Extensive Form Games

Lecture 7

Lecture Overview

1. Perfect-Information Extensive-Form Games
2. Subgame Perfection
3. Backward Induction

Introduction

- The normal form game representation does not incorporate any notion of sequence, or time, of the actions of the players
- The extensive form is an alternative representation that makes the temporal structure explicit.
- Two variants:
  - perfect information extensive-form games
  - imperfect-information extensive-form games

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N is a set of n players

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A is a (single) set of actions

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
  - Choice nodes: H is a set of non-terminal choice nodes

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
    - Choice nodes: H
    - Action function: $\chi : H \to 2^A$ assigns to each choice node a set of possible actions

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
  - Choice nodes: H
  - Action function: $\chi : H \to 2^A$
  - Player function: $\rho : H \to N$ assigns to each non-terminal node h a player $i \in N$ who chooses an action at h

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
    - Choice nodes: H
    - Action function: $\chi : H \to 2^A$
    - Player function: $\rho : H \to N$
- Terminal nodes: Z is a set of terminal nodes, disjoint from H

Definition

A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

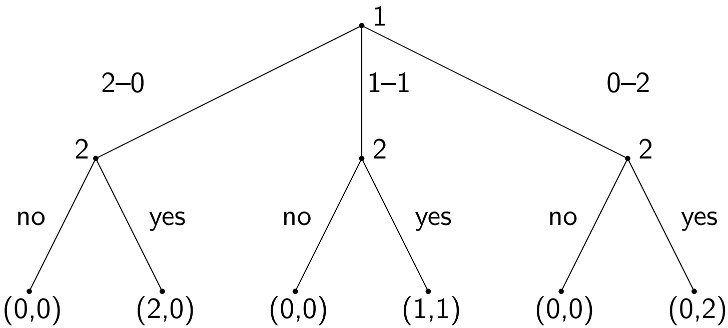
- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
  - Choice nodes: H
  - Action function: $\chi : H \rightarrow 2^A$
  - Player function: $\rho : H \rightarrow N$
- Terminal nodes: Z
- Successor function: $\sigma : H \times A \rightarrow H \cup Z$ maps a choice node and an action to a new choice node or terminal node such that for all $h_1, h_2 \in H$ and $a_1, a_2 \in A$, if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and $a_1 = a_2$
  - The choice nodes form a tree, so we can identify a node with its history.

Definition

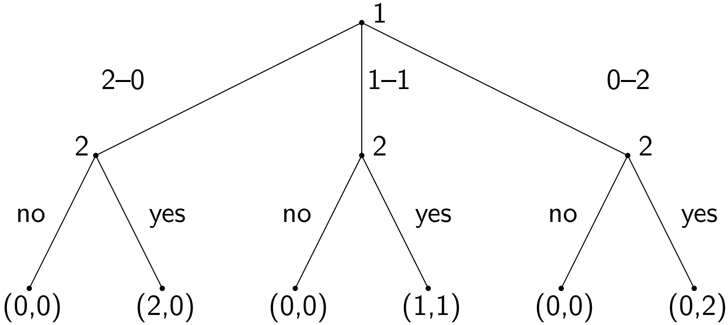
A (finite) perfect-information game (in extensive form) is defined by the tuple $(N, A, H, Z, \chi, \rho, \sigma, u)$, where:

- Players: N
- Actions: A
- Choice nodes and labels for these nodes:
  - Choice nodes: H
  - Action function: $\chi : H \to 2^A$
  - Player function: $\rho : H \to N$
- Terminal nodes: Z
- Successor function: $\sigma : H \times A \to H \cup Z$
- Utility function: $u = (u_1, \ldots, u_n)$; $u_i : Z \to \mathbb{R}$ is a utility function for player i on the terminal nodes Z

Example: the sharing game



Example: the sharing game



Play as a fun game, dividing 100 dollar coins. (Play each partner only once.)

Pure Strategies

- In the sharing game (splitting 2 coins) how many pure strategies does each player have?

Pure Strategies

- In the sharing game (splitting 2 coins) how many pure strategies does each player have?
  - player 1: 3; player 2: 8

Pure Strategies

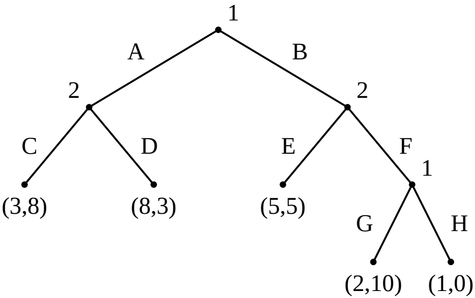
- In the sharing game (splitting 2 coins) how many pure strategies does each player have?
  - player 1: 3; player 2: 8
- Overall, a pure strategy for a player in a perfect-information game is a complete specification of which deterministic action to take at every node belonging to that player.

