Module 4: Game Abstraction & Poker Case Studies

CSCE 631 — Algorithmic Game Theory meets LLMs

Week 4: June 16 (Mon) – June 20 (Fri)
(Juneteenth June 19 — no classes)

Contents

Learning Objectives

By the end of this module, you should be able to:

1. Explain why game abstraction is necessary for large imperfect-information games and distinguish between information abstraction and action abstraction.
2. Define exploitability as a quality metric for abstraction and describe how abstraction pathologies can cause worse performance than the abstraction's granularity would suggest.
3. Describe the blueprint + real-time subgame solving architecture used in Libratus and Pluribus.
4. Explain safe subgame solving and why naive subgame solving is unsafe (can increase exploitability).
5. Compare the Libratus (heads-up) and Pluribus (multiplayer) approaches and identify the key innovations of each.

1 Lecture 15: Game Abstraction I (66 min)

The games studied in Modules 1–3—normal-form games, small extensive-form games, Kuhn poker—are small enough that algorithms such as CFR can operate directly on the full game tree. Real-world imperfect-information games are not so cooperative. Texas Hold'em poker, the canonical benchmark, has approximately 10^{161} game states. No algorithm can enumerate, let alone solve, a game of that size. *Game abstraction* is the principled response: compress the original game into a smaller *abstract game* that is tractable, solve the abstract game (e.g., with CFR), and then map the resulting strategy back to the original game. This lecture introduces the two main dimensions of abstraction and the subtle ways in which abstraction can fail.

1.1 The Scale Problem

The sheer size of real poker games motivates everything in this module. Heads-up no-limit Texas Hold'em (HUNL) has roughly 10^{161} game states—far beyond the reach of any direct solver. Even heads-up *limit* Hold'em, which was "essentially solved" by Bowling et al. (2015), has approximately 10^{14} information sets and required years of computation. The gap between what we can solve exactly and what we want to solve drives the need for abstraction: we must find a way to reduce the game to a solvable size while preserving enough strategic structure that the resulting strategy performs well in the original game.

1.2 Information Abstraction

Definition: Information Abstraction

An **information abstraction** groups information sets in the original game into coarser equivalence classes. Two information sets are merged if the corresponding private states are deemed "similar" according to some metric (e.g., hands with similar strength are grouped together). The result is a smaller game with fewer information sets. Formally, an information abstraction is a mapping $\alpha\colon \mathcal{I} \to \hat{\mathcal{I}}$ from the original information sets to abstract information sets, with $|\hat{\mathcal{I}}| \leq |\mathcal{I}|$.

Information abstraction is the primary tool for reducing game size. In poker, it typically groups hands that have similar winning probabilities, hand potential, or equity distributions into the same "bucket." The player then treats all hands in the same bucket identically, playing the same mixed

strategy for each. The number of buckets controls the granularity of the abstraction: fewer buckets yield a smaller game but coarser strategic distinctions.

1.3 Action Abstraction

Definition: Action Abstraction

An **action abstraction** restricts the set of actions available at each decision point. In no-limit poker, a player can bet any integer number of chips; an action abstraction might allow only a small set of predefined bet sizes (e.g., half-pot, full pot, all-in). This reduces the branching factor of the game tree from potentially thousands of actions per node to a handful.

Action abstraction is essential in no-limit games where the continuous bet-sizing space would otherwise produce an unmanageable branching factor. Even a modest restriction—allowing, say, four bet sizes per decision point instead of hundreds—can reduce the game tree by orders of magnitude. The cost is that the abstract strategy may not include the best response to a particular opponent action, since that action may have been removed from consideration.

1.4 Exploitability

Definition: Exploitability

The **exploitability** of a strategy σ in a two-player zero-sum game is the gap between the strategy's expected value and the value of the game:

$$\varepsilon(\sigma) = v^* - v(\sigma),$$

where v^* is the game value (the value achieved by a Nash equilibrium) and $v(\sigma)$ is the value of σ against a worst-case opponent (i.e., against the opponent's best response to σ). A strategy with exploitability zero is a Nash equilibrium. Exploitability measures how far a strategy is from equilibrium and is the proper metric for evaluating abstraction quality.

