CERTIFIED TEMPORAL EQUILIBRIUM SYNTHESIS WITH REUSABLE DEVIATION EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Temporal equilibrium synthesis seeks joint policies that satisfy a system requirement over infinite executions and admit no profitable unilateral deviation. During policy search, repeated certification can rediscover the same profitable behaviors after controller updates. We propose Certified Temporal Equilibrium Synthesis (CTES) for known finite deterministic games with Boolean linear temporal logic goals. CTES combines certification of constructed candidates with preference-guided finite-memory policy search and retains the best certified policy found. Its cross-profile deviation bank stores profitable infinite behaviors as finite prefixes and repeating loops. Replay checks whether the current opponents reproduce a stored behavior, including their memory updates through loop closure, and whether it strictly improves the deviator's current Boolean payoff. A successful replay lets the same witness reject another profile; otherwise, a complete verifier searches for new deviations. We prove sound rejection and sufficient transfer conditions against arbitrary legal pure history-dependent deviations. On contention games with up to twelve players, CTES improves the lexicographic policy score over certified initialization in 22 of 120 runs. Across matched resource-allocation streams, deviation reuse reduces total exact certification time by 15.95% with complete coverage. Under fixed search budgets, reuse improves certification throughput in contention and resource allocation.

1 INTRODUCTION

Multi-agent temporal equilibrium synthesis constructs joint policies that sustain temporal objectives while remaining stable against unilateral policy changes. Linear temporal logic (LTL) expresses eventual completion and recurring tasks (Pnueli, 1977), while Nash equilibrium (NE) formalizes the agents' incentives to follow a policy profile (Nash, 1950). Synthesis seeks equilibria that satisfy a system requirement and ranks them by operational quality.

Two robots share a passage between their homes and a meeting area. Each must visit the meeting area and return home infinitely often without collisions. Completing one round and parking remains safe but violates these goals. Either robot can improve its temporal payoff by continuing its trips with the opponent fixed. Repeated alternating schedules sustain both goals, with different crossing rates.

This example separates system feasibility on the nominal execution, strategic stability against deviations that turn a losing payoff into a winning one, and operational quality for ranking feasible equilibria. Controller memory also matters: repeated robot positions need not close a loop if memory prescribes a different next turn. Temporal evaluation and deviation checks must track that memory.

Rational synthesis and verification provide foundations for temporal reasoning under strategic behavior (Fisman et al., 2010; Wooldridge et al., 2016). Bounded synthesis and specification-guided learning support policy construction and equilibrium search (Finkbeiner & Schewe, 2013; Jothimurugan et al., 2019; 2022). Yet certifying successive candidates can repeatedly search for the same profitable infinite deviations. A deviation may survive changes elsewhere in a policy, while other updates invalidate its execution or remove its payoff advantage. Reusing it therefore requires checking both realizability and profitability under the current profile.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We propose *Certified Temporal Equilibrium Synthesis* (CTES) for known finite deterministic games. CTES uses certification outcomes throughout policy search: an external preference order selects the best certified profile found, called the incumbent, while profitable deviations enter a bank for rejecting later candidates. These deviations can also constrain symbolic proposals. Nominal scores guide generation even when certification rejects a candidate. Symbolic queries and parameter search use different proposal mechanisms but share the same payoffs and exact equilibrium certification.

The central insight is that one profitable infinite behavior can reject a family of joint policies. The *cross-profile deviation bank* stores that behavior as a prefix and repeating loop together with opponent memories. Replay checks whether the current opponents reproduce the behavior through loop closure and whether it improves on the deviator's current nominal payoff. Changing a robot's rule at an unvisited configuration can preserve both conditions. Changing its turn-taking memory or enabling nominal recurring trips can invalidate them. A successful replay certifies rejection of the current profile; otherwise, CTES invokes fresh verification.

Our contributions are as follows.

- We develop CTES to reuse certification outcomes during temporal policy search, combining certified initialization, preference-guided proposals, and deviation evidence under a shared computational budget.
- We establish checkable sufficient conditions for transferring profitable infinite behaviors across policy updates. Validating current opponent behavior through memory closure and checking current payoff improvement yield legal deviations that soundly reject the current profile.
- We instantiate CTES with symbolic queries and nominal-feedback search. Contention games with up to twelve players measure improvements beyond certified construction. Matched resource streams isolate the cost of reuse, showing a 15.95% reduction in total exact certification time, while fixed-budget searches measure its effect on certification throughput and returned policies.

2 Related Work

2.1 Temporal Games and Equilibrium Synthesis

Rational synthesis constructs system behavior under strategic assumptions on participating agents (Fisman et al., 2010), while rational verification checks temporal properties of strategically stable outcomes (Wooldridge et al., 2016; Gutierrez et al., 2018). Equilibrium existence has been studied for infinite multiplayer games (Ummels, 2006) and concurrent games with ω-regular objectives (Bouyer et al., 2015). Automated analysis combines automata and game solving (Gutierrez et al., 2020). CTES searches a finite controller domain, checks each player's deviations against its fixed opponents, and reuses profitable deviations across candidates.

Other formulations incorporate quantitative or lexicographic preferences into strategic objectives, with the latter comparing goals by priority (Gutierrez et al., 2017). Controllable-Nash synthesis constrains induced equilibria (Almagor et al., 2018), while equilibrium design changes the game (Gutierrez et al., 2024). Temporal coordination and assume-admissible synthesis offer further approaches to strategic outcomes (Fu et al., 2015; Brenguier et al., 2017). In CTES, Boolean temporal payoffs define individual incentives, and an external preference order guides search and selects among certified profiles.

2.2 Specification-Guided Policy Construction

Markov games model interacting decision makers (Littman, 1994) and support cooperative and competitive policy learning (Lowe et al., 2017). Temporal structure can guide learning through reward machines (Icarte et al., 2018), compositional specifications in SPECTRL (Jothimurugan et al., 2019), learned skill composition in Skill Machines (Nangue Tasse et al., 2024), and temporal automata in DeepLTL (Jackermeier & Abate, 2025). HighNashSearch combines welfare-guided enumeration with learning-based equilibrium checks (Jothimurugan et al., 2022). Using simulator samples in finite-horizon stochastic games, it obtains an ϵ-NE with high probability, bounding unilateral gains

by ϵ. CTES certifies exact pure NE for infinite-trace Boolean LTL goals in known deterministic games.

Automata-theoretic verification reduces infinite-trace specifications to finite acceptance problems (Vardi & Wolper, 1986; Baier & Katoen, 2008), and bounded synthesis constructs implementations through finite constraints (Finkbeiner & Schewe, 2013). Objective-formalism choices can change the policy problem (Subramani et al., 2024). CTES preserves the original LTL payoffs in both its symbolic and parameter-search branches, each followed by equilibrium certification.

2.3 Incremental Verification and Witness Reuse

Verification reuse takes several forms: conditional model checking records unresolved obligations (Beyer & Wendler, 2013), precision reuse transfers abstraction information (Beyer et al., 2013), and witness validation checks concrete counterexample evidence (Beyer et al., 2015). Verification-aided debugging stores witnesses and revalidates them after program changes (Beyer & Dangl, 2016). CTES transfers unilateral infinite behaviors across joint policy updates, checking both current opponent memory through loop closure and strict improvement over the current nominal payoff. Counterexample-guided inductive synthesis (CEGIS) constrains subsequent proposals using verification evidence (Solar-Lezama, 2013). CTES uses this evidence at two stages: symbolic repair excludes unstable controller families during generation, while bank replay checks stored deviations after a candidate is decoded.

3 Preliminaries and Problem Formulation

We model a finite deterministic multi-agent system with individual temporal goals and an external order over policy quality as

$$\mathcal{G} = \big(I, \mathcal{X}, \{A_i^0, A_i(\cdot)\}_{i\in I}, T_0, T, \varphi_{\text{sys}}, \{\varphi_i\}_{i\in I}, \Gamma, \preceq\big). \tag{1}$$

There are N players indexed by $I = \{1, \dots, N\}$. The finite state set $\mathcal{X} \subseteq 2^{AP}$ consists of valuations of the finite atomic-proposition set AP. Each state records which propositions hold. Player i has a finite nonempty initial action set A_i^0 and finite nonempty legal action sets $A_i(s)$. Write $a^0 = (a_i^0)_{i\in I}$ and $a = (a_i)_{i\in I}$ for joint actions. The maps $T_0(a^0)$ and $T(s, a)$ specify unique initial and successor states for every legal joint choice, so every legal execution is infinite. All players observe the state history. The transition maps include environment dynamics; in the passage example, guarded commands encode robot moves and make collisions absorbing. Appendix A gives the full semantics, with guarded-command execution in Appendix A.1.

We generate deterministic finite-memory controllers

$$\pi_i = (M_i, m_i^0, a_i^0, \alpha_i, \mu_i). \tag{2}$$

Here M_i is a finite nonempty memory set, with initial action $a_i^0 \in A_i^0$ and memory $m_i^0 \in M_i$. At time t, the controller outputs $a_{i,t} = \alpha_i(m_{i,t}, s_t) \in A_i(s_t)$ and updates its memory to $m_{i,t+1} = \mu_i(m_{i,t}, s_t, s_{t+1}) \in M_i$. Both maps are total, and outputs remain legal on deviating histories. A single memory state gives a memoryless policy; additional memory can record whose turn follows a completed trip (Appendix A.2). A joint profile $\pi = (\pi_i)_{i\in I}$ induces the unique state sequence $\text{out}(\pi) = s_0 s_1 \cdots$ from $s_0 = T_0(a^0)$. The finite state–memory system eventually repeats, yielding a *lasso*: a finite prefix followed by a nonempty loop repeated forever.

The system requirement φ_{sys} and player goals φ_i are Boolean LTL formulas on infinite state words. Appendix A.3 defines their syntax and truth conditions. Appendix A.5 gives the passage source code and follows candidate generation, temporal evaluation, and equilibrium selection in Example 1.

Goal satisfaction defines the Boolean payoff

$$u_i(\pi) = \mathbf{1}[\text{out}(\pi) \models \varphi_i] \in \{0, 1\}. \tag{3}$$

Here $\rho \models \varphi$ denotes satisfaction of φ by the state sequence ρ, and $\mathbf{1}[\cdot]$ is its Boolean indicator. A pure deviation σ_i chooses a legal initial action and then selects legal actions deterministically from the state history. Let Σ_i contain all such deviations, unrestricted by the candidate's memory or parameterization. A Nash equilibrium admits no strictly profitable unilateral deviation from the

initial game. In (σ_i, π_{-i}), only player i changes strategy; the opponents $\pi_{-i} = (\pi_j)_{j \neq i}$ remain fixed. Only a nominal loser can strictly improve a Boolean payoff. An equilibrium may therefore contain unsatisfied players if their fixed opponents prevent every goal-satisfying deviation.

The ordered collection Γ contains quality functions evaluated on the canonical nominal lasso, obtained at the first repeated state–memory configuration. Their values form the score vector $q_\Gamma(\pi)$. A supplied total preorder $\preceq$ compares every pair of score vectors and is reflexive and transitive; larger values are preferred. Its strict part satisfies $z \succ z'$ exactly when $z' \preceq z$ and $z \not\preceq z'$ for score vectors z, z'. Quality functions can measure crossing rate or the fraction of satisfied players (Appendix A.4). Preferences guide search, while individual payoffs determine equilibrium. The system requirement constrains the delivered nominal execution; legal deviations may violate φ_{sys}. Setting $\varphi_{\text{sys}} = \texttt{true}$ removes this additional requirement.

Let Θ be a finite parameter domain. Each $\theta \in \Theta$ decodes to a legal profile π_θ. Then $\Pi_\Theta = \{\pi_\theta : \theta \in \Theta\}$. Feasible profiles satisfy the system requirement and the Nash condition together,

$$\mathcal{F}_\Theta(\mathcal{G}) = \left\{ \pi \in \Pi_\Theta \;\middle|\; \begin{array}{l} \text{out}(\pi) \models \varphi_{\text{sys}}, \\ u_i(\sigma_i, \pi_{-i}) \leq u_i(\pi) \quad \forall i \in I,\ \forall \sigma_i \in \Sigma_i \end{array} \right\}. \tag{4}$$

Given $(\mathcal{G}, \Theta)$ and a computational budget, synthesis seeks the best feasible profile it can certify. It returns this incumbent $\pi^\star$ with its nominal lasso and objective values, or reports that none was certified. Global optimality requires completing the search or soundly excluding every better candidate.

4 Certified Temporal Equilibrium Synthesis

4.1 Framework Overview

CTES evaluates and certifies proposed profiles π before selection (Figure 1). Constructors provide initial candidates, followed by parameter search or symbolic queries. Nominal evaluation computes system satisfaction, player payoffs, and $q_\Gamma(\pi)$ on the canonical lasso, guiding search and identifying profiles for strategic checking.

For a system-valid candidate, the bank $\mathcal{B}$ checks stored deviations against current opponent actions, memory closure, and payoff improvement. A profitable replay rejects π. Otherwise, fresh certification searches for a profitable deviation by any nominal loser. A discovered deviation rejects the profile and enters the bank after validation. Proving absence for every loser certifies the profile for selection.

Selection retains the best certified profile; nominal scores guide parameter search even after rejection. Validated deviations constrain symbolic queries and support replay. All stages share one budget. Any incumbent is returned with its lasso, payoffs, and score.

4.2 Preference-Guided Candidate Generation

Both search branches use scores computed from the nominal execution of π_θ for $\theta \in \Theta$. Write $m = (m_j)_{j \in I}$ for joint memory and $x = (s, m)$ for the full configuration. Initially $x_0 = (T_0(a^0), m^0)$ with $m^0 = (m_j^0)_{j \in I}$. The joint output $a = (\alpha_j(m_j, s))_{j \in I}$ yields $s' = T(s, a)$ and the transition

$$F_\theta(s, m) = \big(s', (\mu_j(m_j, s, s'))_{j \in I}\big). \tag{5}$$

The first repeated configuration defines the canonical lasso λ_π; write $\lambda_\theta = \lambda_{\pi_\theta}$. Evaluating this lasso supplies the payoffs for certification and the score for search and selection. Including memory distinguishes periodic turn-taking from one-round parking.

Initialization evaluates and attempts to certify an ordered sequence $\mathcal{H}_{\text{init}}$ of constructed profiles. The best certified feasible profile initializes the incumbent and seeds the parameter queue, with ties retaining the first profile. If none is certified, the queue uses a default root. Coupled contention uses weighted and count-oriented greedy constructions.

The queue expands from this root through local parameter changes. Let Θ_{seen} contain previously queued parameters and d index coordinates. The unvisited neighborhood is

$$\mathcal{N}(\theta) = \{\theta' \in \Theta \setminus \Theta_{\text{seen}} : |\{d : \theta'_d \neq \theta_d\}| = 1\}. \tag{6}$$

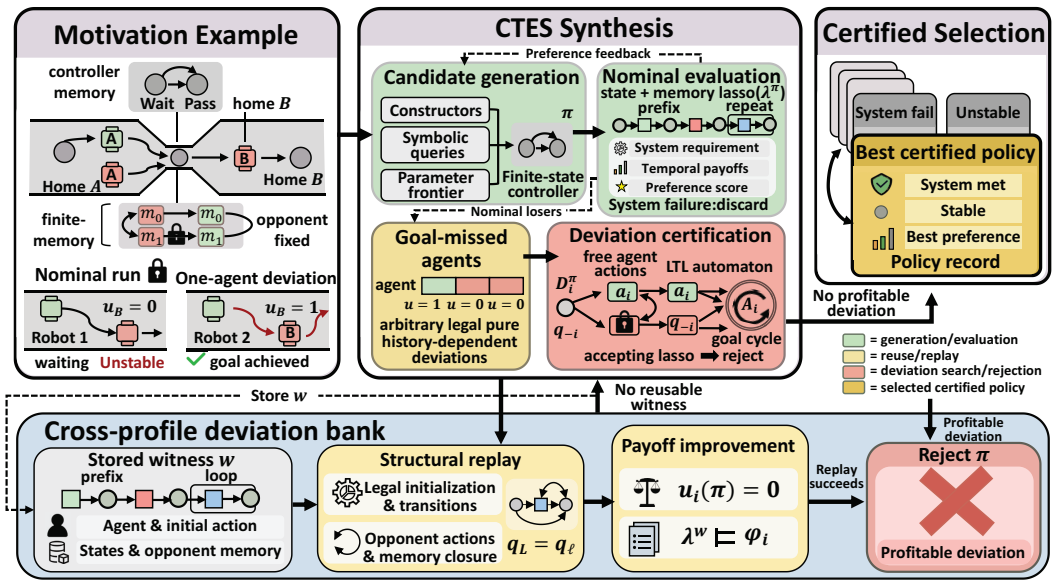

Figure 1: CTES passes candidate profiles through evaluation and certification. Scores guide further proposals; validated deviations feed the bank and symbolic repair.

Each queued neighbor inherits priority $\mathrm{prio}(\theta') = q_\Gamma(\pi_{\mathrm{par}(\theta')})$ from its first proposing parent $\mathrm{par}(\theta')$. The queue selects a greatest priority under $\preceq$ with seeded tie-breaking. After evaluation, the candidate's own nominal score determines the priorities of newly queued neighbors, even after rejection.

