Game Trees and Sequential Moves

Game Theory · Module 3 — Extensive Form Games

Node 3.1

Position in Knowledge Graph

System: Game Theory **Structural Domain:** Extensive Form Games **Node:** 3.1

The strategic form of Module 1 treats a game as a single simultaneous-choice matrix. This is a conscious flattening: most real strategic situations unfold over time, with players moving in sequence, sometimes observing past moves, sometimes not, sometimes with chance events interleaved. The **extensive form** restores this temporal and informational structure. A game tree is a rooted tree whose nodes are decision points, whose edges are actions, and whose leaves are payoff outcomes. This lesson defines the tree formally and sets the stage for information sets (3.2), sequential strategies (3.3), backward induction (3.4), and the refinement theory of subgame perfection (3.5). Everything about poker — the entire betting tree of a hand — lives in the extensive form.

Formal Definition

An **extensive form game** is a tuple $\Gamma = (N, T, P, A, u)$ where:

- $N = \{1, 2, \ldots, n\}$ is the set of players (plus possibly a special "chance" player 0).
- T is a finite rooted tree with root t_0 and leaves Z. Interior nodes are **decision nodes**; leaves are **terminal nodes**.
- $P : T \setminus Z \to N \cup \{0\}$ assigns each decision node to the player who moves there. $P(t) = 0$ means "chance moves" — outcomes at t are drawn from a fixed probability distribution.
- $A(t)$ is the set of actions available at decision node t. Edges from t are labeled with elements of $A(t)$.
- $u = (u_1, \ldots, u_n)$ where each $u_i : Z \to \mathbb{R}$ gives player i's payoff at each terminal node.

A **path** from root to leaf is a sequence of nodes $t_0, t_1, \ldots, t_k$ where each t_{j+1} is a child of t_j, and $t_k \in Z$. A path corresponds to a complete play of the game.

A **perfect-information** extensive-form game is one where, at each decision node t, the acting player knows the entire history of past moves — i.e., knows t itself. Module 3.2 generalizes to **imperfect information** using *information sets*.

A game with chance nodes ($P(t) = 0$) is a **game with chance moves**; the distribution at a chance node is part of the game specification.

Intuition

Picture the game as a decision tree you can walk through. Starting from the root, the designated player chooses an action, producing a new node; then the next player moves, and so on, until a leaf is reached. At each leaf, the payoffs to all players are written down.

In a perfect-information game, each player sees everything — every past move is visible. Chess is the canonical example (ignoring time pressure): both players see the full history of the game at every move. Backward induction (Node 3.4) solves such games by working from the leaves to the root.

Chance nodes introduce randomness. In poker, the dealing of hole cards is a chance node; a subsequent betting decision by player 1 is a decision node; the dealing of the flop is another chance node; a betting decision by player 2 is another decision node, and so on. The tree alternates between player and chance nodes, with payoffs only at the terminal (showdown or fold) leaves.

The tree is rigid about order. Every play of the game is a sequence, not a set of simultaneous choices. Even "simultaneous" games can be written in the extensive form by using information sets (Node 3.2) to hide the order of moves from one of the players.

Structural Role

Game trees are the structural foundation for every subsequent dynamic concept:

- **3.1 ⇒ 3.2.** Information sets partition decision nodes by what the acting player knows.
- **3.1 ⇒ 3.3.** Strategies in extensive form assign an action at every information set.
- **3.1 ⇒ 3.4.** Backward induction computes optimal play by recursing from leaves to root.
- **3.1 ⇒ 3.5.** Subgame perfect equilibrium is Nash equilibrium of the extensive form that survives backward-induction-like pruning.
- **3.1 ⇒ 4.1.** Bayesian games add type-level chance nodes at the root.
- **3.1 ⇒ 5.1.** Repeated games are infinite trees that unfold over time with each stage.
- **3.1 ⇒ 9.3.** Counterfactual regret minimization operates on the extensive-form tree, computing regret at each information set.

Without the extensive form, we could not distinguish credible from incredible threats, could not define "what a player knows when they move," and could not formalize poker as a game. The tree is the unifying data structure for Modules 3, 4, and 9.

Mathematical Development

Paths and histories

A **history** at node t is the sequence of actions taken to reach t from the root. In perfect-information games, the history uniquely identifies t. In imperfect-information games, several histories may share the same node-ID (they become indistinguishable for the acting player).

Ex-ante strategy of player i: a function assigning an action to every decision node owned by i (perfect information) or to every information set of i (imperfect information, Node 3.2).