Head-to-head results against specific opponents can be misleading: a strategy may beat weak opponents decisively while being highly exploitable by a well-chosen counter-strategy. Exploitability, by contrast, measures performance against the *worst case* and is therefore the gold standard for evaluating abstract strategies.

1.5 Abstraction Pathologies

One might expect that refining an abstraction—adding more buckets, splitting merged information sets—should always improve the resulting strategy. This intuition is wrong. Waugh et al. (2009) demonstrated that refining an abstraction can actually *increase* exploitability, a phenomenon called an **abstraction pathology**. The mechanism is subtle: the Nash equilibrium of the abstract game may not map to a good strategy in the full game, and a finer abstract game can produce an equilibrium whose lifted strategy is worse than the one obtained from the coarser abstraction.

This result has important practical implications. It means that abstraction design cannot simply follow the heuristic of "more buckets is better." The quality of an abstraction depends not only on the granularity of the bucketing but also on how well the abstract equilibrium structure aligns with the strategic structure of the full game.

1.6 Perfect Recall and Imperfect Recall

In a **perfect recall** abstraction, every player remembers all of their own past actions and all public information they have observed. Standard CFR convergence guarantees require perfect recall. An **imperfect recall** abstraction allows a player to "forget" certain aspects of their own history (e.g., forgetting the suit of a card after the flop). Imperfect recall abstractions can be significantly smaller, but they break the equivalence between behavioral strategies and mixed strategies (Kuhn's theorem no longer applies), and standard CFR may not converge. Special variants such as CFR-D are needed for imperfect recall settings.

1.7 Lossless and Lossy Abstraction

Definition: Lossless Abstraction

A **lossless abstraction** preserves all strategically relevant information: the Nash equilibria of the abstract game correspond exactly to Nash equilibria of the original game (after lifting). Lossless abstraction is only possible when the game exhibits specific symmetries. In poker, the primary example is **suit isomorphism**: since the suits in a standard deck are strategically interchangeable (no suit dominates another), hands that differ only by a permutation of suits can be merged without any loss.

Definition: Lossy Abstraction

A **lossy abstraction** introduces approximation error: the equilibrium of the abstract game, when lifted back to the original game, is not an exact equilibrium. The central question for lossy abstraction is how much error is introduced and whether bounds on that error can be established.

In practice, almost all abstractions used in large-scale poker solving are lossy. The design goal is to minimize the exploitability of the resulting strategy, balancing abstraction coarseness (which determines computational tractability) against approximation fidelity.

1.8 Strategy Lifting and Sources of Error

Strategy lifting is the process of translating an abstract strategy back to the full game. When a player encounters a hand in the full game, the hand is mapped to its abstract bucket, and the corresponding abstract strategy is played. Similarly, when the player observes an opponent action that is not in the abstract action set, it must be mapped to an abstract action.

There are three distinct sources of approximation error in the abstraction pipeline: (1) *information abstraction error*, from merging strategically distinct hands into the same bucket; (2) *action abstraction error*, from restricting the available actions; and (3) *strategy lifting error*, from the imperfect translation between abstract and full-game strategies. The overall quality of an abstraction-based approach depends on all three.

Common Pitfalls

- Do not assume that finer abstraction is always better—abstraction pathologies are real and well-documented (Waugh et al., 2009).
- The quality of an abstraction depends on both the abstraction itself and the strategy lifting procedure. A good bucketing with a poor lifting method can yield a highly exploitable

strategy.
- Perfect recall is required for standard CFR convergence. Imperfect recall abstractions require specialized algorithm variants such as CFR-D.

2 Lecture 16: Game Abstraction II (69 min)

Lecture 15 introduced the *why* and *what* of game abstraction. This lecture addresses the *how*: what features should we use to measure hand similarity, what algorithms should we use to group hands into buckets, and what pitfalls arise in the evaluation of abstract strategies? The answers draw on techniques from machine learning (clustering), probability theory (earth mover's distance), and game theory (potential-aware reasoning about future rounds).

