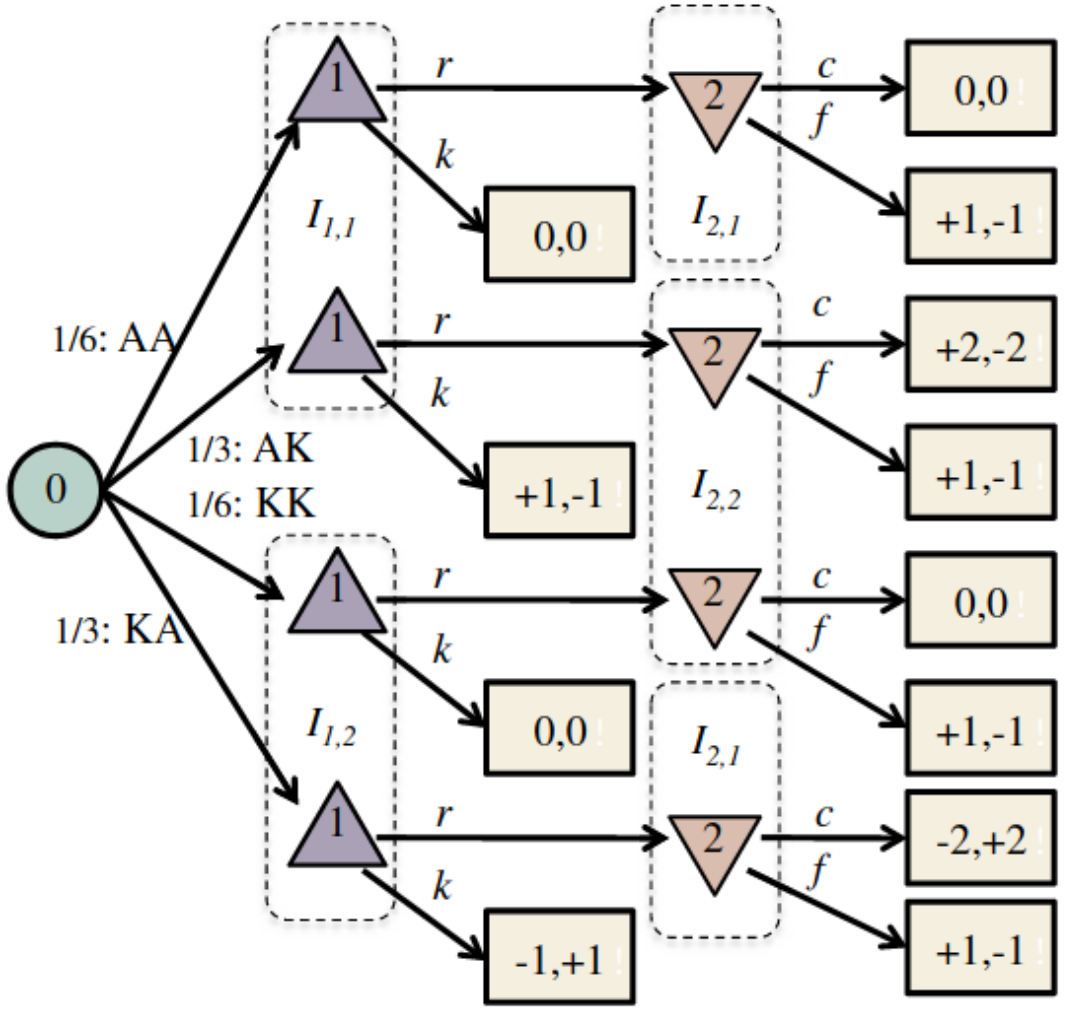
Extensive to Matrix Form



	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