Symbolic generation instead constrains a controller and a bounded nominal lasso by legality, the system requirement, and a preference query (Appendix A.6). The decoded profile enters the same nominal evaluator and strategic checker as a parameter-search candidate.

4.3 Temporal and Equilibrium Certification

Nominal evaluation first discards profiles that violate the system requirement. For each remaining profile, its payoffs identify the losers $I_0(\pi) = \{j \in I : u_j(\pi) = 0\}$. If bank replay does not establish rejection, fresh certification tests whether any $i \in I_0(\pi)$ can improve. Fixing its opponents gives an action graph D_i^π with vertex set $\mathcal{V}_i = \mathcal{X} \times \prod_{j \neq i} M_j$. At (s, q), the tuple $q = (q_j)_{j \neq i}$ records opponent memories. The deviator chooses any $a_i \in A_i(s)$ while the opponents produce $\alpha_{-i}(q, s) = (\alpha_j(q_j, s))_{j \neq i}$, yielding

$$s' = T\big(s, (a_i, \alpha_{-i}(q, s))\big). \tag{7}$$

The updates $q_j' = \mu_j(q_j, s, s')$ complete the edge $(s, q) \xrightarrow{a_i} (s', q')$. Initial configurations range over all legal a_i^0 with a_{-i}^0 and m_{-i}^0 fixed. The graph thus varies the deviator's choices while applying the opponents' unchanged controllers along each resulting history. Appendix B specifies admission and certification procedures.

For a nominal loser, a profitable deviation exists exactly when this graph has a path satisfying φ_i; denote this condition by $\mathrm{Dev}_i(\pi)$. For general LTL, we synchronize the graph with a Büchi automaton for φ_i, which accepts runs visiting its accepting states infinitely often. A reachable accepting product cycle supplies a deviation lasso. Conversely, every satisfying history-dependent deviation yields an accepting path. This search covers all legal pure deviations without imposing the proposal's memory bound (Appendix C.1).

Combining the deviation decisions with the nominal system check gives the feasibility indicator

$$\mathsf{Cert}(\pi) = \mathbf{1}\left[\lambda_\pi \models \varphi_{\mathrm{sys}} \ \wedge \bigwedge_{i \in I_0(\pi)} \neg \mathsf{Dev}_i(\pi)\right]. \tag{8}$$

Boolean winners cannot improve, so absence proofs for every loser suffice to certify the profile. A profitable lasso instead supplies rejection evidence for the bank. Resource exhaustion leaves certification unresolved. Appendix B.3 describes the evaluated search procedures.

4.4 Cross-Profile Deviation Bank

A profitable deviation lasso may survive a policy update. The bank stores it for checking later candidates before fresh search. Each witness records the deviator, initial joint action, and state–memory lasso

$$w = \big(i, a^0, (s_k, q_k, a_k)_{k=0}^{L-1}, \ell\big). \tag{9}$$

Here $L \geq 1$ counts stored positions and $0 \leq \ell < L$ is the loop entry. The tuple q_k records opponent memories and a_k the next joint action. The word $\lambda_w = s_0 \cdots s_{\ell-1}(s_\ell \cdots s_{L-1})^\omega$ uses ω for infinite repetition. Its prefix is empty when $\ell = 0$. Define $\nu(k) = k + 1$ for $k < L - 1$ and $\nu(L-1) = \ell$.

The recorded memories follow the deviation, so an unchanged nominal outcome need not preserve the witness. Structural replay checks whether current opponents reproduce every recorded transition through loop closure.

Definition 1 (Structural replay)**.** A well-formed witness w structurally replays against π if its recorded actions are legal and its opponent memories belong to the current memory sets. Initialization requires $T_0(a^0) = s_0$. Each opponent $j \neq i$ retains $a_j^0(w) = a_j^0(\pi)$ and starts at $q_{j,0} = m_j^0$. For every $0 \leq k < L$, the recorded edge obeys $T(s_k, a_k) = s_{\nu(k)}$ and each opponent supplies $a_{j,k} = \alpha_j(q_{j,k}, s_k)$. Its memory update satisfies

$$q_{j,\nu(k)} = \mu_j(q_{j,k}, s_k, s_{\nu(k)}). \tag{10}$$

These conditions include the closing edge with $\nu(L-1) = \ell$.

Restoring q_ℓ at loop closure ensures that every traversal reproduces the recorded opponent actions and state word.

Structural replay establishes realizability. Rejection additionally requires $u_i(\pi) = 0$ and $\lambda_w \models \varphi_i$, so the deviation is profitable for the current profile. The stored word's satisfaction is fixed within a game, but the nominal payoff must be recomputed because an updated profile may already satisfy the goal.

Theorem 1 (Sound negative reuse)**.** Under the game conditions of Section 3, a structurally replayable witness with $u_i(\pi) = 0$ and $\lambda_w \models \varphi_i$ proves that π is not a Nash equilibrium.

The deviator realizes the stored word by following its actions at successive history lengths. Profitability makes this a strict Boolean improvement. Appendix C proves realizability and rejection and gives sufficient transfer conditions.

A validated witness can therefore reject multiple profiles after off-trace changes. CTES tries witnesses in reverse insertion order. If none establishes rejection, fresh certification searches for a deviation and adds validated evidence to the bank. Appendices B.2 and B.4 describe the procedure and its overhead.

4.5 Evidence Feedback and Certified Selection

The rejection criterion also guides symbolic generation. Bank replay tests a decoded candidate, whereas symbolic repair excludes candidates satisfying the rejection criterion from a query. Let $\mathsf{Replay}_i(w; \pi)$ mean that Definition 1 holds. The corresponding repair constraint is

$$\Phi_{\mathrm{block}}(w) = \neg\big[u_i(\pi) = 0 \ \wedge \ \mathsf{Replay}_i(w; \pi) \ \wedge \ \lambda_w \models \varphi_i\big]. \tag{11}$$

Theorem 1 justifies both direct encoding and conservative exclusion of a smaller unstable family (Appendix A.6). Parameter search uses the nominal score to prioritize neighbors and applies replay

after decoding. Evidence therefore acts during generation for symbolic queries and during certification for both branches.

Selection admits only certified feasible candidates. The first initializes the incumbent; later candidates replace it only when their scores strictly improve, with ties retaining the earlier policy. The incumbent is therefore the best certified feasible profile found so far (Appendix C.5). The symbol $\perp$ denotes absence of a candidate or witness.

Algorithm 1 combines generation, certification, feedback, and selection at search iteration τ. Appendix B.1 gives the complete branching and initialization procedure.

Algorithm 1 CTES with reusable deviation evidence

Require: Game $\mathcal{G}$, space Θ, generator Gen, initial sequence $\mathcal{H}_{\text{init}}$, budget b
1: Initialize Gen with $\mathcal{H}_{\text{init}}$ first; set $\mathcal{B} \leftarrow \emptyset$, $\pi^\star \leftarrow \perp$.
2: **while** budget remains and Gen has a proposal **do**
3: Propose π_τ; evaluate legality, nominal lasso, payoffs, and preference score.
4: For a legal, system-valid candidate, try profitable bank replay.
5: On a miss, check deviations for every nominal loser within the budget.
6: Validate and retain new witnesses; add symbolic blocks when enabled.
7: A certified feasible candidate initializes $\pi^\star$ or replaces it on strict improvement.
8: Pass nominal feedback to Gen; after $\mathcal{H}_{\text{init}}$, seed the frontier from $\pi^\star$ or its root.
9: **end while**
10: **return** The incumbent and its evaluated record, or no certified policy within the budget.

5 Experiments

We evaluate the quality of synthesized equilibria, the validity of certification decisions, and the computational benefit of deviation reuse.

5.1 Experimental setup

Coupled contention tests n players seeking collision-free recurring service on an interference graph. Resource allocation has n clients and n strategic monitors, while reward trap separates external reward from individual safety goals. Welfare is the fraction of satisfied players, $W = N^{-1} \sum_i u_i(\pi)$. Appendix D specifies the protocols and controller domains; Appendix D.3 defines the resource game.

Contention uses ten new games at each of $n = 3, 5, 6, 8, 10, 12$, with two seeds per game. The 480 runs compare CTES against three controls: guidance disabled, bank disabled, and construction alone. All configurations certify the same two initial candidates within the 600-second run budget. Statistics first average the two seeds within each game.

Resource reuse controls pair identical 32-candidate streams at three, four, and five clients. Each size has five seeds and two timing repetitions. Separate resource searches use 60 seconds per run.

5.2 Quality of synthesized equilibria

In contention, each player has a two-state controller with four request schedules and an optional persistent response. Let f_i^{svc} and f_i^{req} be the fractions of nominal-loop states recording service and a request for player i. With player weights $\beta_i > 0$, weighted service is $S = \sum_i \beta_i f_i^{\text{svc}} / \sum_i \beta_i$ and request rate is $R = n^{-1} \sum_i f_i^{\text{req}}$. Preferences maximize $(W, S, -R)$ lexicographically. Service contributes to S even when a player fails its safety or eligible-phase obligation. The two constructors prioritize weight or player count relative to degree. Within each of the 120 paired blocks, all configurations start from the same certified score.

Construction takes 0.071–1.198 seconds on average across sizes. Relative to construction, CTES improves 22 of 120 paired score vectors and ties the rest (Table 1); eleven improvements increase W. At five and six players, mean W rises from 0.660 and 0.667 to 0.700. All configurations tie at ten

Table 1: Paired certified quality within 600 seconds. Entries count CTES wins/ties/losses against each control under the full lexicographic order $(W, S, -R)$ over twenty runs per size. All methods return a certified policy in every run. Appendix E gives full scores and game-level intervals.

Players	Construction	Without guidance	Without bank
3	2/18/0	0/20/0	0/20/0
5	8/12/0	1/19/0	0/20/0
6	5/15/0	3/16/1	0/20/0
8	3/17/0	2/17/1	1/19/0
10	0/20/0	0/20/0	0/20/0
12	4/16/0	4/16/0	0/20/0
All	22/98/0	10/108/2	1/119/0

players. At twelve, CTES improves S by 1.94% at fixed W, while R increases. The lexicographic order accepts this tradeoff because service has higher priority than request rate.

Against unguided search, CTES yields ten better vectors, 108 ties, and two worse vectors; game-level 95% intervals for mean W differences include zero. Relative to the no-bank control, the bank increases completed definitive candidate checks by 7.51%–28.84% at sizes five through twelve. These checks include both acceptance and rejection; the final certified score vectors tie in 119 of 120 comparisons. At twelve players, CTES averages 681.85 definitive checks within 600 seconds (Appendix E.3). Appendix E gives paired results and time curves, and Appendix E.4 reports the separate cold-start controls.

Appendix E.2 tracks certified welfare and weighted service through time, including initialization.

Figure 2 pairs the contention check-count gains with total certification times on matched resource streams. Three-player contention exhausts the domain with either configuration, yielding equal check counts.

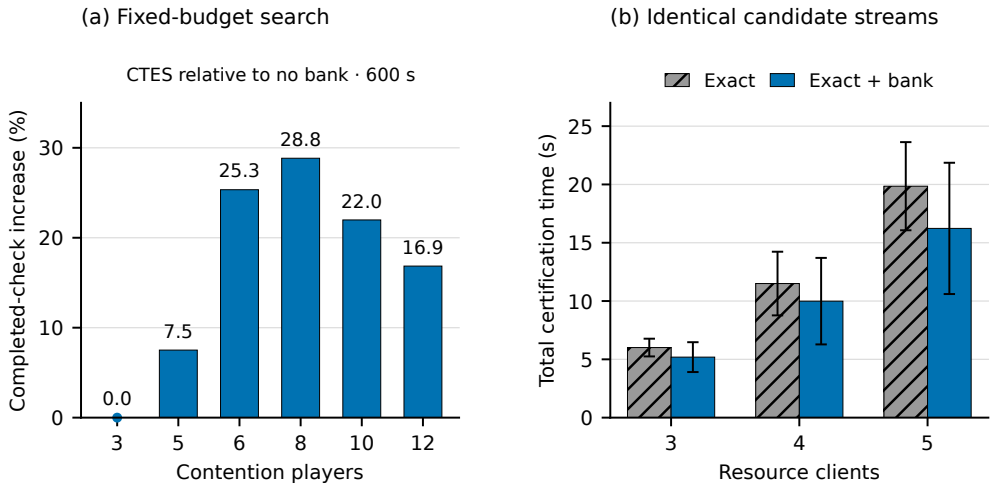

Figure 2: Computational benefit of deviation reuse. (a) Relative increase in mean completed candidate checks over the no-bank control within 600 seconds; checks include acceptance and rejection. (b) Total time for identical 32-candidate resource streams, including all bank costs; bars show means and error bars show seed SDs across five seeds after averaging two repetitions.

Why strategic certification matters. Reward trap tests whether certification rejects reward-seeking proposals that conflict with individual goals. Agent i controls on_i and requires $\mathsf{G}\neg on_i$, whereas the external reward counts occurrences of on_0. Any positive reward violates agent 0's goal and admits a profitable unilateral deviation: keeping on_0 false from initialization onward. Every

equilibrium therefore has zero reward. All 40 runs retain equilibria satisfying every goal and reject unstable proposals, enforcing individual incentives during reward optimization (Appendix F).

Controlled cases test the bank's replay conditions. Changing an opponent's remembered response can invalidate a stored deviation even when nominal play is unchanged. A previously profitable witness also offers no strict improvement once its deviator becomes a nominal winner. Both changes correctly prevent reuse. All nine replay cases and six strong-until cases agree with their reference checks (Appendices F.1 and F.2).

5.3 Certification efficiency

Reuse and its cost. The fixed 32-candidate streams isolate the work saved and added by reuse. Both Exact configurations complete all 960 matched candidate evaluations. Appendix G.1 reports total runtime and replay activity. Stored deviations reject an average of 10.0–12.4 candidates per stream, each avoiding fresh deviation search for that candidate. Replay costs 0.78–1.13 seconds per stream, including unsuccessful attempts.

Total cost falls by 15.95% across matched evaluations, measured as a ratio of summed times. This total includes unsuccessful replay and insertion, so the avoided verification cost outweighs the bank's overhead. Internal timers overlap and are reported separately. Appendix G gives individual seed effects; Appendix G.2 traces a witness that transfers across changes to off-trace opponent behavior.

5.4 Effect on fixed-budget synthesis

Within 60 seconds, reuse increases equilibrium-certification totals over ten runs from 743 to 782, 352 to 367, and 258 to 291 at three, four, and five clients, respectively (Figure 3): gains of 5.25%, 4.26%, and 12.79%. Counts increase in 13 of 15 size–seed means and tie in two; they count certification events across runs.

Both Exact variants always return an incumbent. Full welfare holds in all ten runs at three and four clients; at five, attainment rises from six to eight runs. Appendix H gives welfare curves and paired counts; Appendix H.2 reports capacity/order controls.

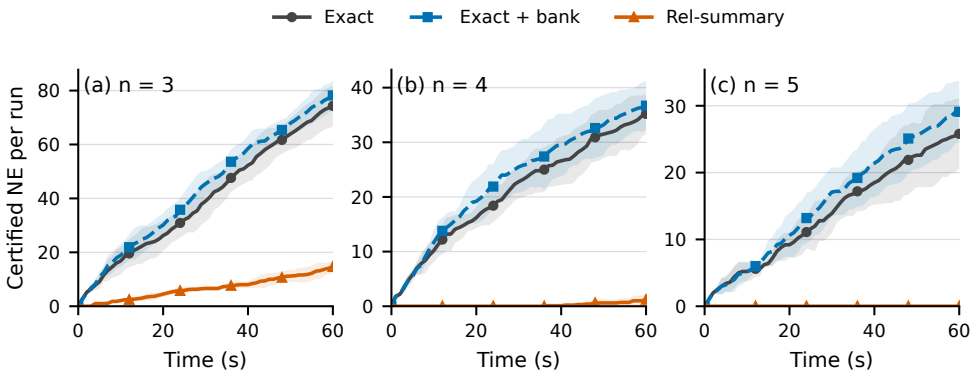

Figure 3: Resource search over 60 seconds at three, four, and five clients. Curves show mean equilibrium-certification counts per run; bands show one seed SD across five seeds after averaging two repetitions. Appendix H reports endpoint counts and welfare.

6 Conclusion

CTES reuses profitable deviations in certified temporal policy search. Checking current opponent behavior through memory closure and current payoff improvement lets one witness soundly reject multiple profiles. Contention search improves certified initialization in 22 of 120 runs. Matched resource streams show net certification savings, and fixed-budget resource search yields more equilibrium certifications and higher full-welfare attainment at five clients. Future work includes deviation-guided learned proposals and transfer conditions for stochastic or partially observable games.

AI USE STATEMENT

Generative AI tools were used solely for language editing of the manuscript, including grammar correction, wording refinement, and improvements to clarity and readability. Generative AI tools were not used for research ideation, literature search, methodology or algorithm development, mathematical derivations or proofs, experimental design or implementation, data analysis, result interpretation, or figure generation.