2.1 Hand Evaluation Features

Effective information abstraction requires features that capture the strategically relevant aspects of a poker hand. A single number like hand rank is far too coarse; multiple complementary features are needed.

Definition: Effective Hand Strength (EHS)

The **Effective Hand Strength (EHS)** of a hand is the probability that the hand wins at showdown, computed by averaging over all possible opponent hands and (if applicable) all possible future community cards. EHS integrates both the current hand strength and the potential for improvement, providing a single scalar summary of a hand's value.

Beyond EHS, several additional features capture aspects of strategic value that a single scalar misses:

- **Hand potential**: the probability that a currently losing hand improves to win (*positive potential*) or that a currently winning hand deteriorates to lose (*negative potential*). Hand potential captures the value of drawing hands—a flush draw with four suited cards has high positive potential even if its current showdown strength is low.
- **Expected hand strength** $E[\mathbf{HS}]$: the expected winning probability over all possible future community cards. This is closely related to EHS but emphasizes the distributional aspect.
- **Hand strength variance**: two hands with the same $E[\mathrm{HS}]$ but different variance have different strategic profiles. A flush draw has high variance (it will either hit or miss), while a made pair has low variance. This distinction affects optimal bet sizing and bluffing frequency.
- **Draw types and outs**: flush draws, straight draws, and the number of remaining cards ("outs") that improve the hand. These categorical features complement the continuous features above.
- **Blockers**: cards in a player's hand that prevent opponents from holding certain combinations. Blocker effects are an advanced feature used in high-level play and become important in sophisticated abstractions.

2.2 Bucketing Algorithms

Given a set of features for each hand, the next step is to partition hands into buckets—groups that will be treated identically by the abstract strategy. Three main approaches are used in practice.

k-Means clustering. Represent each hand by its equity distribution vector (a histogram of winning probabilities against all possible opponent hands), and apply k-means clustering to partition hands

into k groups. This is the most common approach and naturally handles multi-dimensional feature representations.

Quantile bucketing. Sort all hands by a single metric (typically EHS) and assign equal-sized buckets. This approach is simpler and faster than k-means but less expressive: it cannot distinguish hands with the same EHS but different strategic properties (e.g., different variance or draw potential).

Definition: Earth Mover's Distance (EMD)

The **Earth Mover's Distance (EMD)** between two probability distributions P and Q over a metric space measures the minimum "work" required to transform P into Q, where work is defined as the amount of probability mass moved times the distance it is moved. Formally, for discrete distributions over an ordered set:

$$\text{EMD}(P, Q) = \sum_{i=1}^{n} \left| \sum_{j=1}^{i} (P(j) - Q(j)) \right|.$$

EMD is used as the distance metric for clustering hands by their equity distributions. It captures distributional differences that a single summary statistic (like mean equity) misses: two hands with the same mean equity but different shapes (e.g., bimodal vs. unimodal) will have a large EMD.

EMD-based clustering consistently outperforms simpler single-metric approaches in experiments. By measuring the full distributional distance between hands, it produces buckets whose members are genuinely similar in their strategic implications, not merely in their average strength.

2.3 Potential-Aware Abstraction

Potential-aware abstraction (Ganzfried & Sandholm, 2014) goes beyond current hand strength to consider the *distribution over future strengths*. A flush draw and a made pair may have similar current equity, but their distributions over future community cards are very different: the flush draw has high positive potential (it will become very strong if the flush completes) and the made pair has low variance. Potential-aware abstraction clusters hands based on these distributional properties, typically using the full equity distribution vector rather than a single scalar.

This approach significantly reduces exploitability compared to non-potential-aware methods, especially in games with multiple betting rounds where the distinction between "made hands" and "drawing hands" is strategically critical.

2.4 Hierarchical Abstraction

In practice, a common strategy is **hierarchical abstraction**: use a coarse abstraction in early betting rounds (where the game tree is widest and fine distinctions have the least impact) and a finer abstraction in later rounds (where decisions matter most and the remaining game tree is smaller). This provides a practical tradeoff between computational cost and strategic accuracy.