Definition (pure strategies)

Let $G = (N, A, H, Z, \chi, \rho, \sigma, u)$ be a perfect-information extensive-form game. Then the pure strategies of player i consist of the cross product

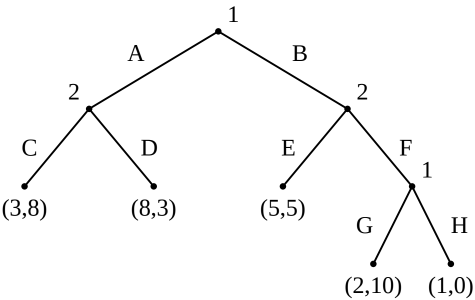
$$\bigtimes_{h \in H, \rho(h)=i} \chi(h)$$

Pure Strategies Example



What are the pure strategies for player 2?

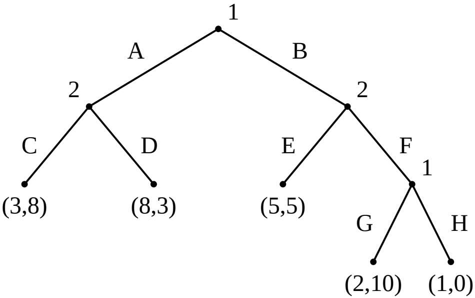
Pure Strategies Example



What are the pure strategies for player 2?

- $S_2 = \{(C, E); (C, F); (D, E); (D, F)\}$

Pure Strategies Example

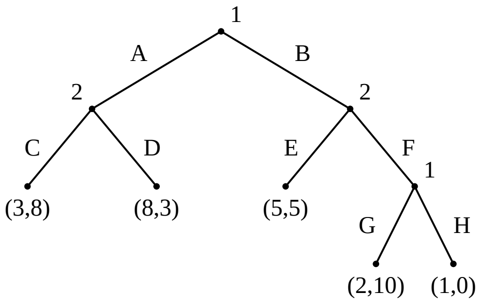


What are the pure strategies for player 2?

- $S_2 = \{(C, E); (C, F); (D, E); (D, F)\}$

What are the pure strategies for player 1?

Pure Strategies Example



What are the pure strategies for player 2?

- $S_2 = \{(C, E); (C, F); (D, E); (D, F)\}$

What are the pure strategies for player 1?

- $S_1 = \{(B, G); (B, H), (A, G), (A, H)\}$
- This is true even though, conditional on taking A, the choice between G and H will never have to be made

Nash Equilibria

Given our new definition of pure strategy, we are able to reuse our old definitions of:

- mixed strategies
- best response
- Nash equilibrium

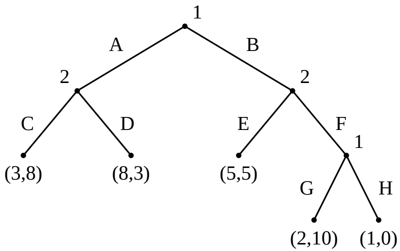
Theorem

Every perfect information game in extensive form has a PSNE

This is easy to see, since the players move sequentially.

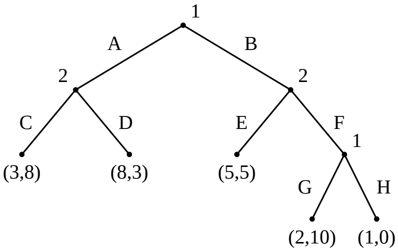
Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can "convert" an extensive-form game into normal form



Induced Normal Form

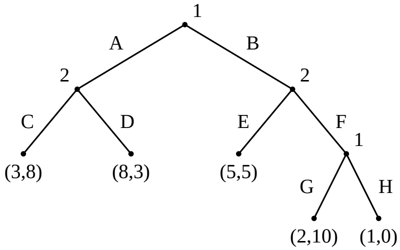
- In fact, the connection to the normal form is even tighter
  - we can “convert” an extensive-form game into normal form



	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can "convert" an extensive-form game into normal form