ETHICS STATEMENT

The evaluation uses generated finite games and contains no human-subject or personal data. The equilibrium guarantees apply to the stated game dynamics and objectives. Application-specific safety and fairness requirements must be expressed in the model and checked separately.

REPRODUCIBILITY STATEMENT

Appendix A specifies the game and controller semantics, and Appendix C gives the proofs. Appendix B describes the verification procedures. Appendix D details game generation, candidate domains, budgets, aggregation, and execution settings. The result appendices report certification coverage and paired comparisons, including seed-level throughput differences in Table 19.

REFERENCES

Shaull Almagor, Orna Kupferman, and Giuseppe Perelli. Synthesis of controllable Nash equilibria in quantitative objective games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 35–41, 2018. doi: 10.24963/ijcai.2018/5.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Dirk Beyer and Matthias Dangl. Verification-aided debugging: An interactive web-service for exploring error witnesses. In *Computer Aided Verification (CAV), Part II*, volume 9780 of *Lecture Notes in Computer Science*, pp. 502–509. Springer, 2016. doi: 10.1007/978-3-319-41540-6_28. URL https://link.springer.com/chapter/10.1007/978-3-319-41540-6_28.

Dirk Beyer and Philipp Wendler. Reuse of verification results: Conditional model checking, precision reuse, and verification witnesses. In *Model Checking Software (SPIN)*, volume 7976 of *Lecture Notes in Computer Science*, pp. 1–17. Springer, 2013. doi: 10.1007/978-3-642-39176-7_1. URL https://link.springer.com/chapter/10.1007/978-3-642-39176-7_1.

Dirk Beyer, Stefan Löwe, Evgeny Novikov, Andreas Stahlbauer, and Philipp Wendler. Precision reuse for efficient regression verification. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pp. 389–399. ACM, 2013. doi: 10.1145/2491411.2491429. URL https://doi.org/10.1145/2491411.2491429.

Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, and Andreas Stahlbauer. Witness validation and stepwise testification across software verifiers. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pp. 721–733. ACM, 2015. doi: 10.1145/2786805.2786867. URL https://doi.org/10.1145/2786805.2786867.

Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent deterministic games. *Logical Methods in Computer Science*, 11(2):9:1–9:72, 2015. doi: 10.2168/LMCS-11(2:9)2015. URL https://doi.org/10.2168/LMCS-11(2:9)2015.

Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017. doi: 10.1007/s00236-016-0273-2.

Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5–6):519–539, 2013. doi: 10.1007/s10009-012-0228-z.

Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 190–204. Springer, 2010. doi: 10.1007/978-3-642-12002-2_16.

Jie Fu, Herbert G. Tanner, and Jeffrey Heinz. Concurrent multi-agent systems with temporal logic objectives: Game theoretic analysis and planning through negotiation. *IET Control Theory & Applications*, 9(3):465–474, 2015. doi: 10.1049/iet-cta.2014.0611.

Julian Gutierrez, Aniello Murano, Giuseppe Perelli, Sasha Rubin, and Michael Wooldridge. Nash equilibria in concurrent games with lexicographic preferences. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1067–1073, 2017. doi: 10.24963/ijcai.2017/148.

Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge. EVE: A tool for temporal equilibrium analysis. In *Automated Technology for Verification and Analysis (ATVA)*, pp. 551–557. Springer, 2018. doi: 10.1007/978-3-030-01090-4_35.

Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artificial Intelligence*, 287:103353, 2020. doi: 10.1016/j.artint.2020.103353. URL https://doi.org/10.1016/j.artint.2020.103353.

Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge. Designing equilibria in concurrent games with social welfare and temporal logic constraints. *Logical Methods in Computer Science*, 20(4):21:1–21:30, 2024. doi: 10.46298/lmcs-20(4:21)2024.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2107–2116, 2018. URL https://proceedings.mlr.press/v80/icarte18a.html.

Mathias Jackermeier and Alessandro Abate. DeepLTL: Learning to efficiently satisfy complex LTL specifications for multi-task RL. In *International Conference on Learning Representations (ICLR)*, pp. 14000–14028, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/24c523085d10743633f9964e0623dbe0-Paper-Conference.pdf.

Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/f5aa4bd09c07d8b2f65bad6c7cd3358f-Abstract.html.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Specification-guided learning of Nash equilibria with high social welfare. In *Computer Aided Verification (CAV), Part II*, volume 13372 of *Lecture Notes in Computer Science*, pp. 343–363. Springer, 2022. doi: 10.1007/978-3-031-13188-2_17. URL https://link.springer.com/chapter/10.1007/978-3-031-13188-2_17.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157–163. Morgan Kaufmann Publishers Inc., 1994.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html.

Geraud Nangue Tasse, Devon Jarvis, Steven James, and Benjamin Rosman. Skill machines: Temporal logic skill composition in reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024. URL

https://proceedings.iclr.cc/paper_files/paper/2024/hash/9ee3a664ccfeabc0da16ac6f1f1cfe59-Abstract-Conference.html.

John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. doi: 10.1073/pnas.36.1.48.

Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pp. 46–57, 1977. doi: 10.1109/SFCS.1977.32.

Armando Solar-Lezama. Program sketching. *International Journal on Software Tools for Technology Transfer*, 15:475–495, 2013. doi: 10.1007/s10009-012-0249-7.

Rohan Subramani, Marcus Williams, Max Heitmann, Halfdan Holm, Charlie Griffin, and Joar Skalse. On the expressivity of objective-specification formalisms in reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/35514d533cdc278a7780daf0dbe7d0b7-Abstract-Conference.html.

Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pp. 212–223, 2006. doi: 10.1007/11944836_21.

Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Annual Symposium on Logic in Computer Science*, pp. 332–344, 1986.

Michael Wooldridge, Julian Gutierrez, Paul Harrenstein, Enrico Marchioni, Giuseppe Perelli, and Alexis Toumi. Rational verification: From model checking to equilibrium checking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. doi: 10.1609/aaai.v30i1.9878. URL https://ojs.aaai.org/index.php/AAAI/article/view/9878.

A Game Semantics and Policy Representation

The source game determines legal executions, controllers select nominal behavior, and temporal formulas assign payoffs to the resulting infinite words. External preferences rank the certified profiles. We make these semantics explicit below, illustrate them in the passage game, and express the proposal requirements as symbolic constraints.

A.1 Guarded-command game semantics

The finite proposition set AP is partitioned into disjoint sets controlled by the strategic modules and an optional environment module. A state is a valuation $s \subseteq AP$. A guarded command $c = (g, \eta)$ has a propositional guard g and a partial assignment η to its module's variables. All right-hand sides are propositional formulas. For an input valuation $v \subseteq AP$, the command is enabled when $v \models g$. Writing $\mathrm{dom}(\eta)$ for its assigned variables, it produces the action

$$\mathrm{act}(c, v) = \{p \in \mathrm{dom}(\eta) : v \models \eta(p)\}. \tag{12}$$

Every controlled variable outside this set is false in the action. Assignments within one command are evaluated simultaneously on v. A variable omitted by an enabled command is therefore set to false rather than retained.

Initial commands are evaluated on the empty valuation, whereas update commands of a strategic module are evaluated on the current state. The distinct outcomes of enabled commands form the legal action set from which the player chooses; several commands may yield the same action. If no update command is enabled, the module retains its current controlled valuation. Absence of an enabled initial command makes the model invalid.

The legal module actions combine into a joint transition. Let $v_P = \bigcup_{i \in I} a_i$ collect the chosen strategic actions. Without an environment module, the successor is v_P. Otherwise, the environment's update commands are evaluated on $s \cup v_P$ and must yield exactly one distinct action $e(s \cup v_P)$. The resulting transition is

$$T(s, a) = v_P \cup e(s \cup v_P). \tag{13}$$

Initialization uses the same construction with an empty old valuation and the environment's initial commands. The environment's update fallback retains its controlled valuation when no guard is enabled. Two enabled commands are permitted if they produce the same action. Distinct environment outcomes invalidate the deterministic model.

The environment input is a set union. An old true player variable therefore remains visible to the environment even when its new action makes that variable false. The final state still uses the new strategic action v_P. This distinction determines the one-step persistence of strategic signals in the evaluated source models. The abstract transition map T includes this convention.

These execution rules determine the admission obligations. Admission requires disjoint variable ownership and well-formed propositional guards and assignments. It checks nonempty initial action sets and deterministic successors for all legal joint choices on the game domain. The finite state set may be restricted to states reachable from legal initial choices under arbitrary legal joint actions. Restricting it to one proposed profile's nominal execution would omit deviation histories. The update fallback supplies an action at each reachable state. These rules yield total dynamics on the admitted game.

A.2 Controllers and unilateral strategies

Controllers resolve the legal choices defined above. A candidate controller has a finite nonempty memory set M_i. Its output map α_i is defined on $M_i \times \mathcal{X}$ and satisfies $\alpha_i(m, s) \in A_i(s)$ throughout that domain. Its update map μ_i is defined on $M_i \times \mathcal{X} \times \mathcal{X}$ and returns a member of M_i. Initial actions execute first. Initial memories are then installed at $s_0 = T_0(a^0)$. Each subsequent step reads (m_i, s) to choose an action and reads (m_i, s, s') to update memory. Ordered guarded memory rules implement a total update map by choosing the first matching rule and using a specified default when none matches.

Certification compares this finite-memory candidate with unrestricted legal deviations. Let Hist contain all nonempty finite legal state histories of the game. A pure unilateral strategy consists of an

initial choice $\sigma_i^0 \in A_i^0$ and a function on Hist. At a history $h = s_0 \cdots s_t$, it selects $\sigma_i(h) \in A_i(s_t)$. The strategy may distinguish histories of arbitrary length and use unrestricted memory. Opponents retain their initial choices and their complete controller functions. Their memories are recomputed along the deviating history rather than copied from the nominal execution.

The deterministic dynamics give a unique infinite outcome for each joint strategy. Nash equilibrium compares the deviating and nominal outcomes from their respective legal initial choices. A strict improvement means changing the player's original Boolean payoff from zero to one. The deviator is constrained by the game dynamics and its own goal. The system formula constrains the delivered nominal outcome and does not impose an additional restriction on deviations. Equilibrium from the initial game also places no optimality requirement on opponent behavior after unreachable histories.

A.3 Temporal formulas and lasso truth

The nominal and deviating executions above receive Boolean payoffs through temporal satisfaction. The language is generated from atomic propositions using the grammar

$$\varphi ::= \mathsf{true} \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \mathsf{X}\varphi \mid (\varphi \,\mathsf{U}\, \varphi), \qquad p \in AP.$$

Boolean disjunction and implication are derived connectives. Eventuality is $\mathsf{F}\varphi \equiv \mathsf{true}\,\mathsf{U}\,\varphi$. Always is $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$. The constant false and the equivalence connective have their usual Boolean definitions.

For an infinite word $\rho = s_0 s_1 \cdots$, write $\rho, t \models \varphi$ for truth at position t. The atomic clause is $\rho, t \models p$ exactly when $p \in s_t$. Negation and conjunction use Boolean truth at the same position. The constant true holds at every position. Next-step truth is

$$\rho, t \models \mathsf{X}\varphi \quad \Longleftrightarrow \quad \rho, t+1 \models \varphi. \tag{14}$$

Strong until requires a future position at which the right operand holds,

$$\rho, t \models \varphi \,\mathsf{U}\, \psi \quad \Longleftrightarrow \quad \exists k \geq t \, \big[\rho, k \models \psi \ \wedge\ \forall r \in [t, k)\ \rho, r \models \varphi\big]. \tag{15}$$

The right operand may hold immediately at $k = t$. The interval condition is then empty. Satisfaction of a complete execution means truth at position zero after initialization.

A deterministic candidate repeats a full configuration $x = (s, m)$. Let L be the first repeated-configuration time and let $\ell < L$ satisfy $x_L = x_\ell$. Its canonical state word is $\lambda = s_0 \cdots s_{\ell-1}(s_\ell \cdots s_{L-1})^\omega$. The distinct configurations before time L make ℓ unique. This length is at most $|\mathcal{X}| \prod_i |M_i|$. Repeating a physical state without its memory tuple does not establish the loop.

For propositional formulas ψ and χ, $\mathsf{F}\psi$ requires an eventual occurrence and $\mathsf{G}\psi$ requires truth at every step. Recurrence $\mathsf{GF}\psi$ requires infinitely many occurrences. Eventual permanence $\mathsf{FG}\psi$ requires truth from some step onward. Safe recurrence combines $\mathsf{G}\chi$ with $\mathsf{GF}\psi$. The framework also supports next-step X and strong-until U with Boolean combinations. Eventuality and safety inspect the whole lasso. Recurrence and eventual permanence depend on its loop.

Set $\mathcal{K} = \{0, \ldots, L-1\}$ and $\mathcal{K}_{\text{loop}} = \{\ell, \ldots, L-1\}$. Table 2 gives direct tests for propositional ψ and χ. General formulas use the infinite-word clauses above. Equivalently, lasso evaluation sends the successor of $L-1$ to ℓ. The until clause uses a least fixed point on these positions, so a cycle that never reaches its right operand cannot satisfy strong until.

Table 2: Truth conditions on a lasso. Prefix and loop positions are both in $\mathcal{K}$; $\mathcal{K}_{\text{loop}}$ contains only the loop.

Objective	Necessary and sufficient condition
$\mathsf{F}\psi$	Some $k \in \mathcal{K}$ satisfies ψ
$\mathsf{G}\psi$	Every $k \in \mathcal{K}$ satisfies ψ
$\mathsf{GF}\psi$	Some $k \in \mathcal{K}_{\text{loop}}$ satisfies ψ
$\mathsf{FG}\psi$	Every $k \in \mathcal{K}_{\text{loop}}$ satisfies ψ
$\mathsf{G}\chi \wedge \mathsf{GF}\psi$	Every $k \in \mathcal{K}$ satisfies χ and some $k \in \mathcal{K}_{\text{loop}}$ satisfies ψ

A.4 External preferences and output meaning

Temporal truth determines feasibility and player incentives, while operational preferences rank the certified profiles. Each primitive $\gamma \in \Gamma$ is a specified computable function of the canonical nominal lasso with values in a declared ordered domain. The vector $q_\Gamma(\pi)$ collects these values, and a fixed total preorder compares the vectors. This interface supports scalar reward maximization and lexicographic combinations. A cost can be negated when larger values are preferred. Equivalent vectors leave the existing incumbent unchanged.

A primitive must specify whether it counts the finite canonical representation or measures the repeating execution. For example, a transition reward g_{rew} has cycle-average value

$$\overline{g}_{\mathrm{rew}}(\lambda) = \frac{1}{L-\ell} \sum_{k=\ell}^{L-1} g_{\mathrm{rew}}(s_k, s_{\nu(k)}). \tag{16}$$

Here $\nu(L-1) = \ell$ and $\nu(k) = k+1$ at other positions. A sum over all $k \in \mathcal{K}$ instead measures one canonical prefix and one loop. The sum evaluates the finite representation; the cycle average measures long-run behavior. Canonicalization fixes its interpretation and prevents score changes caused solely by duplicating a solver's loop. Preference functions may rank a rejected proposal for search feedback. Only certified members of $\mathcal{F}_\Theta$ enter incumbent selection.

These semantics distinguish the outcomes returned by synthesis. A completed positive certificate establishes that the controllers are legal, the nominal word satisfies the system requirement, and no legal unilateral strategy gives a strict improvement. A negative equilibrium certificate identifies a player and a realizable profitable lasso. Failure of the system requirement rejects nominal feasibility independently of equilibrium. An unresolved check supplies neither certificate. Under a finite budget, synthesis therefore returns the best certified incumbent found or reports that none was certified. Candidate-domain optimality additionally requires exhausting or soundly excluding every strictly better candidate.

A.5 A worked passage example

The passage example applies the preceding definitions to one source game. Listing 1 specifies its legal moves and temporal goals. Each Boolean `meeting1` or `meeting2` records a robot's location. False means home and true means the meeting area. The commands after `init` put both robots at home. At a safe state, the first update command waits and the second moves by toggling the location. The third makes joint meeting occupancy absorbing. A prime denotes a successor value, and all right-hand sides read the current state. Joint assignments execute simultaneously. The `goal` clauses define player payoffs; `property` defines the nominal system requirement.

Listing 1: Guarded-command source of the recurring passage game.

```
module robot1 controls meeting1
init :: true ~> meeting1' := false;
update
:: !(meeting1 and meeting2) ~> meeting1' := meeting1;
:: !(meeting1 and meeting2) ~> meeting1' := !meeting1;
:: meeting1 and meeting2 ~> meeting1' := true;
goal
:: (G F !meeting1) and (G F meeting1)
   and G !(meeting1 and meeting2);

module robot2 controls meeting2
init :: true ~> meeting2' := false;
update
:: !(meeting1 and meeting2) ~> meeting2' := meeting2;
:: !(meeting1 and meeting2) ~> meeting2' := !meeting2;
:: meeting1 and meeting2 ~> meeting2' := true;
goal
:: (G F !meeting2) and (G F meeting2)
   and G !(meeting1 and meeting2);

property :: G !(meeting1 and meeting2);
```

Write $H = (0,0)$, $A = (1,0)$, $B = (0,1)$, and $C = (1,1)$ for the location bits. The proposition $collision$ abbreviates their conjunction, while $home_i$ abbreviates the negation of robot i's location bit. A joint move from H produces C. From A, robot 1 can return alone to H or both robots can move simultaneously to B. These successors follow the new joint location values.