2.5 Public Belief States

Definition: Public Belief State (PBS)

A **public belief state** is the probability distribution over each player's private hands, conditioned on the public history (the sequence of public actions and community cards observed so far). Formally, for a public history h, the PBS specifies, for each player i, a distribution $\beta_i(\cdot \mid h)$ over player i's possible private hands.

Public belief states provide the conceptual bridge between abstraction and real-time solving. In depth-limited solving (Lecture 17), continuation strategies conditioned on public belief states provide estimated game values, enabling the solver to truncate the game tree at a manageable depth without solving all the way to terminal nodes. The PBS is also the natural state representation for neural network–based value function approximation in approaches like Deep CFR.

2.6 Evaluation Pitfalls in Abstraction

Several subtle issues arise when evaluating the quality of abstract strategies:

Strategy fusion. When information sets are merged, the abstract strategy must play the same mixed strategy for all hands in the bucket. If the optimal play differs substantially across the merged hands, this forces a compromise that may be poor for all of them. This phenomenon is called *strategy fusion* and is a fundamental limitation of information abstraction.

Abstraction mismatch. An abstract strategy is computed to be optimal within the abstract game. When deployed in the full game against opponents who can play any action (not just the abstract ones), weaknesses that are invisible within the abstract game may be exposed.

Bucket leakage. Opponents who understand the abstraction boundaries can exploit them by playing hands near the boundary differently, forcing the abstract strategy into situations where the bucket assignment is maximally misleading.

Common Pitfalls

- Do not evaluate abstraction quality solely by head-to-head play—exploitability is the proper metric. A strategy can win against weak opponents but be highly exploitable by a well-chosen adversary.
- EMD-based clustering captures hand-level similarity but does not account for strategic interactions between rounds.
- More buckets is not always better due to abstraction pathologies (Lecture 15). The relationship between abstraction granularity and strategy quality is non-monotonic.

Connection: Project Topic 3: Game Abstraction and Solution Quality

This lecture's material on bucketing algorithms, EMD-based clustering, and potential-aware abstraction provides the technical foundation for Project Topic 3. A project in this area would implement and compare different abstraction schemes on a game larger than Kuhn poker, measuring exploitability as the evaluation metric.

3 Lecture 17: Poker Case Study — Libratus and Pluribus (~69 min)

This lecture traces the arc from early poker bots to superhuman play, focusing on the two landmark systems: Libratus (2017), which defeated top professionals in heads-up no-limit Texas Hold'em, and Pluribus (2019), which extended superhuman performance to six-player no-limit Hold'em. The architectural principles underlying these systems—blueprint computation, safe subgame solving, and depth-limited search—form a general paradigm applicable far beyond poker.

3.1 Action Abstraction Techniques

Two main approaches are used for action abstraction in no-limit poker:

Fixed bet sets restrict betting to a small, predefined set of sizes expressed as fractions of the pot (e.g., half-pot, full pot, all-in). This is simple to implement and produces a manageable game tree, but it may miss strategically important bet sizes. If the optimal play in a given situation involves a three-quarter-pot bet and that size is not in the abstract action set, the strategy is forced to use a suboptimal alternative.

Dynamic ladders define bet sizes that scale with the pot, providing more flexibility than fixed sets. As the pot grows, the available bet sizes grow proportionally, better capturing the strategic structure of no-limit games.

3.2 Action Translation

When an opponent makes a bet that is not in the abstract action set, the system must *translate* it to an abstract action. Two approaches are used:

Deterministic action translation maps the opponent's bet to the nearest abstract action. This is simple but can be exploited: an opponent who knows the abstraction boundaries can choose bet sizes that are maximally misleading after rounding.

Randomized action translation probabilistically splits the opponent's bet between the two nearest abstract actions, with probabilities proportional to the distances. This approach consistently outperforms deterministic translation in experiments, as it prevents the opponent from systematically exploiting the rounding.

3.3 Blueprint Strategies

Definition: Blueprint Strategy

A **blueprint strategy** is a coarse-grained strategy computed offline for the full abstract game using Monte Carlo CFR (MCCFR). The blueprint covers every information set in the abstract game and serves as the starting point for real-time refinement during actual play. Blueprint computation is the most computationally expensive phase: Libratus's blueprint required months of computation on a supercomputer.

