Action Abstraction and Endgame Solving

Blueprints, Safe Subgame Re-solving, and Practical Pipelines

Intelligent Agents: Computational Game Solving

November 4, 2025

Roadmap

Today:

1. Action abstraction: what, why, and how
2. Blueprint strategies vs. real-time re-solving
3. Safe subgame re-solving: constraints and guarantees
4. Depth-limited solving and terminal evaluation
5. Case studies: Libratus, Pluribus (high-level)

Goal: From coarse offline strategies to robust real-time play in large EFGs.

Motivation: Why Action Abstraction?

Problem: No-Limit betting allows many raise sizes (from min-raise to all-in).

- At each infoset I, $|A(I)|$ can be large (dozens to hundreds).
- Branching factor explodes across streets (rounds).

Action abstraction: Replace $A(I)$ with a smaller set $A'(I) \subseteq A(I)$.

- Fixed representative bet sizes (e.g., fold, call, 0.5 pot, 1.0 pot, all-in).
- Reduces branching; enables CFR on the abstract game.

Trade-off: Smaller $A'(I)$ = faster solve but larger approximation error.

Key Terms (Betting)

- **Pot**: total chips in the middle before the action.
- **Stack**: chips remaining behind a player (max they can lose).
- **Bet size**: amount a player adds; min-raise rules constrain legal actions.
- **Pot-sized bet**: bet equal to current pot (common reference).
- **All-in**: bet entire remaining stack.

Abstraction needs legal mapping: abstract bets must map to legal real bets.

Action Abstraction: Fixed Bet Sets

Design pattern: Use the same action menu across many infosets.

- Example bet menu: $\{\text{fold}, \text{call}, 0.5 \times \text{pot}, 1.0 \times \text{pot}, \text{all-in}\}$.
- Simplicity: easier to implement and lift.
- Downside: may not adapt to context (stack-to-pot ratio, street, texture).

When to use: As a baseline or for early streets in a blueprint.

Action Abstraction: Dynamic Ladders

Idea: Tailor the bet menu to context at infoset I.

- Choose sizes as fractions of pot (e.g., 33%, 75%, 125%).
- Include min-raise and all-in if legal.
- Street-aware menus (more sizes on river where precision matters).

Pros: Better coverage of strategically relevant sizes.
Cons: More complex lifting; irregular action spaces across infosets.

Forward Restriction and Backward Lifting

Forward (build abstract game): $A(I) \to A'(I)$

- Restrict legal actions to $A'(I)$ when solving with CFR.

Backward (during play): $A'(I) \to A(I)$

- Map abstract action to nearest legal real action (rounding).
- Proportional scaling: target bet $= \alpha \times$ pot; round to legal min-max.
- Tie-break rule (e.g., round up to min-raise if below).

Note: Lifting can introduce additional error if mapping is coarse.

Granularity vs. Compute

- More sizes $\Rightarrow$ better approximation of optimal betting, but:
  - Higher branching factor, larger abstract tree.
  - Slower CFR iterations; more memory for regrets/strategies.
- Fewer sizes $\Rightarrow$ faster, but risks:
  - Miss important thresholds (e.g., overbets, block bets).
  - Strategy becomes predictable in lift (bucket leakage).

Rule-of-thumb: 4–6 sizes per street is common in practice.

Interplay: Information vs. Action Abstraction

Co-design matters:

- Coarse information buckets + many bet sizes can still misplay classes of hands.
- Fine buckets + too few bet sizes can underfit value extraction/bluffing.

Practical pattern:

- Early streets: coarser info buckets + fewer bet sizes.
- Later streets: finer info buckets + more bet sizes.

Blueprint vs. Real-Time Re-solving

Blueprint:

- Precomputed strategy in an abstract game (offline CFR).
- Fast lookup at runtime; covers all states coarsely.

Real-time re-solving:

- At decision time, solve a *subgame* conditioned on current public state.
- Use finer abstraction locally; incorporate current ranges and pot.

Goal: Combine speed (blueprint) with accuracy (local refinement).

What is a Subgame?

Public state: The shared (public) history: street, pot, public cards, bet history.
Subgame rooted at public state S:

- All future histories consistent with S.
- Player ranges at S: distributions over private states consistent with S.

Key inputs to re-solving:

- Our range and opponent range at S.
- Pot size, stacks, legal actions (min-raise rules).

Safe Subgame Re-solving: Intuition

Risk: Naively re-solving can make the overall strategy exploitable.
Safe re-solving: Add constraints so the opponent never gets less value than the blueprint guaranteed at S.

- Compute opponent's *counterfactual value bound* at S from blueprint.
- Solve subgame ensuring opponent's value $\geq$ bound (plus margin).

Effect: Local improvement without increasing exploitability.

Opponent Range and Value Bounds

Opponent range at S: $\rho_{-i}(h)$ over their private hands consistent with S.
Blueprint value bound:

- Compute opponent's expected value in the blueprint at S, $V_{-i}^{\text{blue}}(S)$.
- Optionally add *subgame margin* $\delta \geq 0$:

$$V_{-i}^{\text{bound}}(S) \;=\; V_{-i}^{\text{blue}}(S) - \delta.$$

Constraint: Re-solved subgame must not reduce opponent's EV below $V_{-i}^{\text{bound}}(S)$.