Example 1 instantiates the recurring passage task from the introduction. Each move represents a completed traversal between a robot's home and the meeting area. Simultaneous occupancy of the meeting area is a collision. The example separates a successful finite round from an infinite execution satisfying the temporal goals.

Example 1 Recurring passage goals, unilateral deviations, and policy quality

Game. Both robots start at home in state H. State A places robot 1 at the meeting area and robot 2 at home. State B reverses these positions. State C records a collision and is absorbing. Initialization has a single legal joint choice leading to H. Each robot can choose move or wait at every safe state. A move toggles its location. Waiting retains its location. Joint meeting occupancy leads to C.

Objectives. The system requires $\mathsf{G}\neg collision$. Robot i requires $\mathsf{GF}\, home_i \wedge \mathsf{GF}\, meeting_i \wedge \mathsf{G}\neg collision$. Each location must recur infinitely often. State C has both meeting propositions true and violates the safety conjunct.

Controllers. Write 1 for robot 1 moving while robot 2 waits. Write 2 for the reverse and 0 for both waiting. Fast and Slow repeat their schedules with a clock initialized to zero. The clock advances modulo the schedule length. Once follows 1122 and then waits forever using a clock that saturates at four. Idle always waits. At safe states, outputs follow the clock. At C, both controllers retain their meeting bits. These rules define legal outputs on off-path states too.

Profile	Action word	Nominal word	Safe	(u_1, u_2)	NE	$\overline{g}_{\mathrm{rew}}$
Idle	0^ω	H^ω	Yes	$(0,0)$	No	0
Once	$1122\,0^\omega$	$HAHBH^\omega$	Yes	$(0,0)$	No	0
Fast	$(1122)^\omega$	$(HAHB)^\omega$	Yes	$(1,1)$	Yes	$1/2$
Slow	$(10102020)^\omega$	$(HAAHHBBH)^\omega$	Yes	$(1,1)$	Yes	$1/4$

Certification. All four nominal words are safe. Once visits each meeting location only finitely often and violates both recurring goals. A robot can follow the first round and then keep moving after its opponent parks. This legal unilateral deviation satisfies its own goal. Against Idle, robot 1 can move at every step to obtain $(HA)^\omega$ and improve from zero to one. Fast and Slow are equilibria because both players already receive payoff one.

Selection. Give reward one to each return from A or B to H and zero to other transitions. The reward is g_{rew} from Equation (16). Its cycle average is zero for Idle and Once. It is $1/2$ for Fast and $1/4$ for Slow. CTES rejects Idle and Once, then prefers Fast to Slow among the certified policies.

Synthesis from the source. Consider the finite candidate domain formed by these four schedule templates. The generator chooses a schedule and its periodic or saturating clock rule. For the illustrative order Idle, Once, Slow, Fast, Table 3 follows the certification loop. Nominal evaluation first checks the source property and both recurring goals. Each losing player then faces a deviation search with the opponent's complete controller fixed. Rejection requires a legal deviation that improves the losing payoff.

Table 3: Synthesis over the four passage templates. Every candidate passes the nominal safety requirement. The incumbent changes only after equilibrium certification.

Step	Proposal	Certification evidence	Incumbent after the step
1	Idle	Robot 1 can realize $(HA)^\omega$	None
2	Once	A robot can continue trips after parking	None
3	Slow	Both Boolean payoffs equal one	Slow, $\overline{g}_{\mathrm{rew}} = 1/4$
4	Fast	Both Boolean payoffs equal one	Fast, $\overline{g}_{\mathrm{rew}} = 1/2$

The first two rejections in Table 3 use legal history-dependent deviations. Against Once, following the finite round and then continuing trips changes a losing payoff to one. Slow and Fast already give both players payoff one, so certification passes them to selection. Their return rates then determine the incumbent update. Exhausting the four-template domain establishes Fast as its best candidate.

The source verifier and an explicit-game reference reproduce all four nominal outcomes and equilibrium judgments in Example 1. These checks determine feasibility under the source game; cycle-average return then determines selection among the certified profiles.

The difference between rejection and acceptance depends on the infinite continuation. Once and Fast share the finite prefix $HAHBH$. Fast continues to alternate trips whereas Once remains at H. Finite task completion therefore cannot determine their recurring payoffs. Fast also reaches H at clock positions zero and two with different next actions. Its full configuration first repeats after four transitions. Both temporal evaluation and deviation replay must preserve the memory needed to repeat the behavior indefinitely.

The deviation $(HA)^\omega$ against Idle also illustrates transfer. Changing robot 2's rule only at B leaves the recorded states H and A unchanged. With its initial action and one-state memory preserved, the witness still replays and robot 1 still loses nominally. The same witness rejects this changed profile. If robot 1 instead adopts its successful deviation as its nominal controller, the fixed opponent still reproduces the witness. Robot 1 now receives payoff one, so replay cannot justify rejection for that player.

Replaying the Idle witness against Fast fails at the first opponent-memory update. The recorded opponent memory stays at zero whereas Fast advances to one. Fast also makes robot 2 move later where the witness requires it to wait. These changes invalidate the infinite behavior under the current opponents. Faster crossings alone would not improve a winning player's Boolean payoff. The example thus gives distinct roles to recurrence, strategic improvement, and external quality.

A.6 Symbolic proposal semantics

The example enumerates schedule templates. Symbolic generation instead encodes the controller choices and their nominal execution in one query. For a proposal-length bound $B_{\mathrm{out}} \geq 1$, it represents θ and $L \leq B_{\mathrm{out}}$ lasso positions. Every edge follows Equation (5). Closure $x_L = x_\ell$ returns to loop entry $0 \leq \ell < L$. The query is

$$\Phi_{\mathrm{cand}} = \Phi_{\mathrm{legal}} \wedge \Phi_{\mathrm{run}} \wedge \Phi_{\mathrm{sys}} \wedge \Phi_{\mathrm{query}} \wedge \bigwedge_{w \in \mathcal{C}} \Phi_{\mathrm{block}}(w). \tag{17}$$

Φ_{legal} enforces controller legality, Φ_{run} nominal transitions, and Φ_{sys} the system requirement. Φ_{query} encodes the preference query. Witnesses retained in $\mathcal{C}$ supply symbolic blocks through Φ_{block} (Section 4.5). After decoding, the canonical lasso determines the score used for incumbent selection.

The controller constraints fix a template with finite memory and parameter domains. Its parameters determine initial actions and the output and update rules. A full table uses the same action variable whenever (m_i, s) recurs and shares a memory-update variable across visits to (m_i, s, s'). Template expressions impose the corresponding consistency conditions. The legality constraint includes source and controller admission throughout the controller domain, including deviation histories.

Given those controller choices, the run constraints initialize through T_0 and restore the full state–memory configuration at loop entry ℓ. Temporal truth is evaluated on the infinite word represented by that loop. Decoded nominal execution stops at the first repeated full configuration. Preference primitives evaluate this canonical representation or an equivalent trace-invariant quantity such as a cycle average.

Score consistency further requires a common interpretation of the represented lasso. The evaluated memoryless action-table encoding requires all L represented physical states to be pairwise distinct before the closing edge. Its symbolic lasso is therefore canonical. The reward objective counts weighted proposition occurrences over that prefix and one loop. Decoded scores use the same canonical positions. This aligns the symbolic reward with incumbent selection. For the general interface, proposal priorities guide exploration while recomputed canonical scores determine which certified policy is retained.

The system and preference query constrain the nominal word. Certification fixes the decoded opponents and retains their exact memories while allowing every legal deviator action. The outcome bound restricts proposals while Equation (4) quantifies over unrestricted legal deviations. General LTL certification includes automaton construction, successor generation, and accepting-product search. Appendix C.1 separates these costs.

Certification can then return a witness that constrains later queries. For each such witness, the rejection condition encodes $u_i(\pi) = 0$ as $\lambda_\theta \models \neg\varphi_i$. Witness satisfaction is already established. The remaining replay constraints check current opponent initialization and transitions along the stored word.

Equation (11) gives the semantic condition for excluding a family of unstable candidates. The reward-trap implementation uses a conservative structural block for its memoryless action tables. Let a^0_{old} and $a_{\mathrm{old}}(s)$ be the rejected profile's initial joint action and joint action table. Let V_w contain all physical states visited by its nominal execution and the validated deviation. For a new candidate with initial action a^0 and table $a(s)$, the block is

$$\Phi_{\mathrm{struct}}(w) = (a^0 \neq a^0_{\mathrm{old}}) \ \vee \ \bigvee_{s \in V_w} \big(a(s) \neq a_{\mathrm{old}}(s)\big). \tag{18}$$

It excludes candidates preserving the initial joint action and the complete joint action table on both executions. These equalities preserve the nominal loss and the profitable deviation, so the excluded family satisfies the rejection condition in Equation (11). Fixing the complete joint actions is more restrictive than matching only the opponents along the witness. The reward-trap candidates have no controller-memory variables. Extending this block to stateful templates also requires preserving the memory evolution of both executions.

Structural blocks accumulate across candidate attempts within one symbolic query. Changing the lasso length or loop entry creates a new solver without those blocks. The deviation bank retains its witnesses across these query boundaries and checks decoded candidates directly.

Algorithm 2 Complete CTES search and certification loop

Require: Game $\mathcal{G}$, space Θ, generator Gen, initial sequence $\mathcal{H}_{\mathrm{init}}$, budget b
Initialize Gen with $\mathcal{H}_{\mathrm{init}}$ first; set $\mathcal{B} \leftarrow \emptyset$, $\pi^\star \leftarrow \perp$.
while budget remains and Gen has a proposal **do**
  Generate π_τ and evaluate legality, nominal lasso, and preference score.
  Set $\mathsf{ok}_\tau \leftarrow 0$ and $w \leftarrow \perp$.
  if evaluation completes and the legal profile satisfies φ_{sys} **then**
    if $I_0(\pi_\tau) = \emptyset$ **then**
      Set $\mathsf{ok}_\tau \leftarrow 1$.
    else
      Seek w in the bank satisfying structural replay and current profitability.
      if no profitable witness is established **then**
        Search for deviations within the remaining allocation.
        Validate and store any new witness w within the insertion allocation.
        Set $\mathsf{ok}_\tau \leftarrow 1$ only if absence is proved for every nominal loser.
      end if
    end if
    Update the incumbent by Equation (21).
  end if
  Supply available nominal feedback and add a block for validated w when enabled.
  On completing $\mathcal{H}_{\mathrm{init}}$, seed the parameter frontier once from $\pi^\star$ or its default root.
end while
return $\mathsf{Rec}(\pi^\star)$, or no certified policy found within the budget.

B Certification Procedures and Cost Accounting

CTES implements the semantics of Appendix A through admission, replay, and fresh verification within a shared budget. This section details those procedures and accounts for both the search avoided by replay and the overhead it adds.

B.1 Complete synthesis loop

Algorithm 2 expands the branching summarized in Algorithm 1, applying the same evaluation and certification steps to constructed and subsequently generated profiles. The generator evaluates the initial sequence before expanding its frontier. Nominal feedback remains available after strategic rejection, while incumbent updates require completed certification, recorded by ok_τ. The output $\mathsf{Rec}(\pi^\star)$ contains the incumbent and its evaluated record. Appendix C.5 formalizes both the update and its guarantee.

B.2 Admission and replay checks

The loop can use a witness only after establishing that the game and candidate satisfy Section 3. Global checks establish legal controller actions and deterministic environment responses. Transition evaluation then checks each recorded edge against the game rules. Table 4 separates these obligations from temporal satisfaction and the current payoff comparison.

Table 4: Validation obligations for reusable deviation evidence. Admission applies to the current candidate before a stored behavior can justify rejection.

Obligation	Required condition
Global admission	Nonempty legal actions, deterministic transitions, and legal controller outputs
Initialization	Legal a^0, $T_0(a^0) = s_0$, and matching opponent initial actions and memories
State transitions	Every recorded joint action is legal and produces its recorded successor
Opponent behavior	Current outputs and memory updates satisfy Definition 1, including the loop closure
Temporal improvement	The witness satisfies φ_i and the current nominal execution violates it
Fallback	An unsuccessful or unresolved replay leaves the deviation search to the complete verifier
Budget handling	A candidate remains unresolved unless a definitive decision completes within its deadline; unfinished optional insertion preserves a completed decision

The evaluated bank repeats admission once before trying a nonempty batch of witnesses. It tries witnesses in reverse insertion order, recomputing the nominal outcome and reevaluating the original goal on the stored word for each attempt. The measured costs include these repeated evaluations and attempts that fail the current-payoff check. Algorithm 2 makes explicit the logical shortcut for nominal winners. An empty bank invokes the backend directly.

Retention and budget allocation determine which witnesses remain available to this procedure. The evaluated default capacity is eight witnesses. Each independent run creates an empty bank and fresh backend sessions. A hit or duplicate insertion leaves the order unchanged. When capacity is exceeded, the oldest inserted witness is removed. Let b_{rem} be the candidate time allocation remaining when the bank is invoked. Replay receives at most $\min(1.5\,\mathrm{s}, 0.15 b_{\mathrm{rem}})$. On a miss, the backend receives the remaining time. Validation and insertion of a new witness receive at most $\min(1\,\mathrm{s}, 0.10 b_{\mathrm{rem}}, \tau_{\mathrm{left}} - 0.05 b_{\mathrm{rem}})$, where τ_{left} is the time still available. Insertion proceeds only when this allocation is positive. An unfinished insertion preserves a completed decision if the overall candidate deadline is met.

B.3 Verification backends

If replay does not establish rejection, the remaining allocation passes to a verification backend. All backends use the same decision criteria: a profitable deviation justifies rejection, and completed absence proofs for all nominal losers justify equilibrium. They differ in how they obtain this evidence.

Exact. After current admission and nominal-outcome evaluation, each nominally losing player is checked separately. The deviator's actions are unrestricted legal source actions; opponents follow their controllers. A configuration contains all physical propositions and the exact opponent-memory tuple. Satisfiability modulo theories (SMT) queries enumerate every distinct legal successor on demand. The lazy LTL product search looks for an accepting lasso for the original goal, without constructing the full input game in advance. An accepting lasso is independently replayed against the original profile to establish a profitable deviation. Exhausting the complete reachable product without one establishes absence for that player. NE requires absence for every nominal loser; Boolean nominal winners cannot improve. Exhausting configuration, product, solver, or time limits leaves the check unresolved. The default bounds are 10,000 configurations and 20,000 product states.

The LTL implementation builds a tableau product on demand. It first converts the formula to negation normal form, expands temporal obligations along reachable deviation transitions, and searches strongly connected components under generalized Büchi acceptance. An accepting infinite path must meet every acceptance set infinitely often. The source parser supports Boolean connectives and recursively nested X, F, G, and U. Release R is an internal operator used for temporal duals during normalization. The verifier constructs the tableau internally. Appendix C.1 relates this acceptance condition to the Büchi formulation used in the proof.

Chain. Chain supplements Exact with reusable evidence of deviation absence. Its positive bank stores up to eight portfolios of finite implication chains whose nodes are physical propositions or typed opponent-memory conditions. A chain starts at a necessary safety/recurrence trigger and ends at a condition inconsistent with the goal's safety part. Each chain edge is checked against the full current deviation relation. A validated chain therefore rules out a successful deviation. Each losing player requires one complete valid alternative from the portfolio. The evaluated scheduler first tries old templates and performs an Exact probe. It then discovers or prepares checked chains within the remaining budget before an Exact fallback. The template, probe, and discovery fractions are 0.15, 0.4, and 0.6 of the time available on entry to Chain. In Chain + bank, this is the budget remaining after bank replay. These fractions cap successive stages subject to remaining time and are not additive allocations. The scheduler normally reserves 5% of that entry budget for final checks and serialization. Discovery after an unresolved probe reserves 10%. Appendix D gives node and expansion limits. Successful positive checks or exhaustive Exact search establish absence of profitable deviations. Rejection requires a checked concrete deviation. Optional warming preserves completed Exact decisions.

Rel-summary. This native baseline first attempts checked implication-chain or bounded Boolean-feature ranking summaries that rule out a satisfying infinite deviation. Every summary obligation is rechecked against the full current candidate-conditioned relation. If this fails, existential predicate abstraction retains exact bounded opponent memories. Its predicate stages use goal atoms, then syntax-derived relational predicates, then all physical atoms as well. Absence of accepting abstract paths establishes absence of deviations only after a separate initial and transition coverage check. An accepting abstract lasso must be concretized and independently replayed to certify a profitable deviation. Failed concretization triggers refinement. The evaluated limits are 10,000 configurations, 20,000 product states, and 128 refinements. Ranking synthesis allows 128 iterations with integer coefficients bounded in magnitude by $2^{\min(|AP|,24)}$. Unfinished refinement or resource exhaustion leaves the check unresolved.