The blueprint is not intended to be the final strategy. It provides two essential functions: (1) a default policy for parts of the game tree that are not refined in real time, and (2) boundary conditions (opponent ranges and values) for subgame solving. The quality of the blueprint sets a floor on the quality of the overall system.

3.4 Safe Subgame Solving and Gadget Games

During actual play, the system can improve on the blueprint by re-solving the current subgame with a finer abstraction. However, this must be done carefully.

Naive subgame solving is unsafe. If we replace the strategy in a subgame while keeping the rest of the blueprint fixed, we can actually *increase* exploitability. The reason is that the opponent can best-respond to the combination of the new subgame strategy and the old trunk strategy, exploiting inconsistencies between the two. This is a counterintuitive but critical result.

Definition: Safe Subgame Solving

Safe subgame solving constrains the new subgame strategy so that the opponent receives no more value at the subgame root than they would have received under the blueprint strategy. This ensures that replacing the subgame strategy cannot increase the overall exploitability of the combined strategy.

Definition: Gadget Game

A **gadget game** is an auxiliary game constructed at the root of a subgame to enforce the safety constraint. The gadget encodes the opponent's range (distribution over private hands) at the subgame root and adds an alternative action for the opponent: "opt out and receive the blueprint value." By solving the gadget game, the solver ensures that the new subgame strategy gives the opponent no more value than the blueprint did, while still allowing improvement wherever possible.

Safe subgame solving, implemented via gadget games, is the mechanism that allows real-time refinement without sacrificing the exploitability guarantees established by the blueprint. It preserves the blueprint's exploitability bound while permitting the strategy to adapt to the specific game situation.

3.5 Nested Solving and Depth-Limited Solving

Nested solving applies subgame solving recursively. When the system reaches a subgame boundary during play, it solves that subgame with a finer abstraction. When it later reaches a deeper subgame boundary within the already-refined subtree, it solves again with an even finer abstraction. This telescoping refinement concentrates computational effort on the parts of the game tree that are actually reached during play.

Definition: Depth-Limited Solving

Depth-limited solving truncates the subgame at a fixed depth and uses *continuation strategies*—precomputed policies for the remaining game—to estimate game values at the truncation point, rather than solving all the way to the terminal nodes. At each leaf node, players choose among k continuation strategies; the solver iteratively adds best-response policies to this set and re-solves. This dramatically reduces the computation per solve while maintaining approximation quality.

Depth-limited solving is essential for real-time play in large games. The combination of nested solving and depth-limited solving enables real-time strategic refinement that would be computationally infeasible with full-depth solving.

3.6 History of Computer Poker

The progression from simple rule-based bots to superhuman agents spans several decades. Key milestones include:

- **2015**: Heads-up limit Hold'em is "essentially solved" by Bowling et al. (*Science*). The resulting strategy is so close to a Nash equilibrium that it is indistinguishable from optimal play over a human lifetime of games.
- **2017**: Libratus defeats four top professionals in heads-up no-limit Hold'em, the first AI to achieve superhuman performance in this game.
- **2019**: Pluribus defeats top professionals in six-player no-limit Hold'em, extending superhuman performance to the multiplayer setting—the first AI to achieve superhuman play beyond two-player zero-sum games.

3.7 Libratus: Three-Module Architecture

Libratus (Brown & Sandholm, 2018) is built on three tightly integrated modules:

1. **Module 1: Blueprint strategy.** A strategy computed via Monte Carlo CFR over an abstracted game that applies both information abstraction (hand bucketing) and action abstraction (restricted bet sizes). The blueprint computation ran for months on the Bridges supercomputer at the Pittsburgh Supercomputing Center.
2. **Module 2: Nested safe subgame solving.** During play, when the opponent reaches a subgame boundary (typically by making a bet not in the blueprint's action abstraction, or upon entering a new betting round), Libratus constructs a gadget game and re-solves the subgame with a finer abstraction. The safety constraint ensures the new strategy is no worse than the blueprint. This process is applied recursively via nested solving.
3. **Module 3: Self-improvement.** After each day of play against the human professionals, Libratus identifies branches of the game tree where opponents found exploitable patterns and adds those branches to the blueprint for overnight re-computation. This self-improvement loop patches weaknesses that the human players discover, making Libratus progressively harder to exploit over the course of the match.