Chance nodes and probability distributions

At a chance node t with $P(t) = 0$, a distribution $p_t \in \Delta(A(t))$ is given as part of the game description. When computing expected payoffs, one averages over chance draws. In a finite perfect-information game with chance, the expected payoff of a strategy profile is

$$\mathbb{E}[u_i \mid s] = \sum_{z \in Z} \Pr[z \mid s] \cdot u_i(z),$$

where $\Pr[z \mid s]$ is the product of chance probabilities along the path to z induced by s.

Tree structure: Kuhn's axiomatization

An extensive-form game is a rooted tree (connected, acyclic graph). Each non-terminal node has a player assignment, a set of actions, and a set of child nodes in bijection with its actions. The leaves carry payoff vectors. This structure is minimal and combinatorial — no topology, no calculus, just a finite data structure.

Kuhn (1953) formalized this structure and proved several foundational results, including Kuhn's theorem (Node 1.6) on the equivalence of mixed and behavioral strategies under perfect recall.

Branching and size

A depth-d tree with branching factor b has $O(b^d)$ leaves. Game trees for real games are enormous:

- **Tic-tac-toe:** ~255,000 leaves (fully solvable).
- **Checkers:** ~$5 \times 10\hat{}\{20\}$ reachable positions (solved by Schaeffer 2007).
- **Chess:** ~$10\hat{}\{46\}$ positions, well beyond exact solution.
- **Go:** ~$10\hat{}\{170\}$ positions.
- **HUNL Poker:** ~$10\hat{}\{161\}$ information sets (after reasonable abstraction), solved approximately by Libratus/DeepStack 2017.

Backward induction preview

For finite perfect-information games, **backward induction** computes an optimal strategy by recursing from leaves:

1. At each terminal node, the payoff is known.
2. At a decision node whose children are all terminal, the acting player chooses the action maximizing their payoff.
3. Replace the decision node with the payoff of the optimal child.
4. Repeat until the root is reached.

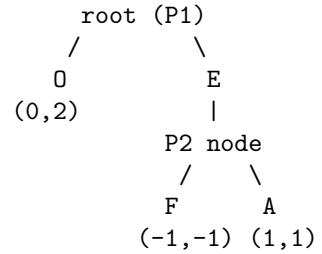
The result is the **backward induction equilibrium** — a specific pure strategy profile that is a Nash equilibrium of the game and, moreover, is subgame perfect (Node 3.5).

Worked Example

Two-stage entry game

Player 1 (Entrant) decides to Enter (E) or Stay Out (O). If Out, payoff $(0, 2)$. If Enter, player 2 (Incumbent) decides to Fight (F) or Accommodate (A). Payoffs: $(E, F) \to (-1, -1); (E, A) \to (1, 1)$.

Tree structure.



Backward induction. At the P2 node, P2 prefers A ($1 > -1$); replace the P2 node with payoff $(1, 1)$. At the root, P1 compares O (0) to E (1); prefers E. Backward induction outcome: (E, A) with payoff $(1, 1)$.

Contrast with normal form. The normal form (Node 1.6) has two Nash equilibria: (E, A) and (O, F). The second relies on the incredible threat "Incumbent fights." Backward induction eliminates it because it

never reaches the P2 node in equilibrium path, but the threat would be suboptimal if the node *were* reached. Subgame perfection (3.5) formalizes this distinction.

A three-level tree

P1 moves, then P2 (observing P1), then P1 again (observing both past moves). Each player has two actions. The tree has 8 leaves. Enumerating pure strategies: P1 has $2 \times 2^2 = 8$ pure strategies (action at root $\times$ action at each of 2 follow-up nodes); P2 has $2^2 = 4$. Normal form is 8×4. For any specific payoff assignment, backward induction gives a unique outcome in generic position.

Poker Anchor

Concept mapping

Formal object	Poker instantiation
Root of game tree	Start of a hand — blinds posted, seats labeled
Chance node (at root)	Hole card dealing — each player receives 2 private cards
Decision node	A player's action point on some street: check/bet/call/fold/raise, and bet sizes
Chance node (mid-game)	Flop, turn, river dealing
Terminal node	Showdown or fold — the hand ends and chips are distributed
Payoff at leaf	Chip delta for each player (winner takes pot; losers pay their contribution)
Path from root to leaf	One complete hand of poker

The entire betting tree of a poker hand is an extensive-form game. Every fact about solver output, every hand-history review, every GTO drill — all of it lives in the tree. The tree structure is what makes poker game theory work.