Gadget Game for Safe Re-solving (High-Level)

Construction idea:

- At the subgame root, introduce a gadget that lets the opponent:
  - Exit to blueprint continuation with guaranteed value $V_{-i}^{\text{bound}}(S)$, or
  - Enter the re-solved subgame.
- The solver must produce a strategy no worse for the opponent than the exit option.

Outcome: Guarantees safety; if we cannot improve, fall back to blueprint.

Continual Re-solving

Approach: Re-solve at every decision (for both players).

- Update ranges after each observed action.
- Solve a shallow subgame rooted at the new public state.
- Use safe constraints to keep exploitability controlled.

Benefit: Adapts to runout and bet sequence; precision where it matters.

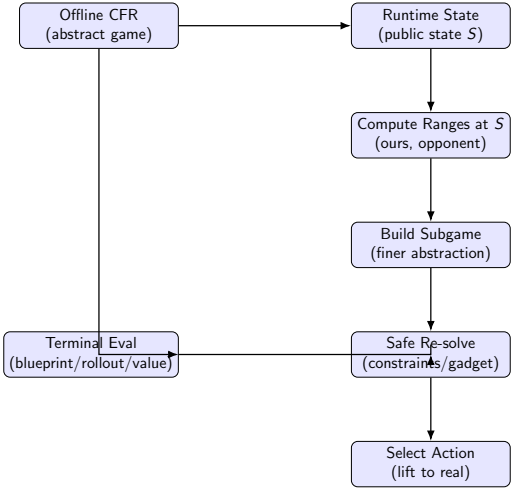
Depth-Limited Subgames and Terminal Evaluation

Practical constraint: Cannot solve to the real end in real time.
Depth-limited approach:

- Expand subgame to a fixed depth or end of street(s).
- At frontier nodes, use a *terminal evaluator*:
  - Blueprint continuation value (tabular lookup).
  - Rollout with a fast default policy.
  - Value function (learned or heuristic).

Trade-off: Accurate evaluator improves re-solving quality.

Pipeline Diagram



Pseudocode: Safe Subgame Re-solving

function SafeResolve(S, blueprint, budget)
    $\rho_i, \rho_{-i} \leftarrow$ ComputeRanges(S, blueprint)
    $V_{-i}^{\text{blue}}(S) \leftarrow$ OppValueFromBlueprint(S, blueprint)
    $V_{-i}^{\text{bound}}(S) \leftarrow V_{-i}^{\text{blue}}(S) - \delta$
    $G_S \leftarrow$ BuildSubgame(S, finer_abstraction)
    AddGadgetOpponentExit(G_S, $V_{-i}^{\text{bound}}(S)$)
    SetTerminalEvaluator(G_S, evaluator = blueprint/rollout/value)
    $\sigma^\star \leftarrow$ CFR_with_constraints(G_S, budget)
    $a \leftarrow$ SampleAction($\sigma^\star(S)$)
    return LiftToRealAction(a, legal_actions(S))
end function

Case Study: Libratus (Heads-Up No-Limit)

Decomposition:

- *Blueprint* for early streets: large abstract game solved offline.
- *Endgame solver* for later streets: fine abstraction, safe re-solving.

Safety:

- Opponent value bounds enforced at subgame root.
- Gadget games ensure no increase in exploitability.

Result: Beat top human pros over long matches.

Case Study: Pluribus (6-Player No-Limit)

Challenges: Multiplayer, huge state space, real-time play.
Approach:

- *Blueprint:* Linear CFR (LCFR) with external sampling.
- *Action abstraction:* limited discrete sizes.
- *Real-time search:* depth-limited lookahead with fast evaluator.

Note: Multiplayer safe re-solving guarantees are weaker, but empirically effective.

Practical Considerations

Action abstraction:

- Include min-raise and all-in when legal.
- Use proportional sizes (fractions of pot) plus context sizes (block bet, overbet).

Re-solving:

- Budget time per decision (e.g., 100ms to 1s).
- Prefer shallow depth + strong evaluator over deep + weak evaluator.
- Cache subgames/evaluations across similar public states if possible.

Common Pitfalls

Action abstraction:

- Too few sizes: predictable and misses optimal thresholds.
- Inconsistent lifting: illegal rounding that changes incentives.

Re-solving:

- Missing or loose safety constraints increases exploitability.
- Poor terminal evaluator yields myopic choices.
- Overfitting to specific boards; lack of robustness tests.

Summary

Action abstraction: Reduce $|A(I)|$ with fixed or dynamic bet menus; map back carefully.
Blueprint vs. re-solving: Offline coarse solve + online fine subgames.
Safe re-solving: Opponent value bounds, gadget exits, subgame margin.
Depth-limited: Use strong terminal evaluators (blueprint, rollout, value).
Practice: Libratus, Pluribus demonstrate scalable pipelines.

References

- Tammelin (2014): Solving Large Imperfect Information Games Using CFR+.
- Brown & Sandholm (2017): Safe and Nested Subgame Solving for Imperfect-Information Games (NIPS).
- Brown & Sandholm (2018): Superhuman AI for Heads-Up No-Limit Poker (Science).
- Brown & Sandholm (2019): Superhuman AI for Multiplayer Poker (Science).
- Moravčík et al. (2017): DeepStack: Expert-level AI in Heads-Up No-Limit Poker (Science).