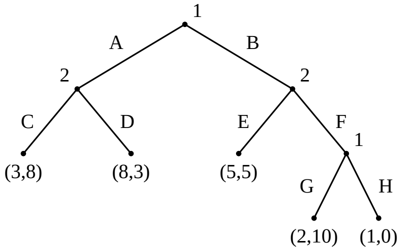


	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

- this illustrates the lack of compactness of the normal form
  - games aren't always this small
  - even here we write down 16 payoff pairs instead of 5

Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can "convert" an extensive-form game into normal form

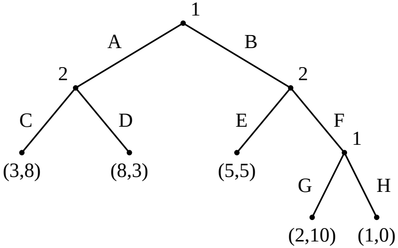


	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

- while we can write any extensive-form game as a NF, we can't do the reverse.
  - e.g., matching pennies cannot be written as a perfect-information extensive form game

Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can “convert” an extensive-form game into normal form

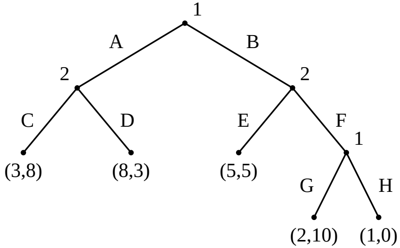


	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

- What are the (three) pure-strategy equilibria?

Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can "convert" an extensive-form game into normal form

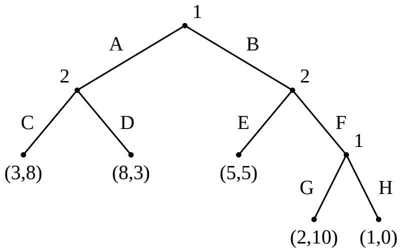


	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

- What are the (three) pure-strategy equilibria?
  - $(A,G),(C,F)$
  - $(A,H),(C,F)$
  - $(B,H),(C,E)$

Induced Normal Form

- In fact, the connection to the normal form is even tighter
  - we can "convert" an extensive-form game into normal form



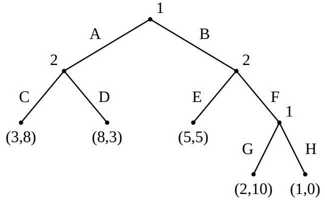
	CE	CF	DE	DF
AG	$3,8$	$3,8$	$8,3$	$8,3$
AH	$3,8$	$3,8$	$8,3$	$8,3$
BG	$5,5$	$2,10$	$5,5$	$2,10$
BH	$5,5$	$1,0$	$5,5$	$1,0$

- What are the (three) pure-strategy equilibria?
  - $(A,G),(C,F)$
  - $(A,H),(C,F)$
  - $(B,H),(C,E)$

Lecture Overview

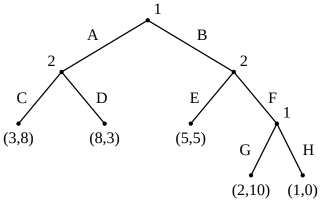
1. Perfect-Information Extensive-Form Games
2. Subgame Perfection
3. Backward Induction

Subgame Perfection



- There's something intuitively wrong with the equilibrium $(B, H), (C, E)$
  - Why would player 1 ever choose to play H if he got to the second choice node?
    - After all, G dominates H for him

Subgame Perfection



- There's something intuitively wrong with the equilibrium $(B, H), (C, E)$
  - Why would player 1 ever choose to play H if he got to the second choice node?
    - After all, G dominates H for him
  - He does it to threaten player 2, to prevent him from choosing F, and so gets 5
    - However, this seems like a non-credible threat
    - If player 1 reached his second decision node, would he really follow through and play H?

Formal Definition

Definition (subgame of G rooted at h)

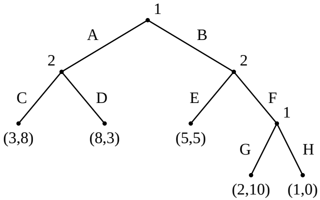
The subgame of G rooted at h is the restriction of G to the descendents of H.

Definition (subgames of G)

The set of subgames of G is defined by the subgames of G rooted at each of the nodes in G.