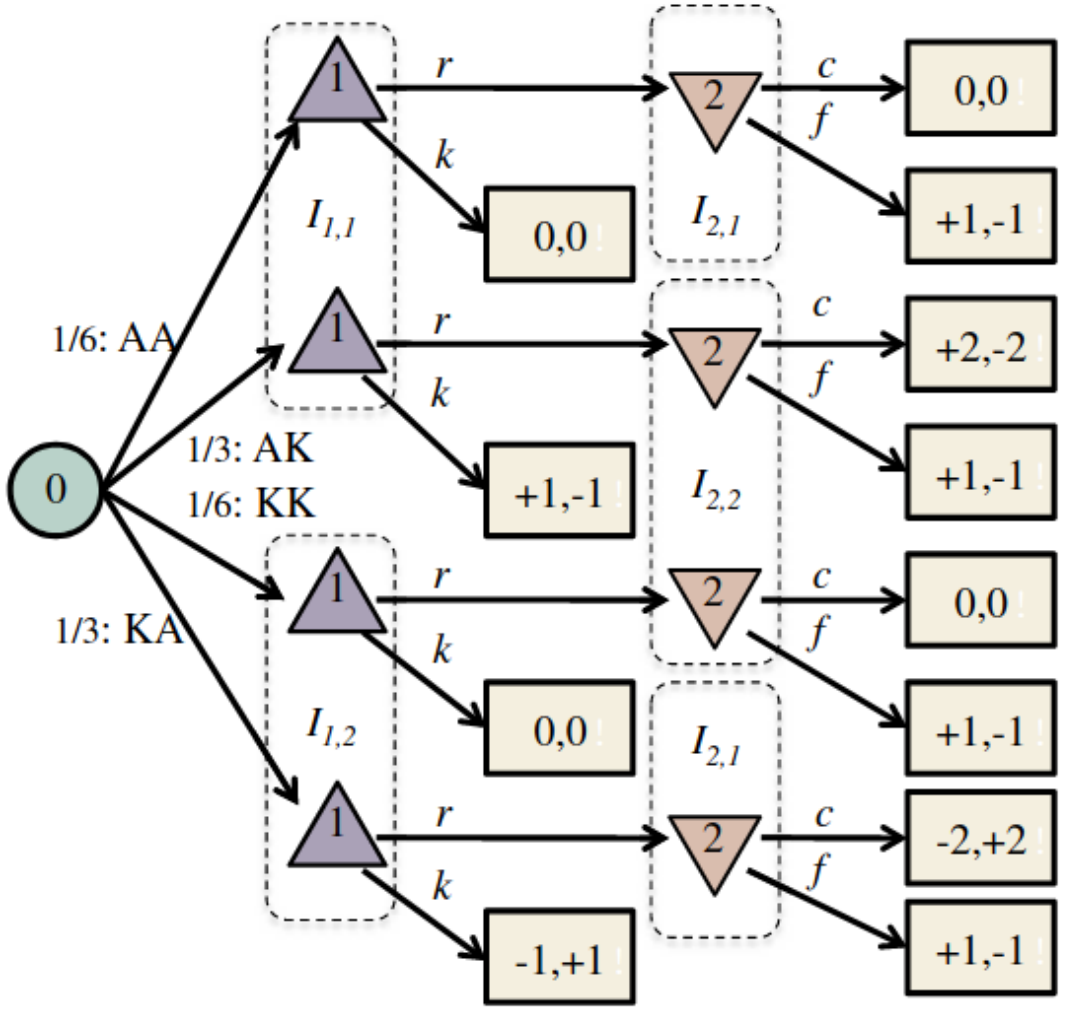
Exponential in number of info states!