B.4 Bank cost accounting

Replay substitutes bank work for some backend calls. To account for both avoided search and added overhead, consider K_{cand} candidates indexed by $j = 1, \ldots, K_{\mathrm{cand}}$. Let τ_j^{base} be baseline verification cost and $\mathcal{J}$ the bank-hit indices. On a miss, fallback costs τ_j^{fall}. All additional bank work is included in τ_j^{aux}. For total costs C_{base} and C_{bank}, the difference is

$$C_{\mathrm{base}} - C_{\mathrm{bank}} = \sum_{j \in \mathcal{J}} \tau_j^{\mathrm{base}} - \sum_{j=1}^{K_{\mathrm{cand}}} \tau_j^{\mathrm{aux}} - \sum_{j \notin \mathcal{J}} (\tau_j^{\mathrm{fall}} - \tau_j^{\mathrm{base}}). \tag{19}$$

Net savings depend on whether avoided search outweighs admission, replay, and insertion costs after accounting for changes in fallback cost. A miss retains the full verification problem and its worst-case complexity.

C Soundness and Selection Proofs

The procedures in Appendix B require two distinct guarantees. Fresh verification must cover unrestricted legal deviations, while bank replay must establish a valid rejection for the current profile. The finite-product argument proves the first. The realizability lemma and payoff comparison prove the second, followed by conditions for transfer across profiles. Sound decisions then support the incumbent-selection invariant.

C.1 Completeness of fixed-opponent lasso search

Fix a nominally losing player i. Holding the opponents' deterministic controllers fixed gives a finite action graph on

$$\mathcal{V}_i = \mathcal{X} \times \prod_{j \neq i} M_j.$$

Its initial configurations result from all legal initial actions of i together with the fixed opponent initial actions and memories. At a configuration (s, q), player i may choose any action in $A_i(s)$. The opponent outputs determine the remaining action components. The maps T and μ_j then determine a unique successor for that choice. Branching represents alternative deviator choices rather than random transitions.

Every pure history-dependent deviation induces an infinite path in this graph. Conversely, a chosen infinite path defines the deviator's actions on its successive finite histories. Arbitrary legal actions extend the strategy to other histories. Thus a profitable deviation exists exactly when the graph has a path satisfying φ_i.

To decide whether such a satisfying path exists for general LTL, synchronize this graph with a nondeterministic Büchi automaton $\mathcal{A}_i$ for φ_i. Its state set is Q_i and its accepting states are $F_i \subseteq Q_i$. The product is $P_i^\pi = D_i^\pi \otimes \mathcal{A}_i$. Initial vertices pair legal graph configurations with automaton states reached after consuming their physical-state labels. Each edge then consumes the successor label. This convention evaluates the formula from s_0 and preserves next-step semantics.

Write $\mathrm{RSCC}(P)$ for the reachable strongly connected components, whose vertices are mutually reachable. A component is cyclic when it contains a nonempty directed cycle. The accepting-product criterion is

$$\mathsf{Dev}_i(\pi) \iff \exists \mathcal{U} \in \mathrm{RSCC}(P_i^\pi)\ \big[\mathsf{Cyclic}(\mathcal{U}) \wedge \mathcal{U} \cap (\mathcal{V}_i \times F_i) \neq \emptyset\big]. \tag{20}$$

The product accepts exactly the satisfying graph paths (Vardi & Wolper, 1986; Baier & Katoen, 2008). An accepting component supplies a finite prefix and a closed walk through an accepting vertex. Projection gives the state–memory lasso and actions in Equation (9). Conversely, every satisfying deviation induces an accepting infinite product path. This proves completeness against unrestricted pure history-dependent deviations. It does not require a memoryless deviator in the physical state space.

The implementation uses generalized Büchi acceptance to require fulfillment of pending until obligations. A reachable cyclic component admits an accepting closed walk when it can meet every acceptance requirement. Strong connectivity lets the walk visit those requirements repeatedly. Equivalently, a finite obligation index can cycle through the acceptance sets to obtain an ordinary Büchi automaton recognizing the same state words. The ordinary Büchi formulation above therefore gives the same existence criterion as the generalized acceptance used by the tableau product.

Simple goals admit direct graph tests. Reachability $\mathsf{F}\psi$ needs a reachable ψ-state. Safety $\mathsf{G}\psi$ needs a reachable cycle along a path inside ψ. Recurrence $\mathsf{GF}\psi$ needs a reachable cyclic component intersecting ψ. Eventual permanence $\mathsf{FG}\psi$ permits an unrestricted prefix followed by a cycle inside ψ. Safe recurrence restricts both prefix and cycle to safe states before testing recurrence.

The product has at most $|\mathcal{X}|\,|Q_i| \prod_{j \neq i} |M_j|$ vertices, apart from an optional initial selector. If its explicit graph has V_i vertices and E_i edges, reachability and accepting-cycle detection take $O(V_i +$

E_i) time and storage. Constructing successors from a symbolic game adds its own cost. LTL translation can make $|Q_i|$ exponential in the formula length. With a common memory bound $m_{\max}$, the opponent factor is at most $m_{\max}^{N-1}$. These factors govern the cost in the formula length and number of stateful opponents. Resource exhaustion leaves the search unresolved.

C.2 Proof of Lemma 1

Lemma 1 (Realizability of a replayed lasso)**.** Assume the game conditions of Section 3. If w structurally replays against π, a legal pure history-dependent strategy σ_i realizes exactly its state word against π_{-i}.

Proof. We construct a legal strategy from the stored lasso and then show that the current opponents reproduce it. Fix a game and current profile satisfying Section 3. Let w satisfy Definition 1. Define $\iota(t) = t$ for $t < L$ and $\iota(t) = \ell + ((t - \ell) \bmod (L - \ell))$ for $t \geq L$. This is well-defined because $L - \ell \geq 1$. Use $s_t^w = s_{\iota(t)}$ for the unfolded state sequence. Define $q_t^w = q_{\iota(t)}$ and $a_t^w = a_{\iota(t)}$ in the same way. The successor relation gives $\iota(t+1) = \nu(\iota(t))$ for every $t \geq 0$.

The deviating strategy σ_i initially chooses the ith component of a^0. At the history $s_0^w \cdots s_t^w$ it chooses $a_{i,t}^w$. At every other legal history it chooses a fixed legal action at the final state. Such a choice exists because every action set is nonempty. This defines a pure legal strategy on all legal histories. Histories ending in the same physical state can have different lengths. Distinct actions at different lasso positions are therefore consistent with history dependence. The construction imposes no bound from the candidate's memory.

We prove by induction that the resulting execution has state s_t^w and opponent memory q_t^w. At initialization, each opponent uses its recorded initial action by structural replay. The deviator uses its recorded component by construction. Legality and $T_0(a^0) = s_0$ establish the first state. Each recorded opponent initial memory equals its current initial memory. This gives the base case.

Suppose the assertion holds at time t. The history matches the witness and σ_i chooses $a_{i,t}^w$. Each opponent chooses the recorded component because $\alpha_j(q_{j,t}^w, s_t^w) = a_{j,t}^w$. The joint action is legal. Deterministic execution gives $T(s_t^w, a_t^w) = s_{t+1}^w$. Each opponent then updates to $\mu_j(q_{j,t}^w, s_t^w, s_{t+1}^w) = q_{j,t+1}^w$. These equalities also apply to the closing edge into ℓ. The induction therefore holds through every loop traversal. The resulting infinite word is exactly the stored lasso word. The same construction covers a one-position self-loop and a loop beginning at position zero. It also permits a nonempty prefix. □

C.3 Proof of Theorem 1

Theorem 1 (Sound negative reuse). Under the game conditions of Section 3, a structurally replayable witness with $u_i(\pi) = 0$ and $\lambda_w \models \varphi_i$ proves that π is not a Nash equilibrium.

Proof. Admission establishes the required source and controller assumptions. Structural replay supplies the premises of Lemma 1. That lemma gives a legal pure strategy σ_i whose outcome against the current π_{-i} is the witness word. Witness satisfaction gives $u_i(\sigma_i, \pi_{-i}) = 1$. The current-payoff check gives $u_i(\pi) = 0$. This player and strategy therefore violate the equilibrium inequality in Equation (4). The conclusion applies to the current candidate regardless of which profile produced the witness. □

C.4 Proof of Proposition 1

Proposition 1 (Sufficient witness-preserving conditions)**.** Suppose w is accepted for π and let π' be another valid profile of the same game with the same goals. Assume every opponent retains its recorded initial action and memory. At each lasso position it must also produce the recorded action and memory successor. If $u_i(\pi') = 0$, then w is accepted for π'.

Proof. It suffices to verify the replay and profitability premises for the new profile. The game is fixed, so the recorded actions remain legal and retain their source successors. In particular, $T_0(a^0) = s_0$ remains true. The hypotheses for π' preserve each opponent's initial action and memory. They

also preserve its output and memory successor at every recorded position. These conditions include the closing edge and establish Definition 1 for π'. The state word and goal are unchanged. The word consequently still satisfies φ_i. The assumption $u_i(\pi') = 0$ establishes current profitability. Theorem 1 applies to π'. □

C.5 Proof of Proposition 2

Proposition 2 (Certified-incumbent invariant)**.** Assume correct model admission and nominal evaluation together with sound backend decisions. At each completed CTES iteration, an existing incumbent belongs to $\mathcal{F}_\Theta(\mathcal{G})$ and is preference-maximal among all candidates certified feasible so far.

Proof. At search iteration τ, set $\mathsf{ok}_\tau = 1$ once legality and $\mathsf{Cert}(\pi_\tau) = 1$ are established and zero otherwise. The symbol $\perp$ denotes absence of a candidate or witness. Starting from $\pi^\star_0 = \perp$, selection updates the incumbent by

$$\pi^\star_\tau = \begin{cases} \pi_\tau, & \mathsf{ok}_\tau = 1 \ \wedge \ \big[\pi^\star_{\tau-1} = \perp \ \vee \ q_\Gamma(\pi_\tau) \succ q_\Gamma(\pi^\star_{\tau-1})\big], \\ \pi^\star_{\tau-1}, & \text{otherwise.} \end{cases} \tag{21}$$

Equivalent scores retain the earlier incumbent. The returned record $\mathsf{Rec}(\pi^\star)$ contains the policy and its canonical lasso together with system satisfaction, player payoffs, preference score, and completed certification.

The same update covers constructive candidates and later proposals. Budget exhaustion preserves the current record.

Fix the game and candidate space together with the total preference preorder. Let $\mathcal{F}^{\mathrm{seen}}_\tau$ contain the candidates certified feasible by the end of iteration τ. Membership requires completed legality and system checks together with a sound equilibrium decision. We prove that the incumbent is absent exactly when $\mathcal{F}^{\mathrm{seen}}_\tau$ is empty. Otherwise it belongs to $\mathcal{F}^{\mathrm{seen}}_\tau \subseteq \mathcal{F}_\Theta(\mathcal{G})$ and no member of $\mathcal{F}^{\mathrm{seen}}_\tau$ is strictly preferred to it.

Initially $\mathcal{F}^{\mathrm{seen}}_0$ is empty. Candidates with incomplete checks or violated legality or system requirements leave the set and incumbent unchanged. A bank hit exhibits a profitable deviation by Theorem 1. It cannot add a feasible member. A freshly verified profitable deviation has the same consequence.

A candidate passing the system check with a completed equilibrium proof belongs to $\mathcal{F}_\Theta(\mathcal{G})$. It is added to $\mathcal{F}^{\mathrm{seen}}_\tau$. If the incumbent is absent, this candidate becomes the incumbent. Otherwise the total preorder compares it with the incumbent. Replacement occurs only on strict improvement. A worse or equivalent candidate leaves the incumbent unchanged. Both cases preserve membership and preference maximality. Induction establishes the claim. Optional bank insertion after a completed decision does not affect this argument.

Symbolic blocking excludes profiles for which structural replay and strict Boolean improvement both hold. Theorem 1 ensures that Equation (11) cannot remove a member of $\mathcal{F}_\Theta$. A conservative block excludes a subset of those profiles and preserves the same property. The incumbent invariant itself does not require a complete generator. Unvisited candidates and unresolved checks make no contribution to $\mathcal{F}^{\mathrm{seen}}_\tau$. The record therefore establishes the best certified policy found. Global optimality requires a completed argument excluding all strictly better candidates. □

C.6 Role of the replay checks

The preceding proofs use initialization, infinite replay, and current profitability at separate steps. Closing only the physical state can create a false loop when an opponent's memory changes its next action. Checking the memory successor at the closing edge prevents that error. A finite prefix alone cannot establish infinitely recurring trips. The original goal must be evaluated on the complete infinite lasso. A policy update can also make a previously losing player a nominal winner. Rechecking $u_i(\pi) = 0$ establishes that an improvement is still possible. Finally, a legal deviation may change the initial action. Recording only update actions would omit such a behavior. The controlled cases in Appendix F.1 exercise these distinctions; Appendix F.2 additionally tests strong-until fulfillment.

D Experimental Protocol and Validation

The experiments measure certified output quality, test the certification conditions, and evaluate the cost of reuse. Matched resource streams isolate certification cost, while online search measures the effect of reuse under a shared deadline. Table 5 summarizes the workloads and budgets; the protocols below define candidate domains, stopping rules, and aggregation units.

Table 5: Experimental workloads and budgets. Counts are runs except for the two semantic suites. Each suite shares a separate 300-second allocation. Candidate and run limits are seconds.

Experiment	Count	Candidate	Run	Sizes and generation
Coupled contention	480	300	600	$n = 3, 5, 6, 8, 10, 12$, ten instances
Cold-start controls	240	300	600	$n = 3, 4, 5$, ten instances each
Contention enumeration	4	300	1800	$n = 2, 3$, two instances each
Reward trap	40	300	1800	$n = 2, 3$, symbolic queries
Resource fixed streams	150	10	340	$n = 3, 4, 5$, 32 candidates
Resource search throughput	90	10	60	$n = 3, 4, 5$, parameter frontier
Bank capacity	24	10	340	$n = 3, 5$, capacities 1, 4, 16
Candidate order	16	10	340	$n = 3, 5$, shuffled streams
Controlled replay	9	n/a	300	Two-player semantic cases
Strong-until checks	6	n/a	300	Two-player temporal cases

D.1 Synthesis controls and budgets

The symbolic synthesis control uses reward trap to separate reward optimization from equilibrium certification. Reward trap uses one fixed game at each of $n = 2, 3$, with four and eight physical states respectively. Seeds 20–24 control the solver and each has two timing repetitions. Each run has a 10,000-candidate guard and an eight-entry bank. Both preference configurations evaluate the true nominal score and use it for incumbent selection.

Reward trap synthesizes memoryless action tables over the explicit physical states. Queries increase the nominal lasso length from one to the number of states. For each length L, loop entries are visited from zero to $L-1$. A first phase seeks a system-feasible policy satisfying every player's goal under a shared 64-candidate allowance. If it finds none, a system-only feasibility phase can use the remaining global allowance. Either feasibility phase ends upon its first certified policy. Preference refinement receives a shared allowance of 16 candidate attempts across its queries. It advances to the next query after the first certified policy for that length and loop entry. Rejected candidates can trigger query-local repair while consuming the phase allowance. Disabling guidance removes symbolic objective optimization during refinement. It preserves the query order and actual-score incumbent selection.

Both reward-trap and coupled-contention synthesis cap each source certification and parameter-feedback evaluation at 300 seconds, truncated by the remaining run time. Native SMT calls within certification and replay use a 500-millisecond limit, whereas symbolic proposal queries receive the remaining run time. Generation, evaluation, repair, and bank operations share the run deadline. An unresolved certification or a feedback or certification timeout ends the current run with its incumbent retained. Search also stops when its candidate allowance or frontier is exhausted and can therefore finish before the run cap. Resource-search continuation after an unresolved candidate is specified separately below.

D.2 Coupled contention and enumeration controls

Coupled contention evaluates parameter search and constructive initialization. The main comparison uses instance identifiers 300–309 and search seeds 60 and 61. These games are separate from initializer development. Earlier cold-start controls use instances 0–9 and search seeds 30 and 31 at $n = 3, 4, 5$. Instance generation uses seed $100000n + \mathrm{id}$ for size n and instance identifier id. A randomly ordered spanning path ensures connectivity. Additional edges are sampled across all unordered player pairs with probability 0.25, 0.55, or 0.85 according to $\mathrm{id} \bmod 3$. Their union with

the path forms the graph. Player weights are uniform integers from 1 to 9. Eligible phase sets are sampled from $\{0,1\}$, $\{0,1\}$, $\{0\}$, and $\{1\}$ with equal probability. The repeated two-phase set gives it probability one half.

Two global phases alternate, beginning in phase zero. Player i freely chooses its next request. Its next service is its current request conjoined with the absence of any neighbor request. Its next collision is its current request conjoined with at least one neighbor request. All three atoms are initially false. The goal requires infinitely recurring service originating in an eligible request phase and global absence of that player's collisions. The goal accounts for the one-step delay between request and service. The system requirement is true. Source-level certification permits arbitrary legal unilateral request histories.