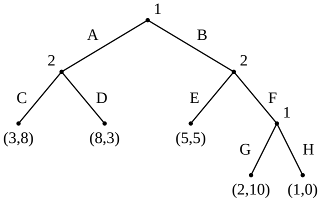
- s is a subgame perfect equilibrium of G iff for any subgame G' of G, the restriction of s to G' is a Nash equilibrium of G'
- Notes:
  - since G is its own subgame, every SPE is a NE.
  - this definition rules out "non-credible threats"

Which equilibria are subgame perfect?



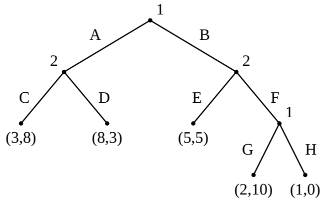
- Which equilibria from the example are subgame perfect?
  - $(A, G), (C, F)$:
  - $(B, H), (C, E)$:
  - $(A, H), (C, F)$:

Which equilibria are subgame perfect?



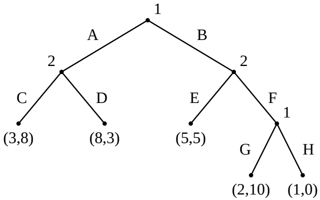
- Which equilibria from the example are subgame perfect?
  - $(A, G), (C, F)$: is subgame perfect
  - $(B, H), (C, E)$:
  - $(A, H), (C, F)$:

Which equilibria are subgame perfect?



- Which equilibria from the example are subgame perfect?
  - $(A, G), (C, F)$: is subgame perfect
  - $(B, H), (C, E)$: (B, H) is an non-credible threat; not subgame perfect
  - $(A, H), (C, F)$:

Which equilibria are subgame perfect?



- Which equilibria from the example are subgame perfect?
  - $(A, G), (C, F)$: is subgame perfect
  - $(B, H), (C, E)$: (B, H) is an non-credible threat; not subgame perfect
  - $(A, H), (C, F)$: (A, H) is also non-credible, even though H is "off-path"

Lecture Overview

1. Perfect-Information Extensive-Form Games
2. Subgame Perfection
3. Backward Induction

Computing Subgame Perfect Equilibria

Idea: Identify the equilibria in the bottom-most trees, and adopt these as one moves up the tree

function BACKWARDINDUCTION (node h) **returns** $u(h)$
if $h \in Z$ **then**
  return $u(h)$
$best\_util \leftarrow -\infty$
forall $a \in \chi(h)$ **do**
  $util\_at\_child \leftarrow$ BACKWARDINDUCTION$(\sigma(h, a))$
  if $util\_at\_child_{\rho(h)} > best\_util_{\rho(h)}$ **then**
    $best\_util \leftarrow util\_at\_child$
return $best\_util$

- $util\_at\_child$ is a vector denoting the utility for each player
- the procedure doesn't return an equilibrium strategy, but rather labels each node with a vector of real numbers.
  - This labeling can be seen as an extension of the game's utility function to the non-terminal nodes
  - The equilibrium strategies: take the best action at each node.

Computing Subgame Perfect Equilibria

Idea: Identify the equilibria in the bottom-most trees, and adopt these as one moves up the tree

function BACKWARDINDUCTION (node h) **returns** $u(h)$
if $h \in Z$ **then**
  return $u(h)$
$best\_util \leftarrow -\infty$
forall $a \in \chi(h)$ **do**
  $util\_at\_child \leftarrow$ BACKWARDINDUCTION$(\sigma(h, a))$
  if $util\_at\_child_{\rho(h)} > best\_util_{\rho(h)}$ **then**
    $best\_util \leftarrow util\_at\_child$
return $best\_util$

- For zero-sum games, BACKWARDINDUCTION has another name: the minimax algorithm.
  - Here it's enough to store one number per node.
  - It's possible to speed things up by pruning nodes that will never be reached in play: "alpha-beta pruning".

```
function AlphaBetaPruning (node h, real α, real β) returns u_1(h)
if h ∈ Z then
    return u_1(h)                                         // h is a terminal node
best_util ← (2ρ(h) − 3) × ∞                              // −∞ for player 1; ∞ for player 2
forall a ∈ χ(h) do
    if ρ(h) = 1 then
        best_util ← max(best_util, AlphaBetaPruning(σ(h, a), α, β))
        if best_util ≥ β then
            return best_util
        α ← max(α, best_util)
    else
        best_util ← min(best_util, AlphaBetaPruning(σ(h, a), α, β))
        if best_util ≤ α then
            return best_util
        β ← min(β, best_util)
return best_util
```