Counterfactual Regret Minimization



Counterfactual Regret Minimization

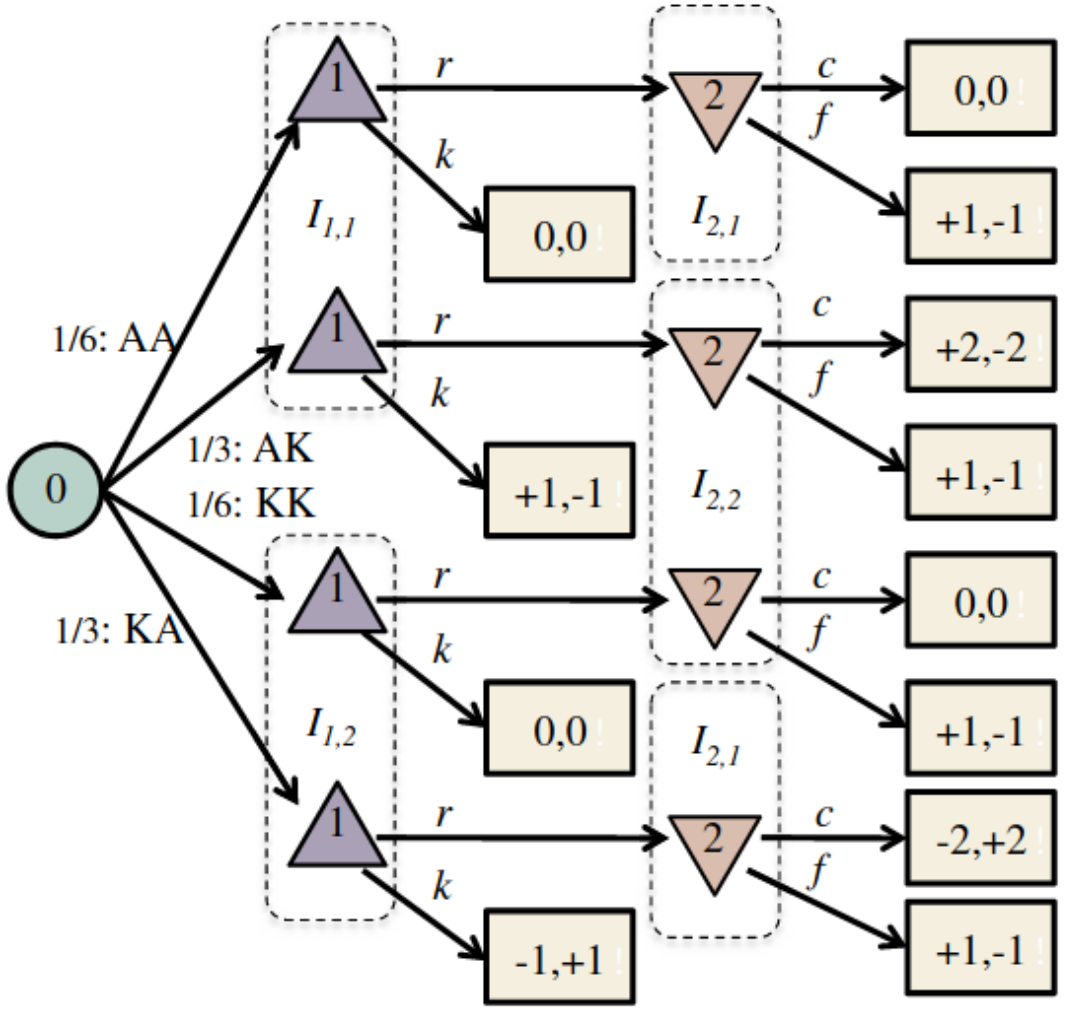
1: Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
2: Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
3: Initialize initial profile: $\sigma^1(I, a) \leftarrow 1/|A(I)|$



Counterfactual Regret Minimization

1: Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
2: Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
3: Initialize initial profile: $\sigma^1(I, a) \leftarrow 1/|A(I)|$

32: **function** Solve():
33: **for** $t = \{1, 2, 3, \ldots, T\}$ **do**
34: **for** $i \in \{1, 2\}$ **do**
35: CFR($\emptyset$, i, t, 1, 1)
36: **end for**
37: **end for**



Counterfactual Regret Minimization

Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

function Solve():
for $t = \{1, 2, 3, \ldots, T\}$ **do**
  for $i \in \{1, 2\}$ **do**
    CFR($\emptyset$, i, t, 1, 1)
  end for
end for

function CFR(h, i, t, π_1, π_2):
if h is terminal **then**
  return $u_i(h)$
else if h is a chance node **then**
  Sample a single outcome $a \sim \sigma_c(h, a)$
  return CFR(ha, i, t, π_1, π_2)
end if
Let I be the information set containing h.
$v_\sigma \leftarrow 0$
$v_{\sigma_{I \to a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
  if $P(h) = 1$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, $\sigma^t(I,a) \cdot \pi_1$, π_2)
  else if $P(h) = 2$ **then**
    $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, π_1, $\sigma^t(I,a) \cdot \pi_2$)
  end if
  $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I \to a}}[a]$
end for
if $P(h) = i$ **then**
  for $a \in A(I)$ **do**
    $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I \to a}}[a] - v_\sigma)$
    $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
  end for
  $\sigma^{t+1}(I) \leftarrow$ regret-matching values
end if
return v_σ