Example: Libratus Results

Libratus defeated four top poker professionals over 120,000 hands of heads-up no-limit Texas Hold'em over 20 days in January 2017, with 99.98% statistical significance ($p = 0.0002$). The winning margin was 147 milli-big-blinds per hand (mbb/hand)—a large margin in professional poker. Each of the four professionals lost individually, ruling out the possibility that Libratus exploited a single player's weakness.

3.8 Pluribus: Multiplayer Poker

Pluribus (Brown & Sandholm, 2019) extends the Libratus paradigm to six-player no-limit Hold'em, confronting several new challenges.

The multiplayer challenge. In two-player zero-sum games, Nash equilibrium is the natural solution concept: it is unique in value, computable (in principle), and guarantees non-exploitability. In multiplayer games, none of these properties hold cleanly. Computing a Nash equilibrium is harder (PPAD-complete even for three players), multiple equilibria with different payoffs can exist, and

equilibrium play is no longer guaranteed to be non-exploitable. Pluribus therefore does not target exact Nash equilibrium but instead aims for a strategy that is empirically strong.

Blueprint computation. Pluribus computes its blueprint via Linear MCCFR (Monte Carlo CFR with linear weighting for both regrets and strategy averaging), which emphasizes later iterations (where the strategy is more refined) over earlier ones. Remarkably, the blueprint computation requires only a single 64-core server with 512 GB RAM (no GPUs) and runs in approximately 8 days, at a cost of about $150 in cloud compute—a dramatic contrast with Libratus's supercomputer requirements.

Depth-limited search. Instead of solving subgames all the way to terminal nodes, Pluribus truncates the search after a limited lookahead and uses continuation strategies (multiple blueprint-derived policies) to estimate game values at the truncation boundary. All players—not just opponents—choose among k continuation strategies at the leaf nodes, and the search iterates: solve the depth-limited subgame, compute a best response to add as a new continuation strategy, and repeat. This converges to a Nash equilibrium in two-player zero-sum games and produces strong strategies empirically in the multiplayer setting.

Selective real-time search. During play, Pluribus initiates real-time search starting from the beginning of the current betting round, keeping its own strategy fixed for actions already taken in that round. Opponents' actual actions (including off-blueprint bet sizes) are incorporated into the subgame model before solving, eliminating the need for reverse action mapping.

Safety in multiplayer. The notion of "safe" subgame solving from the two-player setting does not directly transfer to multiplayer games. Pluribus uses a weaker but practically effective approach: it searches against a range of opponent strategies rather than assuming a single worst-case opponent.

Example: Pluribus Results

Pluribus defeated elite poker professionals in two experimental formats, both at six-player tables. In Experiment 1, a single human professional played against five independent copies of Pluribus; in Experiment 2, five human professionals (selected from a pool of thirteen top players) played alongside one copy of Pluribus. The self-improvement module used in Libratus (Module 3) was not used in Pluribus; the system's strength came entirely from the blueprint and depth-limited real-time search.

3.9 Architectural Insights

Several insights from the Libratus and Pluribus work generalize beyond poker:

- The *abstraction + blueprint + real-time search* paradigm is a general architecture for large imperfect-information games, not a poker-specific trick.
- Real-time subgame solving concentrates computation on the parts of the game tree that are actually reached, making the effective resolution much finer than the blueprint's abstraction.
- Depth-limited solving with continuation strategies enables real-time play in games where solving to terminal nodes is infeasible.
- Self-improvement (Libratus Module 3) was effective in heads-up play but was not needed in Pluribus, suggesting that the blueprint quality and real-time search were sufficient in the multiplayer setting.