Each candidate uses two memory states initialized at zero and one of four schedules, consisting of no requests, either individual phase, or both phases. A Boolean parameter enables a transition to memory one when the player observes its own request together with a neighbor request. Memory one is absorbing and issues requests permanently. With this response disabled, memory remains zero. The schedule and response parameters give 8^n joint candidates. The preference vector is $(W, S, -R)$. For a nominal lasso with loop positions $\ell, \dots, L-1$, define the per-player rates

$$f_i^{\mathrm{svc}} = \frac{1}{L-\ell} \sum_{k=\ell}^{L-1} \mathbf{1}[\mathit{service}_i \in s_k], \qquad f_i^{\mathrm{req}} = \frac{1}{L-\ell} \sum_{k=\ell}^{L-1} \mathbf{1}[\mathit{request}_i \in s_k].$$

Here $\mathit{service}_i$ and $\mathit{request}_i$ are the player's service and request propositions. With the generated positive weight β_i, the aggregate metrics are $S = (\sum_i \beta_i f_i^{\mathrm{svc}})/(\sum_i \beta_i)$ and $R = n^{-1} \sum_i f_i^{\mathrm{req}}$. Both average only the repeating loop. Welfare W instead evaluates each player's full temporal goal on the infinite word. Service can therefore be positive even when a player's safety or eligible-phase obligation fails.

The main comparison first constructs and certifies a weighted candidate and then a count-oriented candidate. It chooses their best certified $(W, S, -R)$ vector with ties retaining the weighted candidate. Both checks count toward the 10,000-candidate guard and 600-second deadline, including when they construct the same profile. Each bank starts empty with capacity eight. The no-bank configuration disables it during initialization and search. An unresolved check ends the run with any earlier incumbent retained. If both constructions are rejected, search starts from the idle controller. Construction alone then has no incumbent.

The weighted constructor orders players by weight divided by degree plus one, with seeded ties. The count-oriented constructor instead uses one divided by degree plus one and seeded random ties. Both constructors use the same phase assignment and spare-slot rules. Each assigns an eligible phase unused by the player's neighbors if one is available. Otherwise the player stays idle. If both phases are available, it selects the one assigned to fewer players globally and then prefers phase zero. A second pass visits already assigned players in descending weight order, breaking weight ties by player index. It tries phase zero and then phase one to fill spare collision-free slots, including slots outside the player's eligible set. All persistent responses are disabled. The earlier greedy baseline certifies only the weighted construction. The main construction-only baseline certifies both and returns their best feasible vector. Construction and certification both count toward runtime.

After initialization, all search variants use the same one-coordinate neighborhoods and enqueue each vector once. If the frontier encounters an initial profile, it reuses that profile's recorded feedback. Guided search assigns the first proposing parent's score as queue priority, while unguided search uses $(0, 0, 0)$. Both use seeded tie-breaking and actual scores for incumbent selection. Earlier cold-start search begins with every player idle and every persistent response disabled.

Complete enumeration supplies finite-domain reference values for the smallest games. Four separate enumerations visit the parameter Cartesian product in fixed order for $n = 2, 3$ and instances 0 and 1. They use Exact without the bank and solver seed 30. Their candidate guards are exactly 64 and 512 under 1800-second run caps. A reference is complete only after every profile has been checked. These references provide no feedback to generation.

D.3 Complete resource fixture

The resource game supports fixed-stream certification and fixed-budget search. Its clients and monitors are indexed by $i = 0, \ldots, n-1$. For each client i, the source permits either Boolean value for the next request but fixes the next service to $request_i(t) \wedge token_i(t)$. Each monitor can legally choose either next penalty value. All request, service, and penalty atoms are initially false. Exactly $token_0$ is initially true and $token_i(t+1) = token_{(i-1) \bmod n}(t)$. Legal deviators change their own requests or penalties under the same source rules.

A client candidate has one memory state. Its service output obeys the transition equation. Idle requests are false and late requests follow $token_i$. The search-throughput experiment also allows anticipatory requests following $token_{(i-1) \bmod n}$. A monitor has memory $m \in \{0,1\}$, initially zero, and Boolean parameters $(\delta_{\mathrm{mem}}, \delta_{\mathrm{now}}) \in \{0,1\}^2$ for remembered and immediate response. Its exact rules are

$$penalty_i(t+1) = \begin{cases} \delta_{\mathrm{now}} \wedge service_i(t), & m(t) = 0, \\ 1, & m(t) = 1, \end{cases} \qquad m(t+1) = \begin{cases} \delta_{\mathrm{mem}}, & service_i(t) = 1, \\ 0, & service_i(t) = 0. \end{cases}$$

All candidate initial actions are empty. Client i seeks $\mathsf{GF}\, service_i \wedge \mathsf{G}\neg penalty_i$. Monitor i seeks $\mathsf{G}\neg penalty_i$. The system requirement is true. Thus n client–monitor pairs form a $2n$-player game. For fixed-stream certification, the baseline parameter vector is $(0,1,0)^n$. At step t the bits of t xor $\lfloor t/2 \rfloor$ toggle a seeded permutation of the $3n$ coordinates. Here xor is bitwise exclusive OR and $\lfloor \cdot \rfloor$ rounds down. This Gray-code order changes one coordinate between consecutive profiles. The first 32 profiles are used without filtering. Each 32-point Gray-code prefix varies only five of the seeded coordinates. These streams measure reuse under local controller updates. Each client has a separate strategic monitor, and the five varied coordinates define a local slice of the full parameter space.

Late requests miss service because $service_i(t+1)$ depends on the old request. Anticipatory requests align with the rotating token and can change nominal welfare. The search-throughput preference vector is $(W, S_{\mathrm{res}}, -R_{\mathrm{res}})$. Here W averages Boolean player payoffs, while S_{res} and R_{res} average service and request atoms over all clients and all states of the nominal cycle. Generation uses this nominal feedback even for rejected candidates. Certified selection uses it only for feasible equilibria.

D.4 Fixed-stream and throughput controls

The fixed-stream control isolates certification cost on a local slice of the resource family. It uses seeds 5–9 with two timing repetitions. Each fixed stream contains the first 32 profiles in each seeded Gray-code stream. Five backends give 150 resource runs. Its 340-second run cap includes a 20-second setup allowance and 32 ten-second candidate allocations.

The throughput experiment then lets search choose candidates within a shared deadline. It compares Exact, Exact + bank, and Rel-summary under 60-second search budgets. Each backend starts from $(0,1,0)^n$ and follows the nominal-feedback frontier without symbolic blocking. Three request modes and two Boolean monitor parameters per client give 12^n profiles. A one-coordinate neighbor enters the queue once with its first parent's score. The search seed resolves equal priorities. Each backend starts with fresh sessions and an empty bank where applicable.

An unresolved resource candidate leaves the incumbent unchanged, and search continues with the available nominal feedback. If nominal feedback is unavailable, its neighbors receive the queue priority $(-1,-1,-1)$ below every valid preference vector. Resource search stops at the run deadline, the 4096-candidate guard, or an empty frontier. All 90 throughput runs reach the time budget. Fixed-stream verification also proceeds after an unresolved result. It stops after all 32 candidates or at the run deadline. Unresolved entries remain in the coverage denominator.

The capacity experiment uses seeds 5 and 6 with capacities 1, 4, and 16. Its capacity-eight control reuses the matching fixed-stream runs. The ordering experiment shuffles the same candidate sets using the workload seed plus 104729. It compares Exact and Exact + bank at capacity eight. The native solver-call timeout is 500 milliseconds. Chain has a 256-node limit, a 4096-expansion limit, and an implication-edge cache of at most 4096 entries. Remaining run time can shorten any individual allocation.

D.5 Aggregation and validation

The aggregation unit follows the source of variation in each workload. Resource timing uses seeds as units after averaging repetitions; contention quality uses games after averaging search seeds. For a fully observed measurement $y_{\xi\upsilon}$ from seed ξ and repetition $\upsilon \in \{0, 1\}$, we first form $\bar{y}_\xi = (y_{\xi 0} + y_{\xi 1})/2$. Means and sample SDs use one averaged observation per seed. Search welfare is conditional on an available certified policy. Within each seed, its mean uses only repetitions with an incumbent. The reported welfare mean and SD then use the seeds with at least one such observation. A missing incumbent is not assigned zero welfare. Table 18 reports the number of contributing seeds. Rel-summary with four clients contributes seven runs across four seeds.

The 32-candidate experiment reports the pooled reduction $100(1 - \sum C_{\mathrm{bank}} / \sum C_{\mathrm{base}})$ using the total costs from Appendix B.4. Matching includes profiles, order, limits, and definitive decisions. Unresolved candidates remain in the coverage denominator. Time to full welfare is recorded as an observed event or right-censored at the budget endpoint.

Outcome validation complements these aggregate measurements. All 40 reward-trap runs return a certified policy. Their saved incumbents pass source-level replay checks for legality, temporal feasibility, and equilibrium.

The nine controlled replay cases in Appendix F.1 are evaluated separately from the synthesis and timing aggregates. The suite has a shared 300-second cap. Each fresh source check, witness insertion, and replay allows at most 30 seconds within that cap. Each case creates a fresh bank containing its origin witness. Explicit-graph and specialized cycle checks provide reference judgments. All nine cases complete with agreement.

The six strong-until cases in Appendix F.2 form a separate deterministic suite with a shared 300-second cap. They compare source-level certification and bank replay with an explicit reference and a specialized temporal monitor. Each case starts with a fresh bank containing a deviation from its origin profile. All six cases complete and match their specified outcomes.

The source checker and explicit reference share the LTL evaluation module. The specialized cycle checks and strong-until monitor provide separate temporal acceptance tests for their respective goal forms. The independent acceptance checks agree with the source checker on these cases. Their test suite covers these goal forms; arbitrary nested LTL formulas remain outside its scope.

For the contention comparisons, stopping outcomes determine how much of each candidate domain was explored. The main comparison contains 480 runs. All save a certified incumbent within 600 seconds. The 120 construction-only runs finish after initialization. All 60 three-player searches exhaust the 512-profile domain. The other 300 searches reach a run or verification time boundary. In 57 of them, a verification allocation expires before the current candidate receives a decision. Their earlier certified incumbents remain available. No run reaches the candidate guard or memory limit. Initializations agree in all 120 paired blocks.

The earlier comparison contains 240 runs and four separate complete enumerations. All retain certified policies within their deadlines, with 80 searches reaching the deadline. Each batch averages seeds within games before computing means and sample SDs over ten games per size. The main comparison also uses 10,000 game-level bootstrap resamples with seed 20260919 for descriptive 95% intervals on paired mean W differences. Full-vector comparisons use unrounded lexicographic scores. Timing and quality statistics remain separate across batches.

D.6 Execution conditions

All experiments use the same Linux x86_64 server with an Intel Core i9-12900KF processor, 16 physical cores, 24 logical CPUs, Python 3.10.4, and Z3 4.13.3. Coupled contention uses eight workers pinned to distinct physical cores. Each matched group runs sequentially on one core with rotating method order. Every job has a 4 GiB address-space limit. Reward trap uses up to eight concurrent workers pinned to one logical CPU each with the same address-space limit. Their paired method order rotates with the seed and repetition. Numerical-library threads are limited to one.

In the fixed-stream, throughput, capacity, and ordering experiments, complete method runs execute serially with rotating order. Each timing comparison uses matched methods under the same workload and budget.

E Coupled Temporal Synthesis

The main contention comparison measures the gains from search after certified construction under the protocol in Appendix D. All four configurations start from the same candidates and selected scores, and every run returns a certified policy. Table 6 reports all preference components. The paired comparisons, time curves, and cost measurements below show the gains achieved, when they arise, and the computation required.

Table 6: Complete new-instance quality. Entries are mean ± sample SD over ten game means after averaging two seeds. The order is $(W, S, -R)$. Each row has 20/20 certified policies.

n	Method	W	S	$-R$
3	CTES	0.8000 ± 0.1721	0.4800 ± 0.1513	-0.4333 ± 0.1165
3	Without guidance	0.8000 ± 0.1721	0.4800 ± 0.1513	-0.4333 ± 0.1165
3	Without bank	0.8000 ± 0.1721	0.4800 ± 0.1513	-0.4333 ± 0.1165
3	Construction only	0.7667 ± 0.1610	0.5150 ± 0.1915	-0.4500 ± 0.1372
5	CTES	0.7000 ± 0.2160	0.4134 ± 0.1048	-0.3800 ± 0.0919
5	Without guidance	0.7000 ± 0.2160	0.4134 ± 0.1048	-0.3850 ± 0.0883
5	Without bank	0.7000 ± 0.2160	0.4134 ± 0.1048	-0.3800 ± 0.0919
5	Construction only	0.6600 ± 0.1838	0.4151 ± 0.1042	-0.3900 ± 0.0876
6	CTES	0.7000 ± 0.1315	0.3637 ± 0.0863	-0.3625 ± 0.0736
6	Without guidance	0.6833 ± 0.1291	0.3729 ± 0.0894	-0.3583 ± 0.0791
6	Without bank	0.7000 ± 0.1315	0.3637 ± 0.0863	-0.3625 ± 0.0736
6	Construction only	0.6667 ± 0.1303	0.3878 ± 0.0972	-0.3625 ± 0.0736
8	CTES	0.5688 ± 0.1396	0.3605 ± 0.0986	-0.3438 ± 0.0846
8	Without guidance	0.5688 ± 0.1396	0.3551 ± 0.1076	-0.3375 ± 0.0941
8	Without bank	0.5625 ± 0.1350	0.3605 ± 0.0986	-0.3438 ± 0.0846
8	Construction only	0.5625 ± 0.1350	0.3519 ± 0.1071	-0.3375 ± 0.0941
10	CTES	0.5450 ± 0.1571	0.3289 ± 0.1145	-0.3100 ± 0.0966
10	Without guidance	0.5450 ± 0.1571	0.3289 ± 0.1145	-0.3100 ± 0.0966
10	Without bank	0.5450 ± 0.1571	0.3289 ± 0.1145	-0.3100 ± 0.0966
10	Construction only	0.5450 ± 0.1571	0.3289 ± 0.1145	-0.3100 ± 0.0966
12	CTES	0.4875 ± 0.1390	0.3246 ± 0.1170	-0.2812 ± 0.0948
12	Without guidance	0.4875 ± 0.1390	0.3184 ± 0.1139	-0.2729 ± 0.0870
12	Without bank	0.4875 ± 0.1390	0.3246 ± 0.1170	-0.2812 ± 0.0948
12	Construction only	0.4875 ± 0.1390	0.3184 ± 0.1139	-0.2729 ± 0.0870

E.1 Gains beyond initialization

CTES strictly improves 22 of the 120 initial vectors shared by all configurations. Eleven gains increase W and eleven improve a later component at fixed W. Continued search produces these gains, and strict incumbent updates preserve or improve the initial vector.

At five players, CTES improves eight vectors beyond initialization, including four increases in W. At six players, five vectors improve and four increase W. At eight players, three vectors improve and one increases W. At ten players, every configuration returns its initial vector. At twelve players, four CTES runs improve S at fixed W. Mean S rises from 0.31840 to 0.32458 and mean R rises from 0.27292 to 0.28125. The preference order accepts this request-rate increase because service has higher priority.

Table 7 compares search with construction and isolates guidance by pairing guided and unguided search. CTES improves ten vectors relative to unguided search and loses two comparisons, with a mean W advantage only at six players. Every nonzero mean W difference has a descriptive game-level interval containing zero. The full lexicographic order determines whether changes in individual components constitute an improvement.

Table 7: Paired new-instance outcomes. Counts are wins/ties/losses across twenty paired runs. Intervals resample the ten independent games after averaging their two seeds. ΔW is the first method's welfare minus the second's. All comparisons have complete coverage.

n	Comparison	Full vector	W	Mean ΔW [95% interval]
3	CTES vs. Construction	2/18/0	2/18/0	0.0333 [0.0000, 0.1000]
3	Unguided vs. Construction	2/18/0	2/18/0	0.0333 [0.0000, 0.1000]
3	CTES vs. Unguided	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
3	CTES vs. No bank	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
5	CTES vs. Construction	8/12/0	4/16/0	0.0400 [0.0000, 0.0800]
5	Unguided vs. Construction	7/13/0	4/16/0	0.0400 [0.0000, 0.0800]
5	CTES vs. Unguided	1/19/0	0/20/0	0.0000 [0.0000, 0.0000]
5	CTES vs. No bank	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
6	CTES vs. Construction	5/15/0	4/16/0	0.0333 [0.0000, 0.0667]
6	Unguided vs. Construction	3/17/0	2/18/0	0.0167 [0.0000, 0.0417]
6	CTES vs. Unguided	3/16/1	2/18/0	0.0167 [0.0000, 0.0417]
6	CTES vs. No bank	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
8	CTES vs. Construction	3/17/0	1/19/0	0.0063 [0.0000, 0.0187]
8	Unguided vs. Construction	1/19/0	1/19/0	0.0063 [0.0000, 0.0187]
8	CTES vs. Unguided	2/17/1	0/20/0	0.0000 [0.0000, 0.0000]
8	CTES vs. No bank	1/19/0	1/19/0	0.0063 [0.0000, 0.0187]
10	CTES vs. Construction	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
10	Unguided vs. Construction	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
10	CTES vs. Unguided	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
10	CTES vs. No bank	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
12	CTES vs. Construction	4/16/0	0/20/0	0.0000 [0.0000, 0.0000]
12	Unguided vs. Construction	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]
12	CTES vs. Unguided	4/16/0	0/20/0	0.0000 [0.0000, 0.0000]
12	CTES vs. No bank	0/20/0	0/20/0	0.0000 [0.0000, 0.0000]

E.2 Quality through time

Figures 4 and 5 show when the endpoint gains arise by tracking welfare and service separately through time, including construction and certification. Curves average seeds within games and then average ten games, starting after every run has a certified incumbent. Construction-only policies remain available after their runs finish. An incumbent update can increase W while reducing S, so service curves need not be monotone.