Concrete situation

HU no-limit hold'em, 100 BB effective, single-raised pot. Sketch the first few levels of the tree:

1. Root: chance node, deals hole cards. Branching factor $= \binom{52}{2}\binom{50}{2} \approx 1.7 \times 10^6$.
2. Depth 1: BTN decision. Actions: Fold, Limp (size 1x), Raise 2x, Raise 2.5x, Raise 3x, ..., All-in. For a typical solver abstraction, ~5 actions.
3. Depth 2 (if BTN raised): BB decision. Actions: Fold, Call, Raise (several sizes). ~5 actions.
4. Depth 3 (if BB called): chance node, deals flop. Branching $= \binom{48}{3} = 17296$.
5. Depth 4: first postflop decision. ~5 actions. ... and so on through turn, river, and each betting round.

By the time you reach the river with significant betting history, the path has ~10 decision nodes and ~3 chance nodes. Each information set has many histories mapped to it (hole cards and positional info create the indistinguishability). The total tree size is astronomical, but the *structure* is the same as any other extensive-form game: rooted tree, decision/chance nodes, terminal leaves, payoffs.

What this understanding buys you

"Solving a spot" means solving a subtree. When a solver tool asks for "a specific spot" (e.g., "turn after flop c-bet call"), what it's doing is taking the *subtree* rooted at that point and solving that subtree independently, assuming the parent game context is fixed. This is legitimate because of subgame perfection (Node 3.5): optimal play in a subtree depends only on the subtree, given consistent beliefs at its root.

Hand histories are walks in the tree. Every hand you play traces a single root-to-leaf path through the game tree. Hand review amounts to asking: "at each of my decision nodes along this path, was I taking the best-response action against my opponent's equilibrium distribution over the rest of the subtree?" This is a node-by-node inspection of the actual walk.

Explosion is the real problem. The reason poker is hard is that the tree is enormous. Every solver technique — abstraction, subgame re-solving, CFR iteration — is a method of managing the astronomical size of the tree without literally traversing it. Node 9.3 (CFR) is the central algorithm here.

Cross-Links

- **1.1 Players, strategies, payoffs** — the normal form that extensive form will flatten to.
- **1.6 Normal form as universal representation** — the flattening operation.
- **3.2 Information sets** — how imperfect information is modeled in the tree.
- **3.3 Strategies in extensive form** — functions from information sets to actions.
- **3.4 Backward induction** — the solution algorithm for perfect-information trees.
- **3.5 Subgame perfect equilibrium** — the refinement of Nash in extensive form.
- **9.3 Counterfactual regret minimization** — the algorithm that exploits tree structure.
- **AI systems — game tree search** — the CS analogue.

Structural Compression

Invariant: A sequential strategic situation is represented as a rooted tree where interior nodes are player or chance decisions, edges are actions, and leaves are payoff outcomes. Every play of the game is a root-to-leaf path, and every payoff is computed at a leaf.

Mechanism: Finite tree data structure with player labels on decision nodes, action labels on edges, probability distributions on chance nodes, and real-valued payoff vectors on leaves.

Cross-System Echo: Game trees are the game-theoretic form of computation trees (CS theory), decision trees (machine learning), ancestry trees (phylogenetics), and search trees (AI). In each case, a rooted tree represents a sequence of choices leading to a terminal outcome; the specific structure makes different questions tractable.

Exercises

1. Draw the complete game tree for Rock–Paper–Scissors, treating it as simultaneous — what device must you use to represent the simultaneity on a tree? (Preview of information sets.)
2. Draw the game tree for the two-stage entry game in the worked example and verify the backward-induction outcome.
3. Count the pure strategies of player 1 and player 2 in a two-stage perfect-information game where each player moves once and each has 3 actions.
4. Construct a 3-level perfect-information tree and compute its backward induction equilibrium. Verify that the backward induction profile is a Nash equilibrium of the normal form.
5. Describe the game tree of a one-street poker game where P1 chooses to Bet or Check, and if Bet, P2 chooses to Call or Fold. Include a chance node at the root dealing a hand category ({strong, weak}) to P1. How many leaves are there?
6. Show that the size of the game tree (number of leaves) can be exponential in the depth of the tree even when the branching factor is small.

7. **Argue, structurally,** why the extensive form is a strictly more expressive representation of a strategic situation than the normal form. Identify a specific strategic fact (not just "the order of moves") that is directly visible in the tree but invisible in the flattened matrix.

Game Theory · Module 3 — Extensive Form Games · Node 3.1
Concepts: graph-theory, probability-space
Prerequisites: 1.1, 1.6
Enables: 3.2, 3.3, 3.4, 3.5

hbar.university — own.your.cognition