Common Pitfalls

- Subgame solving requires a correct *boundary condition*: the opponent's range and values at the subgame root. Getting this wrong makes the solution unsafe—the resulting strategy may be more exploitable than the original blueprint.
- Do not confuse "safe" subgame solving (preserves the exploitability bound) with "optimal" subgame solving. Safe solving may be conservative: it guarantees no worse than the blueprint but does not guarantee the best possible strategy.
- Naive subgame solving—re-solving a subgame without the safety constraint—can increase exploitability. The gadget game construction is essential, not optional.

Connection: Project Topic 2: CFR beyond Kuhn Poker

The blueprint computation in both Libratus and Pluribus uses Monte Carlo CFR with various acceleration techniques (linear weighting, sampling schemes). Project Topic 2 asks you to implement CFR for a game larger than Kuhn poker, directly applying the algorithmic foundations from Module 3 at a scale where abstraction begins to matter.

Connection: Project Topic 3: Game Abstraction and Solution Quality

The abstraction techniques from Lectures 15–16—information abstraction, action abstraction, EHS-based bucketing, EMD-based clustering, and potential-aware abstraction—are the core material for Project Topic 3. A project in this area would implement and evaluate different abstraction schemes, measuring exploitability as the quality metric.

Connection: Project Topic 4: Deep CFR Replication

Deep CFR replaces the tabular regret storage in CFR with neural network function approximation, using public belief states as input. The PBS framework from Lecture 16 and the depth-limited solving approach from Lecture 17 are closely related to the architecture of Deep CFR. Project Topic 4 asks you to replicate or extend the Deep CFR approach.

Key Concepts Summary

Concept	Definition	Why It Matters
Information abstraction	Merge similar information sets	Reduces game to solvable size
Action abstraction	Restrict available actions	Reduces branching factor
Exploitability	Gap from NE value: $\varepsilon(\sigma) = v^* - v(\sigma)$	Measures strategy quality
Abstraction pathology	Finer abstraction $\rightarrow$ worse strategy	Must be careful with abstraction design
Safe subgame solving	New strategy no worse than blueprint	Prevents exploitability increase
Blueprint strategy	Coarse-grained offline strategy via MCCFR	Foundation for real-time refinement

Key Takeaways

1. Game abstraction—both information abstraction and action abstraction—is essential for solving large imperfect-information games. Without it, games like no-limit Texas Hold'em ($\sim 10^{161}$ states) are completely intractable.
2. Abstraction pathologies mean that finer is not always better. The relationship between abstraction granularity and strategy quality is non-monotonic, and exploitability (not head-to-head performance) is the correct evaluation metric.
3. The *blueprint + real-time subgame solving* architecture is the key paradigm. The blueprint provides a coarse default strategy and boundary conditions; real-time solving refines the strategy at the points actually reached during play.
4. Safe subgame solving via gadget games is essential to prevent exploitability increases when refining subgame strategies. Naive subgame solving is unsafe.
5. Libratus (2017) achieved superhuman heads-up play with three modules: blueprint, nested safe subgame solving, and self-improvement. Pluribus (2019) extended this to multiplayer with depth-limited search and a dramatically cheaper blueprint computation.

Suggested Reading

- Brown & Sandholm (2018): "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals" (*Science*).
- Brown & Sandholm (2019): "Superhuman AI for multiplayer poker" (*Science*).
- Gilpin & Sandholm (2007): "Lossless Abstraction of Imperfect Information Games."
- Ganzfried & Sandholm (2014): "Potential-Aware Imperfect-Recall Abstraction with EMD" (AAAI).
- Waugh et al. (2009): "A Practical Use of Imperfect Recall."

Supplementary Materials

The `refs/week4/` directory contains supplementary PDFs from prior course offerings:

- `Lecture_12_Libratus_and_subgame_solving.pdf` — detailed Libratus slides from Fall 2025.
- `pluribus-slides.pdf` — Pluribus presentation slides.
- `science.aao1733.pdf` — Brown & Sandholm (2018), Libratus *Science* paper.
- `science.aay2400.pdf` — Brown & Sandholm (2019), Pluribus *Science* paper.