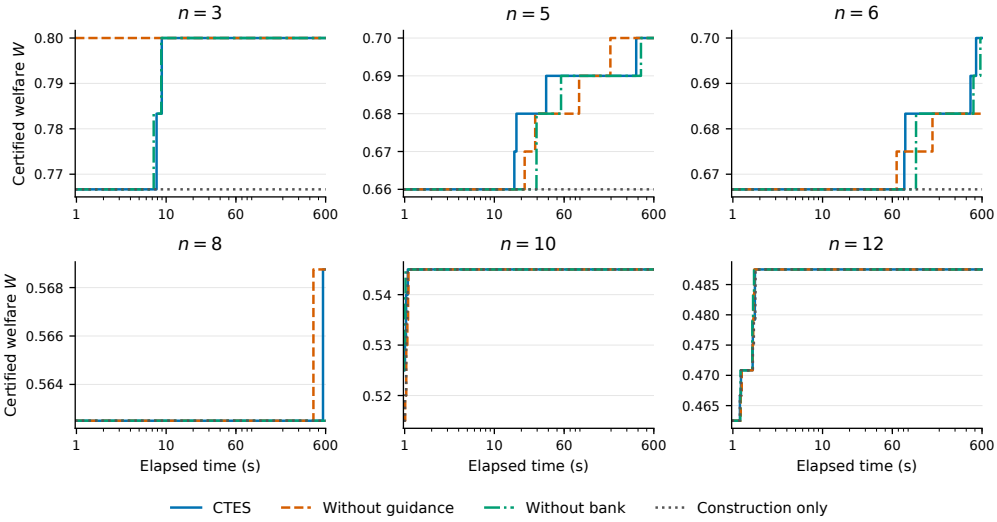

Figure 4: Certified goal satisfaction W over time on all six contention sizes. Curves use the complete incumbent traces and start at common certificate availability. The time axis is logarithmic.

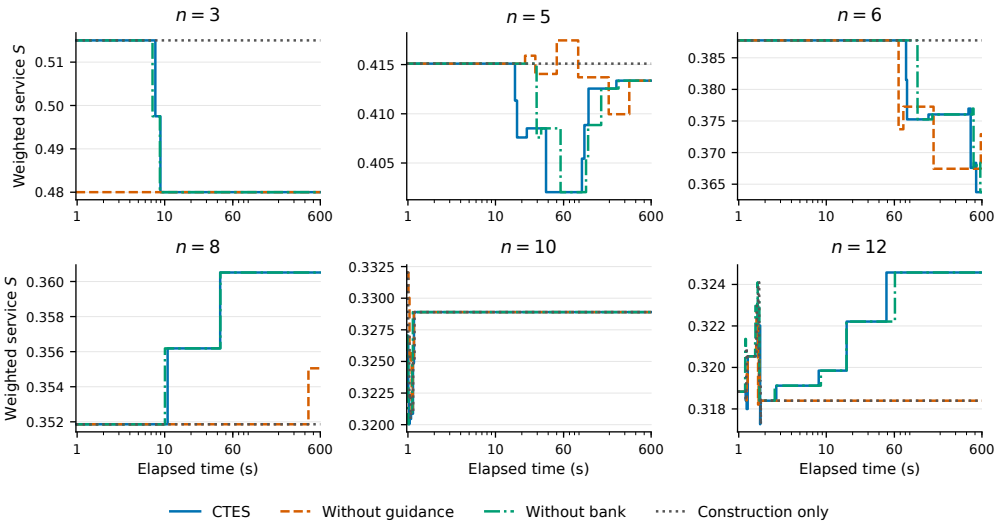

Figure 5: Certified weighted cycle service S over time on all six contention sizes. Curves use the complete incumbent traces and start at common certificate availability. The time axis is logarithmic.

E.3 Search cost and scale

Table 8 reports the computation behind the quality curves, including initialization, nominal feedback, bank operations, and fresh verification. Construction alone finishes in 0.071–1.198 seconds on average across sizes. All three-player searches examine 512 distinct profiles and return the same optimal vector within each matched domain. The mean of 512.4 definitive checks includes repeated certification when both constructors produce an identical profile.

At sizes five through twelve, the bank increases completed definitive checks by 7.51%, 25.34%, 28.84%, 21.98%, and 16.85%, respectively. Bank and no-bank final vectors tie in 119 runs. One eight-player run attains higher W with the bank. These counts compare search throughput under a common budget; the candidate trajectories can differ. Table 15 measures certification cost on identical candidates.

The twelve-player parameter domain contains $8^{12} = 68{,}719{,}476{,}736$ joint controllers. CTES completes 681.85 definitive checks on average and retains a certified policy in every run, after exploring a subset of the possible controller assignments. The three-player searches exhaust their domains and establish domain optimality. Across sizes, the generator's edge probabilities also change the expected graph degree.

E.4 Earlier cold-start controls

The main comparison starts from certified construction. The earlier controls instead start search from the idle controller and compare it with one weighted greedy candidate. They use separate instances 0–9 and seeds 30 and 31. Tables 9 and 10 report scores and costs for this initialization and instance setting.

Eight five-player game instances contain at least one strict improvement over greedy construction across their two seeds. Welfare itself improves in four runs across two instances. Relative to unguided search, four runs across three instances improve the full vector. Two of those runs improve welfare. These counts distinguish changes in the primary objective from changes in later preference components.

At three and four players, greedy construction has higher mean weighted service but lower mean welfare than Cold-start CTES. The lexicographic order prioritizes satisfied goals. Service and request rates decide comparisons only when welfare ties. The complete vectors therefore show which operational tradeoffs accompany the primary-objective gain.

Table 8: New-instance computation costs. Values are mean ± sample SD over ten game means. Checks are completed definitive certifications, including the two initial checks. Scores and checks use events within 600 seconds. Time measures run termination. RSS is measured worker peak resident memory in MiB.

n	Method	Time (s)	Checks	Peak RSS (MiB)
3	CTES	30.081 ± 4.756	512.4 ± 0.5	57.3 ± 0.1
3	Without guidance	33.363 ± 4.564	512.4 ± 0.5	57.6 ± 0.1
3	Without bank	30.849 ± 3.741	512.4 ± 0.5	56.1 ± 0.1
3	Construction only	0.071 ± 0.015	2.0 ± 0.0	54.5 ± 1.2
5	CTES	600.002 ± 0.000	4536.1 ± 1260.6	82.6 ± 7.5
5	Without guidance	600.003 ± 0.001	2747.4 ± 577.5	78.8 ± 4.6
5	Without bank	600.002 ± 0.001	4219.2 ± 1012.6	71.2 ± 3.8
5	Construction only	0.178 ± 0.055	2.0 ± 0.0	55.2 ± 0.2
6	CTES	600.003 ± 0.001	3631.0 ± 978.4	82.2 ± 6.2
6	Without guidance	600.003 ± 0.000	1732.6 ± 266.9	75.7 ± 4.0
6	Without bank	600.002 ± 0.000	2896.9 ± 604.5	70.9 ± 3.0
6	Construction only	0.236 ± 0.065	2.0 ± 0.0	55.4 ± 0.1
8	CTES	600.004 ± 0.002	2007.5 ± 849.3	78.8 ± 6.8
8	Without guidance	600.003 ± 0.000	902.5 ± 141.5	71.7 ± 2.3
8	Without bank	600.003 ± 0.001	1558.1 ± 467.1	69.9 ± 3.5
8	Construction only	0.481 ± 0.139	2.0 ± 0.0	55.8 ± 0.2
10	CTES	600.005 ± 0.003	1201.2 ± 608.7	78.6 ± 9.4
10	Without guidance	600.003 ± 0.001	578.2 ± 172.9	68.9 ± 4.0
10	Without bank	600.003 ± 0.001	984.7 ± 491.7	70.6 ± 6.7
10	Construction only	0.790 ± 0.270	2.0 ± 0.0	56.1 ± 0.4
12	CTES	600.003 ± 0.001	681.9 ± 243.4	73.7 ± 5.1
12	Without guidance	600.003 ± 0.004	389.4 ± 69.5	68.1 ± 2.4
12	Without bank	600.003 ± 0.001	583.5 ± 171.3	67.8 ± 3.3
12	Construction only	1.198 ± 0.412	2.0 ± 0.0	56.8 ± 0.6

Table 9: Earlier cold-start scores. Statistics use ten instance means after averaging two search seeds. The preference vector is ordered lexicographically as $(W, S, -R)$. All configurations return certified policies in 20/20 runs per size.

n	Method	Welfare W	Weighted service S	Negative requests $-R$
3	Cold-start CTES	0.733 ± 0.141	0.439 ± 0.141	-0.400 ± 0.117
3	Without guidance	0.733 ± 0.141	0.439 ± 0.141	-0.400 ± 0.117
3	Without bank	0.733 ± 0.141	0.439 ± 0.141	-0.400 ± 0.117
3	Constructive greedy	0.700 ± 0.105	0.456 ± 0.158	-0.417 ± 0.142
4	Cold-start CTES	0.725 ± 0.079	0.414 ± 0.089	-0.400 ± 0.079
4	Without guidance	0.725 ± 0.079	0.414 ± 0.089	-0.400 ± 0.079
4	Without bank	0.725 ± 0.079	0.414 ± 0.089	-0.400 ± 0.079
4	Constructive greedy	0.675 ± 0.121	0.418 ± 0.090	-0.425 ± 0.087
5	Cold-start CTES	0.620 ± 0.114	0.405 ± 0.099	-0.420 ± 0.123
5	Without guidance	0.600 ± 0.094	0.411 ± 0.084	-0.405 ± 0.121
5	Without bank	0.620 ± 0.114	0.405 ± 0.099	-0.420 ± 0.123
5	Constructive greedy	0.570 ± 0.116	0.382 ± 0.071	-0.355 ± 0.090

For five-player games, mean welfare at 60 seconds is 0.500 for Cold-start CTES and 0.560 without guidance. At 300 seconds these values are 0.620 and 0.600. Greedy construction gives 0.570 and finishes in about 0.109 seconds. The two search variants reverse their welfare ranking between 60 and 300 seconds.

Table 10 separates the bank's online effect from final quality. Cold-start CTES and the no-bank control return identical final vectors in every paired run. With five players, bank reuse increases completed candidate checks by 6.13% under the common deadline. With four players, enabling the bank increases mean total time by 5.33%. This total includes feedback, admission, replay, insertion, and fresh verification across the candidates reached by each run.

Table 10: Earlier cold-start search cost. Time and candidate counts are mean ± SD across ten instance averages. Hits are totals over twenty runs. Runtime includes construction, feedback, bank operations, and exact certification. Five-player search configurations reach the 600-second deadline.

n	Method	Time (s)	Candidates	Bank hits
3	Cold-start CTES	33.29 ± 4.02	512.0 ± 0.0	3185
3	Without guidance	35.24 ± 2.89	512.0 ± 0.0	2653
3	Without bank	33.64 ± 3.12	512.0 ± 0.0	0
3	Constructive greedy	0.04 ± 0.01	1.0 ± 0.0	0
4	Cold-start CTES	554.29 ± 56.72	4058.8 ± 112.9	14743
4	Without guidance	582.84 ± 33.15	3959.3 ± 167.8	5679
4	Without bank	526.25 ± 54.32	4088.8 ± 22.9	0
4	Constructive greedy	0.06 ± 0.02	1.0 ± 0.0	0
5	Cold-start CTES	600.00 ± 0.00	4039.9 ± 876.8	26729
5	Without guidance	600.00 ± 0.00	2545.8 ± 485.1	3819
5	Without bank	600.00 ± 0.00	3806.6 ± 705.6	0
5	Constructive greedy	0.11 ± 0.02	1.0 ± 0.0	0

Complete enumeration yields $(1, 0.5, -0.5)$ for both two-player reference instances. For three players the reference vectors are $(2/3, 0.40625, -1/3)$ and $(2/3, 0.38235294, -1/3)$. Cold-start CTES, unguided search, and the no-bank control attain these vectors on the corresponding three-player instances. Each enumeration checks all 64 or 512 joint controllers. The reference values establish optimality within those finite controller domains.

All three search variants examine the full 512-profile domain in every three-player run and return identical final vectors. Their timings and bank-hit counts differ. Five-player runs reach the 600-second limit with only a subset of the candidate domain evaluated.

F Strategic and Temporal Certification Checks

The following controls examine the decisions underlying the certified outputs. Reward trap separates external quality from individual incentives, replay controls test whether prior rejection evidence remains applicable, and strong-until cases distinguish eventual fulfillment from indefinite waiting.

In reward trap, agents are indexed by $i = 0, \ldots, n-1$. Agent i controls a Boolean variable on_i and can choose either value initially and at every update. Its individual goal is $\mathsf{G}\neg on_i$ and its system requirement is true. The external reward counts occurrences of on_0 on the canonical lasso. Any player whose goal fails can choose false initially and after every history. This unilateral policy satisfies its goal independently of the opponents. Hence every equilibrium satisfies all goals and has reward zero.

Table 11 shows how certification filters candidates during preference refinement. Candidates with profitable deviations are rejected, while the feasible incumbent from the initial phase remains available. The reported counts include repeated candidate attempts.

Both variants make 17 attempts per run and retain zero-reward equilibria. Their identical final quality follows from the equilibrium condition derived above. At three players, mean runtime is 1.608 seconds for CTES and 0.888 seconds without guidance.

Table 11: All reward-trap configurations. Certified gives runs returning a certified policy. Attempts and rejections are per-run counts and do not vary across the ten runs of a setting. Runtime is the complete run in seconds.

n	Method	Certified	Reward	Attempts	Rejections	Runtime
2	CTES	10/10	0	17	16	0.393 ± 0.024
2	Unguided	10/10	0	17	15	0.387 ± 0.026
3	CTES	10/10	0	17	16	1.608 ± 0.037
3	Unguided	10/10	0	17	16	0.888 ± 0.075

F.1 Controlled checks of deviation replay

Reward trap tests the need for strategic certification. Nine controlled cases now test whether an old deviation still justifies rejection after a policy change. Two players control Boolean atoms $done$ and flag. Each can choose either value at every update. Their goals are $\varphi_1 = \mathsf{GF}\, done \wedge \mathsf{G}\neg \mathit{flag}$ and $\varphi_2 = \mathsf{G}\neg done$. The default initialization fixes both atoms to false. The initial-action case also permits flag to be true. Every origin profile keeps both atoms false forever. Player 1 loses nominally but can satisfy its goal by making $done$ recur while the fixed opponent keeps flag false. We store a deviation found by exact certification and evaluate it against each modified profile. Each case retains the same game between the origin and target profiles.

Table 12 reports profitable reuse and the equilibrium status of each current profile. Fresh source-level certification agrees with an explicit finite-controller reference in all nine cases. These two checks share the LTL evaluation engine. A separate check evaluates safety and accepting cycles directly on the explicit deviation graph, tracking the first transition for the next-time cases. It agrees in every case, providing a separate temporal acceptance test for these specific semantic conditions.

Table 12: Controlled replay outcomes. Profitable reuse requires both current structural compatibility and strict payoff improvement. An unsuccessful replay is followed by fresh certification. All nine equilibrium judgments agree with the explicit reference and the separate cycle check.

Change from the origin profile	Profitable reuse	Current equilibrium
Unchanged profile	Yes	No
Output change at an unreachable opponent memory	Yes	No
Opponent remembers an occurrence of $done$	No	Yes
Player 1 already satisfies its goal	No	Yes
Opponent initially sets flag true	No	Yes
Equivalent opponent behavior with different memory encoding	No	No
Next-augmented goal, unchanged profile	Yes	No
Next-augmented goal, player 1 already satisfies its goal	No	Yes
Next-augmented goal, remembered opponent response	No	Yes

Current opponent memory. The remembered-response controller starts in memory zero and keeps flag false there. Observing $done$ moves it permanently to memory one. In that memory it sets $\mathit{flag}(t+1) = done(t)$. Nominal play remains unchanged because player 1 stays idle. A deviation that makes $done$ recur now forces flag to occur and violates $\mathsf{G}\neg \mathit{flag}$. The old witness no longer follows the current memory transitions. Fresh certification establishes equilibrium. By comparison, adding an unreachable memory with output $\mathit{flag} = \mathit{true}$ preserves the stored behavior and allows profitable reuse.

Current payoff and initial behavior. When player 1 sets $done$ true at every update, its nominal payoff becomes one. The old deviation still satisfies its goal against the unchanged opponent. It provides no strict improvement, so the bank does not reject this equilibrium. The initial-action case instead permits either initial value of flag in the common source game. The origin chooses false and the target chooses true. The target violates player 1's safety goal at initialization. Subsequent unilateral choices cannot repair that violation.

Memory representation and temporal coverage. Encoding the constant-false opponent with two memories initialized at one preserves its behavior. The stored witness records memory zero and therefore cannot replay directly. Fresh verification still finds a profitable deviation. The final three cases replace φ_1 by $\varphi_1 \wedge \mathsf{X}done \wedge (\neg \mathit{flag}\ \mathsf{U}\ done)$. They reproduce the unchanged, nominal-winner, and remembered-response outcomes. Here $\mathsf{G}\neg \mathit{flag}$ and $\mathsf{X}done$ already imply the until conjunct. These cases exercise next-time behavior within a mixed temporal goal whose until conjunct is redundant.

F.2 Nonredundant strong-until obligations

The preceding mixed goals contain a redundant until conjunct. Six additional cases make satisfaction depend on strong until, so indefinite waiting and eventual fulfillment must be distinguished. Two players freely choose Boolean atoms $done$ and *flag* at each update. Initially $done$ is false and player 2 may choose either value of *flag*. Player 2 has goal true. Player 1 has goal *flag* U $done$ in the first three cases and $\mathsf{X}\neg done \wedge (flag\ \mathsf{U}\ done)$ in the remaining three. The second goal requires $done$ to remain false at the first update before eventual completion. The two-memory controller in the delayed-completion case makes $done$ true at the second update.

Each origin profile keeps *flag* true and $done$ false forever. Player 1 loses because strong until requires an eventual occurrence of $done$. Exact certification supplies a profitable deviation for the bank. We then test the stored deviation against a target profile in the same game. Source-level certification and an explicit finite-controller reference evaluate the target independently of the replay decision. A separate temporal monitor checks that *flag* holds before the first $done$ and enforces $\mathsf{X}\neg done$ when present. It requires a reachable cycle after fulfillment. The monitor shares transition-graph construction with the explicit reference and uses its own temporal acceptance test.

Table 13: Strong-until checks. All six certification judgments agree across the source checker, explicit reference, and temporal monitor. Bank replay also matches the specified outcome in every case. The first update is position one after initialization.

Target behavior	Profitable reuse	Current equilibrium
Goal *flag* U $done$		
flag remains true; $done$ never occurs	Yes	No
flag remains true; $done$ starts at the first update	No	Yes
flag is initially false; $done$ never occurs	No	Yes
Goal $\mathsf{X}\neg done \wedge (flag\ \mathsf{U}\ done)$		
flag remains true; $done$ never occurs	Yes	No
flag remains true; $done$ starts at the second update	No	Yes
flag becomes false at the first update; $done$ never occurs	No	Yes

Table 13 separates three outcomes for each goal. Perpetual waiting admits a profitable deviation and successful reuse. A target that already completes the obligation has payoff one, so the stored deviation offers no strict improvement. When *flag* is false initially, player 1 cannot repair the violated obligation through later choices of $done$. This remains true when the opponent restores *flag* at the first update. For the delayed goal, losing *flag* at the first update also prevents completion because $\mathsf{X}\neg done$ forbids $done$ at that step. Both cases are equilibria with an unsatisfied player. Together these checks distinguish eventual completion from indefinite waiting and goal failure from the existence of a profitable deviation.

G Verification Reuse Results

Fixed candidate streams measure the cost of evidence reuse through total runtime and replay activity. The results below quantify net savings and the work that produces them, followed by a concrete witness whose rejection transfers between profiles.

G.1 Fixed-stream cost and replay activity

Table 14 summarizes total cost, replay overhead, and hits for the matched Exact configurations.

Table 15 compares complete stream costs and coverage for each backend. Figure 6 gives the paired Exact cost reductions at every size and seed. Their common candidate sequences isolate certification work from changes in the search trajectory.

Table 16 reports replay activity and internal timings for every resource backend. A hit is a candidate rejection supported by successful replay; an attempt tests one stored deviation. Exact + bank averages 12.4, 10.0, and 10.8 hits per stream across the three sizes, from 59.6, 34.6, and 51.4 attempts. Aggregate savings therefore include the cost of unsuccessful replay. Because internal timers

Table 14: Matched 32-candidate streams. Total time includes bank costs (seconds, mean ± seed SD, five seeds after averaging two repetitions). Replay times and hits are means. Diagnostics: Appendix G.1.

Clients	Exact total	Exact + bank total	Replay time	Hits per stream
3	6.01 ± 0.76	5.19 ± 1.28	0.832	12.4
4	11.50 ± 2.73	9.99 ± 3.71	0.784	10.0
5	19.85 ± 3.78	16.23 ± 5.63	1.133	10.8

Table 15: Fixed-stream certification controls. Backends share each seeded 32-candidate stream. Times are mean ± seed SD in seconds and include bank operations. Coverage gives definitive/-planned decisions under a 340 s run cap.

Method	$n = 3$: time	Def.	$n = 4$: time	Def.	$n = 5$: time	Def.
Exact	6.01 ± 0.76	320/320	11.50 ± 2.73	320/320	19.85 ± 3.78	320/320
Exact + bank	5.19 ± 1.28	320/320	9.99 ± 3.71	320/320	16.23 ± 5.63	320/320
Chain	5.49 ± 0.95	320/320	10.11 ± 4.30	320/320	16.05 ± 4.74	320/320
Chain + bank	4.02 ± 0.35	320/320	7.16 ± 1.74	320/320	10.50 ± 1.03	320/320
Rel-summary	60.80 ± 8.19	320/320	216.90 ± 60.76	204/320	280.19 ± 35.44	44/320

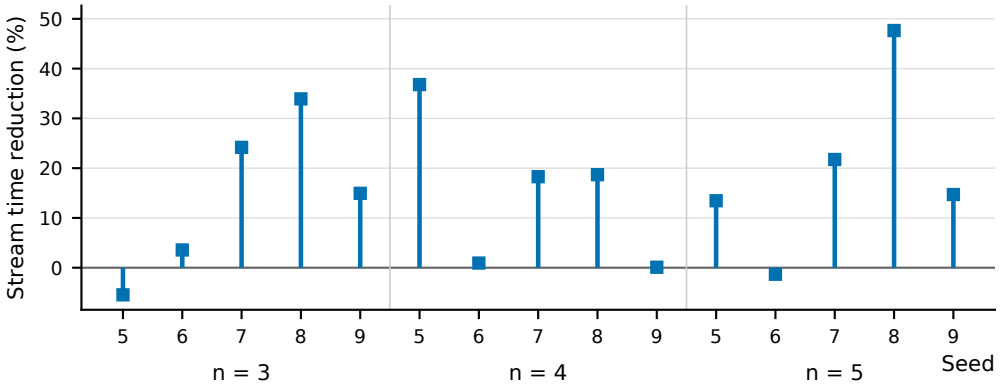

Figure 6: Paired stream-time reduction for every resource size–seed unit. Two repetitions are averaged before computing $100(1 - C_{\text{bank}}/C_{\text{base}})$ with Exact as the baseline. Positive values indicate lower cost with the bank and negative values indicate higher cost. Both arms have matching complete decisions on all 960 candidate evaluations. The pooled reduction is 15.95%, computed as a ratio of total times rather than the mean of these percentages.

can overlap, net cost reduction is determined by the end-to-end totals in Table 15. Equation (19) identifies the work exchanged by reuse.

Table 16: Mean recorded timing diagnostics (seconds) per fixed resource stream, averaging two repetitions within each of five seeds. Internal timers overlap; total costs are reported separately. Backend is native exact/probe/fallback time; discovery includes warming; learn is witness insertion. Hits/attempts count profitable bank hits and individual replay attempts.

n	Method	Backend	Replay	Learn	Discover	Template	Hits/attempts
3	Exact	5.987	0.000	0.000	0.000	0.000	n/a
3	Exact + bank	4.093	0.832	0.140	0.000	0.000	12.4/59.6
3	Chain	2.780	0.000	0.000	0.345	2.182	n/a
3	Chain + bank	1.032	0.842	0.140	0.352	1.483	12.4/59.6
3	Rel-summary	60.772	0.000	0.000	0.000	0.000	n/a
4	Exact	11.464	0.000	0.000	0.000	0.000	n/a
4	Exact + bank	8.952	0.784	0.111	0.000	0.000	10.0/34.6
4	Chain	4.156	0.000	0.000	0.977	4.750	n/a
4	Chain + bank	1.800	0.768	0.108	0.982	3.279	10.0/34.6
4	Rel-summary	216.865	0.000	0.000	0.000	0.000	n/a
5	Exact	19.822	0.000	0.000	0.000	0.000	n/a
5	Exact + bank	14.702	1.133	0.189	0.000	0.000	10.8/51.4
5	Chain	7.964	0.000	0.000	0.890	6.891	n/a
5	Chain + bank	2.979	1.132	0.188	0.899	5.031	10.8/51.4
5	Rel-summary	280.158	0.000	0.000	0.000	0.000	n/a

G.2 A CONCRETE REPLAY BETWEEN RESOURCE POLICIES

One Exact + bank replay makes the aggregate hit counts concrete. This run has three clients and uses seed 5 with repetition 0. A profitable deviation of client 0 enters the bank at stream index 7. The same deviation replays successfully at index 24. Each client's parameter triple records its slot request and the monitor's remembered and immediate responses. The origin vector is $(0,0,0,\ 0,1,0,\ 0,1,0)$. The target vector is $(0,0,0,\ 0,1,0,\ 0,1,1)$. Only monitor 2's immediate-response output changes. The two candidates are separated by seventeen positions on the same stream. All initial strategic actions are empty. Every recorded opponent memory is zero.

Table 17: Actual stored deviation of client 0. The initial state is row 0. Row 6 returns to row 1. The prefix has length one and the loop has length six. All unlisted atoms are false and every opponent's recorded action is empty.

Position	True state atoms	Client 0's next action (true atoms)
0	$token_0$	$request_0$
1	$request_0$, $token_1$	$request_0$
2	$request_0$, $token_2$	$request_0$
3	$request_0$, $token_0$	$request_0$, $service_0$
4	$request_0$, $service_0$, $token_1$	$request_0$
5	$request_0$, $token_2$	$\varnothing$
6	$token_0$	$request_0$

Table 17 contains recurring $service_0$ without $penalty_0$. $service_2$ never holds along the trace. Toggling monitor 2's immediate response to $service_2$ therefore preserves all its recorded outputs. Its memory remains zero throughout the lasso. The other opponents are unchanged. Client 0 remains nominally idle and fails its recurring-service goal. The witness thus satisfies structural replay and the current $0 \to 1$ payoff comparison.

H FIXED-BUDGET RESOURCE SEARCH AND ADDITIONAL CONTROLS

Fixed streams quantify certification cost on identical inputs. Online search measures certification counts and incumbent quality under a shared deadline. The final controls return to fixed streams to isolate witness capacity and candidate order.

H.1 Search throughput

Table 18 expands the throughput comparison with definitive-check counts, per-seed variability, and incumbent availability for every backend. Counts sum certification events across runs, so the same policy found in different runs contributes more than once. Figure 7 shows their progress over time.

Both Exact configurations produce an incumbent in every run. With four clients, Rel-summary produces incumbents in seven of ten runs across four of five seeds; with five clients, it produces none. Its conditional welfare therefore uses a different denominator at four clients and is undefined at five.

Table 18: Search under 60 s per run. Def. counts definitive candidate evaluations and NE counts evaluations yielding a certified equilibrium. Counts show per-run mean ± seed SD with ten-run totals in brackets. Count SDs use five seed means after averaging two timing repetitions. Welfare is conditional on incumbent availability and uses only observed repetitions and seeds. Obs. gives observed seeds and Full gives runs attaining welfare one.

n	Method	Def.	NE	Best welfare	Obs.	Full
3	Exact	336.8 ± 7.2 [3368]	74.3 ± 7.7 [743]	1.000 ± 0.000	5/5	10/10
3	Exact + bank	358.8 ± 23.9 [3588]	78.2 ± 5.4 [782]	1.000 ± 0.000	5/5	10/10
3	Rel-summary	45.3 ± 15.9 [453]	14.6 ± 2.4 [146]	0.767 ± 0.253	5/5	4/10
4	Exact	241.6 ± 7.4 [2416]	35.2 ± 3.6 [352]	1.000 ± 0.000	5/5	10/10
4	Exact + bank	266.5 ± 15.4 [2665]	36.7 ± 4.5 [367]	1.000 ± 0.000	5/5	10/10
4	Rel-summary	5.9 ± 2.3 [59]	1.2 ± 1.0 [12]	0.547 ± 0.060	4/5	0/10
5	Exact	151.9 ± 17.9 [1519]	25.8 ± 5.3 [258]	0.940 ± 0.089	5/5	6/10
5	Exact + bank	185.0 ± 25.8 [1850]	29.1 ± 4.6 [291]	0.960 ± 0.089	5/5	8/10
5	Rel-summary	2.1 ± 1.1 [21]	0.0 ± 0.0 [0]	n/a	0/5	0/10

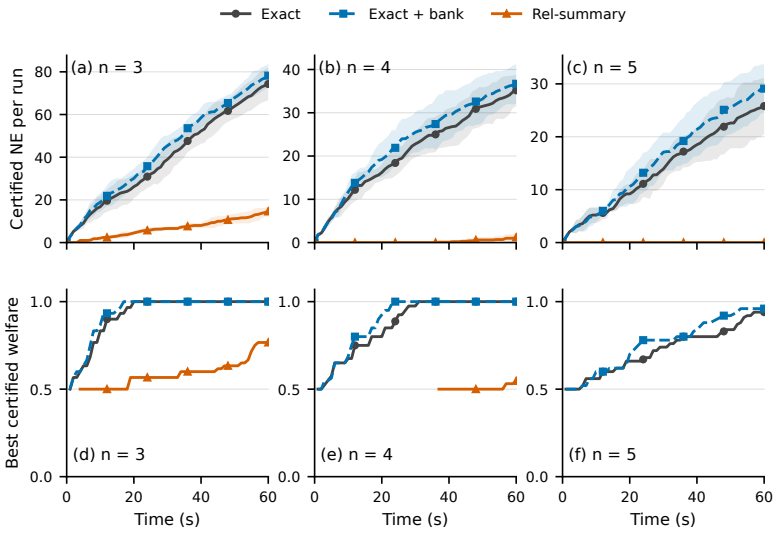

Figure 7: Search progress across three resource sizes. The top row reports equilibrium-certification counts per run with one seed SD. The bottom row reports best certified welfare conditional on availability. Table 18 gives endpoint counts and availability.

Table 19 resolves the pooled search gains into seed-level differences. The bank increases the mean endpoint count in 13 of 15 size–seed units and ties in the other two. Each size retains its five paired seeds.

Table 19: Paired endpoint count differences for Exact + bank minus Exact at 60 seconds. Each cell first averages the two timing repetitions for that size and seed. Positive values mean more certified NE per run; zero means a tie.

n	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9
3	+8.0	+7.5	+1.0	+1.5	+1.5
4	0.0	0.0	+3.0	+2.5	+2.0
5	+2.5	+2.0	+3.0	+6.0	+3.0

Table 20: Capacity and order controls use exactly seeds 5 and 6 at $n \in \{3, 5\}$ (two repetitions each), including the matched fixed-stream controls. Times are mean $\pm$ seed SD; coverage is definitive/-planned. Every run has 32 candidates and a 340 s cap.

n	Method	Capacity	Order	Time (s)	Coverage
3	Exact	n/a	original	6.34 ± 0.42	100%
3	Exact + bank	1	original	6.56 ± 0.19	100%
3	Exact + bank	4	original	6.42 ± 0.08	100%
3	Exact + bank	8	original	6.39 ± 0.02	100%
3	Exact + bank	16	original	6.40 ± 0.04	100%
3	Exact	n/a	shuffled	6.25 ± 0.39	100%
3	Exact + bank	8	shuffled	6.48 ± 0.06	100%
5	Exact	n/a	original	23.39 ± 0.81	100%
5	Exact + bank	1	original	22.18 ± 0.88	100%
5	Exact + bank	4	original	21.92 ± 1.71	100%
5	Exact + bank	8	original	21.93 ± 1.68	100%
5	Exact + bank	16	original	22.11 ± 1.62	100%
5	Exact	n/a	shuffled	23.62 ± 0.59	100%
5	Exact + bank	8	shuffled	22.18 ± 1.85	100%

H.2 Capacity and ordering

Capacity limits which witnesses remain available, while candidate order changes when they can be reused. Table 20 varies these factors on matched fixed streams using the same two seeds. Capacity one is slower than capacity four in both tested sizes. Increasing capacity beyond four gives little further change. With three clients, the bank is slightly slower than Exact on these two seeds under either ordering. With five clients, reuse lowers mean cost under both orders.

Replay activity helps explain the small differences beyond capacity four. Capacities 4, 8, and 16 each average 6.0 hits and 41.5 attempts per stream with three clients. They each average 3.0 hits and 24.5 attempts with five clients. Capacity one instead gives 4.5 hits and 25.0 attempts at three clients, and 2.5 hits and 13.5 attempts at five.

Ordering affects the number of replay attempts even when hits are unchanged. Shuffling preserves the three-client hit count at 6.0 but increases attempts from 41.5 to 50.5. With five clients, both orders give 3.0 hits and 24.5 attempts.